

Bus Manager (Stand-Alone)

# Implementation Guide

For Bus Manager 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:	<a href="mailto:info@dspace.de">info@dspace.de</a>
Web:	<a href="http://www.dspace.com">http://www.dspace.com</a>

## 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](mailto: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.

© 2016 - 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 Guide	9
------------------	---

Safety Precautions	13
--------------------	----

General Warning .....	13
-----------------------	----

Introduction to the Bus Manager	15
---------------------------------	----

Overview of the Bus Manager.....	15
----------------------------------	----

Introduction to Bus Communication Defined in Communication Matrices.....	19
---	----

Use Scenarios for Configuring Bus Communication with the Bus Manager.....	20
--	----

Basics on Licenses for Working with the Bus Manager (Stand-Alone).....	24
--	----

Notes for Starting the Bus Manager (Stand-Alone).....	25
---	----

Elements of the Bus Manager.....	27
----------------------------------	----

Typical Workflows for Using the Bus Manager and Generating Bus Simulation Containers	31
---	----

Workflow for Configuring Bus Communication and Generating Bus Simulation Containers.....	31
---	----

Generating Bus Simulation Containers with a Behavior Model.....	33
---	----

Generating Bus Simulation Containers Without a Behavior Model.....	34
--	----

Aspects of Supported Bus Communication	37
--	----

Supported PDU Types and Signal Data Types.....	37
--	----

Aspects of Supported AUTOSAR Features.....	39
--	----

Aspects of Supported CAN Bus Features.....	47
--	----

Aspects of Supported LIN Bus Features.....	52
--	----

Signal Conversion by the Bus Manager.....	54
---	----

Basics on the Bus Manager	57
---------------------------	----

Working with Communication Matrices.....	58
--	----

Basics on Bus Configurations.....	63
-----------------------------------	----

Bus Configuration Tables.....	66
-------------------------------	----

Bus Configuration Function Block.....	71
Assigning Communication Matrix Elements to Bus Configurations.....	75
Removing Communication Matrix Elements from Bus Configurations.....	83
Working with User Code.....	84
Implementing Secure Onboard Communication in Executable Applications.....	87
Implementing Global Time Synchronization in Executable Applications.....	88
Basics on Bus Access Requests.....	91
Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager.....	94
Accessing Bus Manager Features via ConfigurationDesk's Automation Interface.....	97
<b>Handling Bus Manager-Related Conflicts</b>	<b>99</b>
Basics on Bus Manager-Related Conflicts.....	99
Resolving Bus Manager-Related Conflicts.....	103
<b>Specifying CAN Frame Captures and Gateways</b>	<b>107</b>
Capturing CAN Frames.....	107
Specifying CAN Gateways.....	109
Specifying Filters for Frame Captures and Frame Gateways.....	111
<b>Working with Bus Configuration Features</b>	<b>115</b>
Basics on Working with Bus Configuration Features.....	116
Basics on Bus Configuration Features.....	116
Configuring Function Ports for Bus Configuration Features.....	120
Accessing Function Ports with Enabled Test Automation Support in Variable Description Files.....	124
Triggering the Transmission of PDUs and Frames via User-Defined Triggers.....	128
Manipulating Bus Communication via Bus Configuration Features.....	131
Basics on Bus Manipulation Features.....	131
Recalculating End-to-End Protection and Authentication Information .....	135
Execution Sequence of Bus Manipulation Features.....	137
Bus Configuration Features Available for ISignals.....	139
Working with ISignal Values.....	139
Working with Counter Signals.....	141
Overwriting ISignal Values.....	146
Adding Offset Values to ISignal Values.....	149

Bus Configuration Features Available for ISignal Groups.....	152
Observing the Status of Received End-to-End-Protected ISignal Groups.....	152
Bus Configuration Features Available for PDUs.....	155
Enabling and Disabling the Transmission of PDUs.....	155
Accessing the Payload of PDUs in Raw Data Format.....	157
Controlling the Cyclic Timing of CAN PDUs.....	159
Specifying User-Defined Triggers for Transmitting PDUs.....	162
Accessing the Payload Length of CAN PDUs.....	165
Observing the Status of Received PDUs.....	167
Applying User Code to PDUs.....	170
Verifying the Authentication Information of Received Secured IPDUs.....	175
Invalidate the Authenticator of Secured IPDUs.....	177
Overwriting the Freshness Value of Secured IPDUs.....	180
Triggering the Execution of Functions in a Behavior Model via RX Interrupts.....	183
Bus Configuration Features Available for Frames.....	190
Accessing CAN Frame Settings.....	190
Manipulating the Payload Length of CAN Frames.....	197
Suspending the Transmission of Frames.....	200
Bus Configuration Features Available for Communication Controllers.....	203
Enabling and Disabling Communication Controllers.....	203
Sending Wake-Up Signals on a LIN Bus.....	205
Working with LIN Schedule Tables.....	207
Enabling and Disabling J1939 Network Management.....	211
Bus Configuration Features Available for Global Time Domains.....	214
Controlling the Timing of Time Synchronization.....	214
Accessing the Time Base Data of Time Masters and Time Slaves.....	216
Accessing Validity Checks for Time Synchronization Messages.....	218
Bus Configuration Features Available for Frame Captures and Frame Gateways.....	222
Accessing the Data of Captured Frames.....	222
Specifying the Direction of CAN Frame Gateways.....	226
Controlling Filters of Frame Captures and Frame Gateways.....	227
Bus Configuration Features Available for Bus Configurations.....	230
Enabling and Disabling Bus Configurations.....	230

<b>Implementing Bus Communication in the Signal Chain Using the Bus Manager</b>	<b>233</b>
How to Add a Communication Matrix to a ConfigurationDesk Application.....	233
How to Add a Bus Configuration to a ConfigurationDesk Application.....	236
How to Assign Communication Matrix Elements to a Bus Configuration.....	238
How to Enable Model Access for Function Ports.....	242
How to Add Behavior Models to a ConfigurationDesk Application.....	244
Example of Selecting Multiple Similar Communication Matrix Elements at Once.....	247
Example of Mapping Model Ports to Bus Configuration Ports via Tables.....	249
 <b>Generating Bus Simulation Containers</b>	 <b>253</b>
Basics on Bus Simulation Containers.....	253
Port Interface of Bus Simulation Containers.....	256
How to Generate Bus Simulation Containers.....	259
 <b>Working Without Behavior Models</b>	 <b>263</b>
Basics on Working Without Behavior Models.....	263
How to Create Application Processes that Provide Default Tasks.....	264
How to Manually Assign Bus Configurations to Application Processes.....	265
 <b>Modifying Communication Matrices</b>	 <b>267</b>
Basics on Modifying Communication Matrices.....	267
Adding ISignal PDUs to CAN Channels.....	271
Adding ISignals to CAN PDUs.....	272
Adding Cyclic Timing Elements to Basic PDUs.....	274
Configurable Settings of Communication Clusters.....	276
Configurable Settings of Frames.....	277
Configurable Settings of PDUs.....	281
Configurable Settings of ISignals.....	284

Replacing Assigned Communication Matrices	291
Basics on Replacing Assigned Communication Matrices.....	291
How to Replace Assigned Communication Matrices.....	294
Working with the Bus Manager Demo	297
Introduction to the Bus Manager Demo.....	297
Overview of the Example Used in the Bus Manager Demo.....	298
How to Run the Bus Manager Demo Script via the Bus Manager.....	301
Running the Bus Manager Demo Script via an External Interpreter.....	304
Steps of the Bus Manager Demo Script.....	306
Example of Experimenting in ControlDesk.....	312
Limitations for Using the Bus Manager	319
Limitations for Communication Matrices and Communication Standards.....	319
Limitations for CAN Communication.....	323
Limitations for LIN Communication.....	325
Limitations for Bus Configuration Handling.....	328
Limitations for Bus Simulation Containers.....	331
Appendix	335
Mapping of Bus Manager PDU Types to Communication Standard PDU Types.....	335
Optimized Set of Variables for Bus Configuration Function Ports.....	338
Resolving the Bus Configuration Conflict: No Valid Application Process Assigned.....	341
How to Assign Behavior Models to Separate Application Processes.....	344
ConfigurationDesk Glossary	347
Index	373



# About This Guide

---

**Content**

Introduces you to the Bus Manager (stand-alone) and shows you how to configure bus communication. This includes:

- Implementing bus communication by using communication matrices and bus configurations.
- Specifying the model interface to exchange bus signals with a behavior model.
- Generating bus simulation containers.

**Note**

Two variants of the Bus Manager are available, the Bus Manager in ConfigurationDesk and the Bus Manager (stand-alone). For information on working with the Bus Manager in ConfigurationDesk, refer to [ConfigurationDesk Bus Manager Implementation Guide](#).

---

**Target group**

This guide is primarily intended for engineers who configure bus communication for simulation, inspection, and/or manipulation purposes. The target group performs some or all of the following tasks:

- Set up bus communication based on communication matrices.
- Specify a model interface to exchange bus signals between the Bus Manager and a behavior model modeled in MATLAB®/Simulink®.
- Generate bus simulation containers that include the configured bus communication.

---

**Required knowledge**

Basic knowledge in working with the used bus protocols 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.
	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](http://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.



# Safety Precautions

## General Warning

### Danger potential

*Using dSPACE software can be dangerous. You must observe the following safety instructions and the relevant instructions in the user documentation.*

#### **WARNING**

**Improper or negligent use can result in serious personal injury and/or property damage.**

Using the ConfigurationDesk software can have a direct effect on dSPACE systems and technical (electrical, hydraulic, mechanical) systems connected to them.

- **Only persons who are qualified to use dSPACE software, and who have been informed of the above dangers and possible consequences, are permitted to use it.**
- All applications where malfunctions or misoperation involve the danger of injury or death must be examined for potential hazards by the user, who must if necessary take additional measures for protection (for example, an emergency off switch).

### Liability

It is your responsibility to adhere to instructions and warnings. Any unskilled operation or other improper use of this product in violation of the respective safety instructions, warnings, or other instructions contained in the user documentation constitutes contributory negligence, which may lead to a limitation of liability by dSPACE GmbH, its representatives, agents and regional dSPACE companies, to the point of total exclusion, as the case may be. Any exclusion or limitation of liability according to other applicable regulations, individual agreements, and applicable general terms and conditions remain unaffected.

**Data loss during operating system shutdown**

The shutdown procedure of Microsoft Windows operating systems causes some required processes to be aborted although they are still being used by dSPACE software. To avoid data loss, the dSPACE software must be terminated manually before a PC shutdown is performed.

# Introduction to the Bus Manager

## Where to go from here

## Information in this section

Overview of the Bus Manager.....	15
Introduction to Bus Communication Defined in Communication Matrices.....	19
Use Scenarios for Configuring Bus Communication with the Bus Manager.....	20
Basics on Licenses for Working with the Bus Manager (Stand-Alone).....	24
Notes for Starting the Bus Manager (Stand-Alone).....	25
Elements of the Bus Manager.....	27

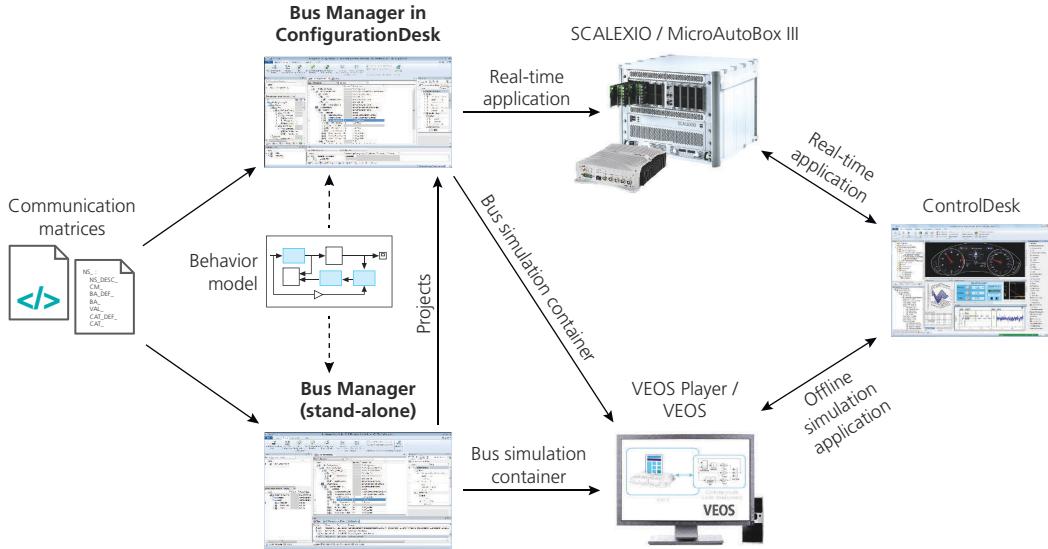
## Overview of the Bus Manager

### Purpose of the Bus Manager

The Bus Manager lets you configure bus communication for simulation, inspection, and manipulation purposes. You can work with multiple communication matrix files of various file formats and configure the communication of multiple communication clusters at the same time.

## Variants of the Bus Manager

Two variants of the Bus Manager are available. The following illustration provides an overview of the variants and their fields of application.



**Bus Manager in ConfigurationDesk** The Bus Manager in ConfigurationDesk lets you configure bus communication and implement it directly in a real-time application for dSPACE SCALEXIO or MicroAutoBox III systems. Alternatively, it lets you generate bus simulation containers. Bus simulation containers can be used on various dSPACE simulation platforms. For example, you can use bus simulation containers in the VEOS Player to implement the configured bus communication in an offline simulation application for VEOS.

**Bus Manager (stand-alone)** The Bus Manager (stand-alone) lets you configure bus communication and generate bus simulation containers. To use the configured bus communication on a specific dSPACE simulation platform, you can import the bus simulation containers to the VEOS Player or ConfigurationDesk. The VEOS Player lets you implement the bus communication in an offline simulation application, ConfigurationDesk lets you implement the bus communication in a real-time application. You can also reuse projects you used with the Bus Manager (stand-alone) with the Bus Manager in ConfigurationDesk (e.g., to work with an already configured bus communication and implement it in a real-time application).

## Bus Manager license

The Bus Manager (stand-alone) is license-protected. Refer to [Basics on Licenses for Working with the Bus Manager \(Stand-Alone\)](#) on page 24.

## Bus Manager (stand-alone) and ConfigurationDesk

The Bus Manager (stand-alone) uses elements of ConfigurationDesk, and ConfigurationDesk concepts apply to your work.

Whenever working with the Bus Manager (stand-alone) requires knowledge about ConfigurationDesk, this guide provides links to the relevant information in

the ConfigurationDesk documentation, such as the ConfigurationDesk Real-Time Implementation Guide.

**Note**

The ConfigurationDesk documentation provides information on the full functionality of ConfigurationDesk. When working with the Bus Manager (stand-alone), you cannot use the full ConfigurationDesk functionality.

In the documentation of the Bus Manager (stand-alone) and of ConfigurationDesk, a common terminology is used (e.g., 'ConfigurationDesk application' or 'signal chain'). For a brief description of frequently used terms, refer to [ConfigurationDesk Glossary](#) on page 347.

**Consistent user interface**

The Bus Manager provides a consistent user interface for different bus systems and use cases, letting you configure the bus communication according to your requirements. For example, you can configure the bus communication for various bus systems simultaneously without switching between different tools or views.

**Supported bus systems**

The Bus Manager supports the following bus systems:

- CAN
- LIN

For details on supported bus system features, refer to [Aspects of Supported Bus Communication](#) on page 37.

**Implementing bus communication in a ConfigurationDesk application**

The main elements for implementing bus communication in a ConfigurationDesk application are:

- Communication matrices
- Bus configurations

Communication matrices contain the definitions for bus communication. The Bus Manager lets you import communication matrices and create bus configurations. When you assign communication matrix elements to a bus configuration, you implement bus communication in the signal chain. You can configure this bus communication for simulation, inspection, and manipulation purposes before you generate bus simulation containers.

For more information, refer to:

- [Introduction to Bus Communication Defined in Communication Matrices](#) on page 19
- [Use Scenarios for Configuring Bus Communication with the Bus Manager](#) on page 20
- [Working with Communication Matrices](#) on page 58
- [Basics on Bus Configurations](#) on page 63

**Tip**

If you do not have a communication matrix, you can modify the `Basic_ComMatrix.arxml` file and use it as a starting point to set up CAN communication. For more information, refer to [Basics on Modifying Communication Matrices](#) on page 267.

**Supported communication matrix file formats**

The Bus Manager supports the following communication matrix file formats:

- AUTOSAR system description files
- FIBEX files
- DBC files
- LDF files

For details on the supported communication matrix file formats, refer to [Working with Communication Matrices](#) on page 58.

**Working with behavior models**

Behavior models can define the behavior of ECUs, for example.

When you configure bus communication with the Bus Manager, you can work with or without behavior models. If you work with behavior models, the behavior models must be Simulink implementation containers. To work with Simulink implementation containers, no local MATLAB/Simulink installation is needed.

However, if you want to use the full functionality of the Bus Manager when working with behavior models, MATLAB/Simulink must be installed and connected to your dSPACE installation.

For overviews of typical workflows for generating bus simulation containers with and without behavior models, refer to [Typical Workflows for Using the Bus Manager and Generating Bus Simulation Containers](#) on page 31.

**Conflict handling**

The Bus Manager lets you configure the bus communication without strict constraints. This lets you work more freely, but it can lead to conflicting configuration settings, such as duplicate CAN identifiers for CAN frame triggerings of one channel. Before generating bus simulation containers, you must resolve at least the most severe conflicts. You can view and resolve conflicts via the [Conflicts Viewer](#).

For more information, refer to [Basics on Bus Manager-Related Conflicts](#) on page 99.

**Support of ConfigurationDesk's automation interface**

You can automate Bus Manager features via ConfigurationDesk's automation interface. For more information, refer to [Accessing Bus Manager Features via ConfigurationDesk's Automation Interface](#) on page 97.

**Bus Manager Tutorial**

The Bus Manager Tutorial guides you through the basic steps when you start working with Bus Manager. Refer to [Bus Manager Tutorial](#).

**Bus Manager demo**

A Bus Manager demo is available, which introduces you to the basic steps of accessing the Bus Manager via automation. For more information, refer to [Working with the Bus Manager Demo](#) on page 297.

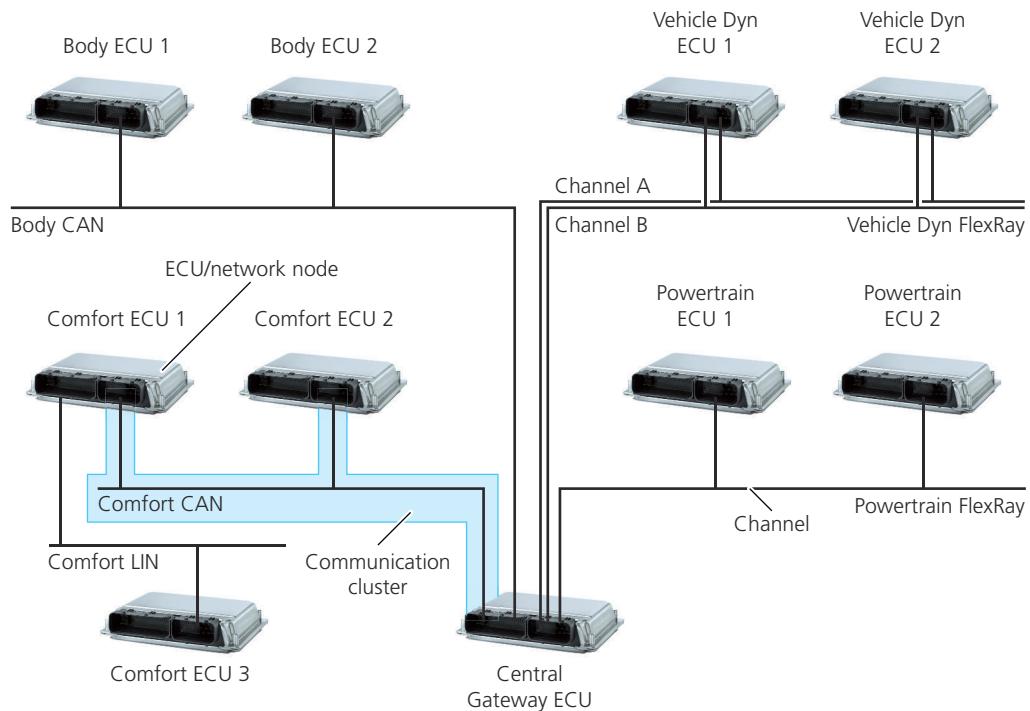
## Introduction to Bus Communication Defined in Communication Matrices

**Purpose of communication matrices**

To work with the Bus Manager, you need at least one communication matrix. Communication matrices define the communication of a bus network. A communication matrix can describe the bus communication of one communication cluster or a bus network consisting of different bus systems and communication clusters. The topology of the bus communication defined in a communication matrix consists of various elements.

**Example for bus communication defined in a communication matrix**

The following illustration is an example of a communication matrix that defines a simple bus communication network of a vehicle consisting of three bus systems (CAN, LIN, FlexRay) and five communication clusters.



## Elements of a bus communication topology

The following terms are used to describe the topology of the bus communication defined in a communication matrix:

- **Channel**  
Describes the physical channel that is used for bus communication. Depending on the bus protocol, one or more channels are used for bus communication: for example, CAN and LIN use one channel, FlexRay uses up to two channels.
- **Communication cluster**  
Describes all the network nodes that are connected to the same physical channel(s) and share the same bus protocol and address range.
- **ECU/network node**  
An ECU can be a member of multiple bus systems and communication clusters. Depending on the focus of the bus communication that is performed by an ECU, the following terms are used:
  - *ECU* addresses the entire bus communication of an ECU, i.e., the communication for all the connected communication clusters.
  - *Network node* addresses the bus communication of an ECU for only one communication cluster.

# Use Scenarios for Configuring Bus Communication with the Bus Manager

---

## Overview

The Bus Manager lets you implement bus communication in a ConfigurationDesk application. You can import communication matrices and configure the bus communication of various bus systems at the same time, for example, to perform a restbus simulation, manipulate bus communication, or inspect the bus communication of a bus channel. Additionally, you can configure gateways for CAN communication between two bus channels.

The Bus Manager lets you independently configure the bus communication for simulation, manipulation, and/or inspection purposes. For example, if you configure bus communication for simulation purposes or change the already configured bus communication, this does not affect the bus communication that is configured for manipulation or inspection purposes.

You can generate bus simulation containers that include the configured bus communication. You can import the generated bus simulation containers to VEOS Player or ConfigurationDesk to implement the configured bus communication in an offline simulation application or real-time application, respectively. You can access the bus communication of the real-time application and offline simulation application via experiment software.

## Use scenarios for simulating bus communication

The Bus Manager lets you simulate the bus communication between ECUs of the same or different bus systems or communication clusters.

**ECU or restbus simulation** You can simulate individual ECUs or perform a restbus simulation.

- ECU simulation

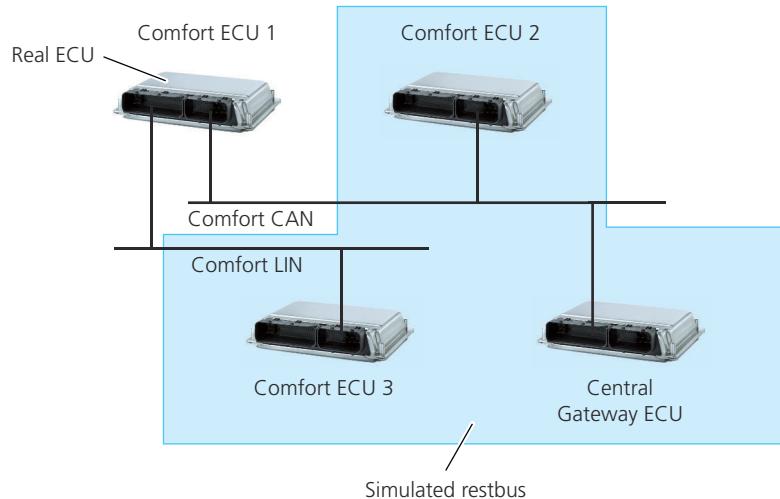
Simulating individual ECUs lets you, for example, test the behavior model of an ECU before the real ECU is available.

- Restbus simulation

Restbus simulation lets you test one or more ECUs while simulating the other ECUs of the related communication clusters. ECUs under test can be one of the following:

- Real ECUs that are connected to a simulator.
- V-ECU implementations that are implemented in an offline simulation.

For example, 'Comfort ECU 1' is available as a real ECU and connected to a simulator. To test this ECU, you can simulate all the other members of the 'Comfort CAN' and 'Comfort LIN' clusters (restbus simulation). The following illustration displays this example graphically:



**Static and dynamic simulation** Depending on the requirements of your simulation, you can perform a static and/or dynamic simulation:

- Static simulation

You can perform a static simulation to transmit or receive constant signal values on the bus. This is often sufficient in most use cases: For example, you perform a restbus simulation and the ECU under test expects a cabin temperature within a specific temperature range from an ECU that is part of the simulated restbus. In this case, you can let the simulated ECU transmit a constant temperature value that is within this temperature range.

You can use constant values that are defined in the communication matrix or specify user-defined constant values in ConfigurationDesk. Static simulation lets you work without a behavior model that defines, for example, the behavior of ECUs.

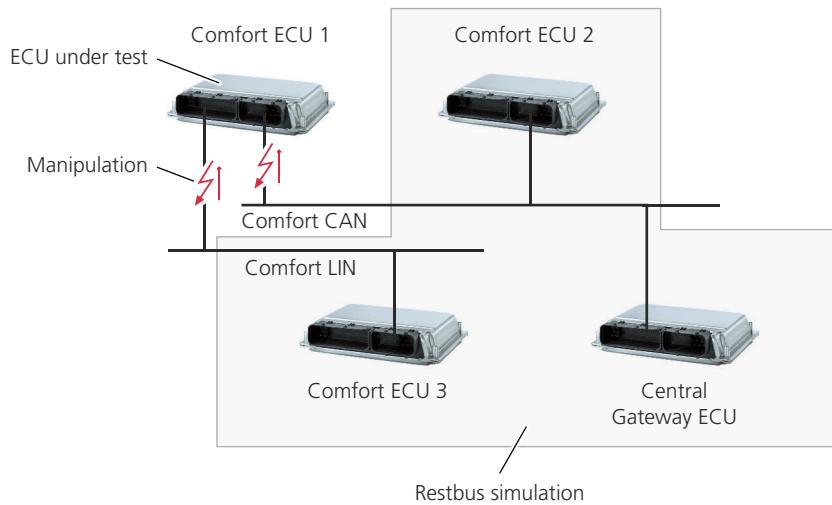
- Dynamic simulation

When signals of a simulated ECU must change dynamically during run time (e.g., cabin temperature values that are regulated by the ECU under test and exchanged with the simulated ECU), you must perform a dynamic simulation:

You must connect these signals to a behavior model that is, for example, modeled in MATLAB/Simulink. The control algorithms of the behavior model calculate the dynamic signal values of the simulated ECU during run time. If required, you can also switch between static and dynamic signal values during run time.

### Use scenario for manipulating bus communication

The Bus Manager lets you manipulate the bus communication before it is transmitted to the ECU under test (e.g., a real ECU that is connected to the simulator). This lets you test the response of the ECU under test in case it receives faulty bus communication, for example. The following illustration is an example of bus communication manipulation in combination with a restbus simulation.



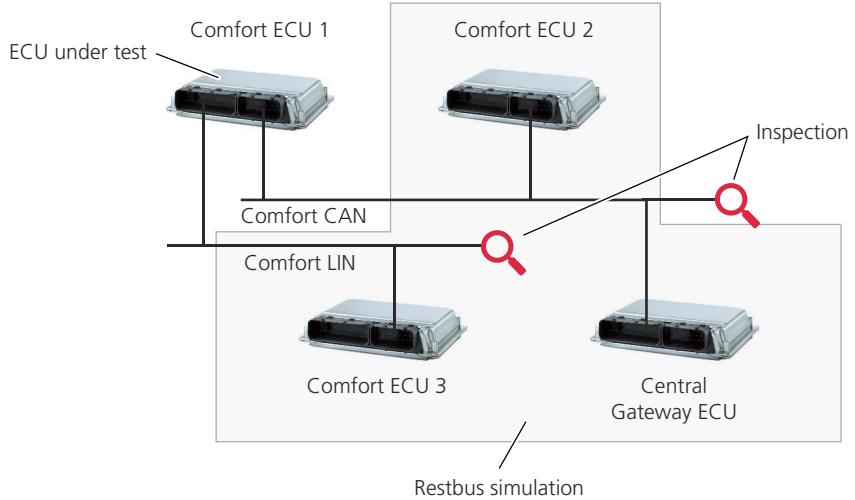
The manipulation of bus communication is based on communication clusters, i.e., you can manipulate the bus communication that is transmitted to the ECU under test independently from the sending ECUs.

Depending on your use scenario, you can manipulate the bus communication permanently or temporarily:

- Permanent manipulation lets you test, for example, if and how the ECU under test reacts on the faulty bus communication.
- Temporarily manipulation lets you test, for example, the fault memory of the ECU under test.

### Use scenarios for inspecting bus communication

The Bus Manager lets you inspect the bus communication of communication clusters independently from the involved ECUs. The following illustration is an example of inspecting bus communication in combination with a restbus simulation.



Bus communication inspection is performed for each cluster and channel separately. This allows you, for example:

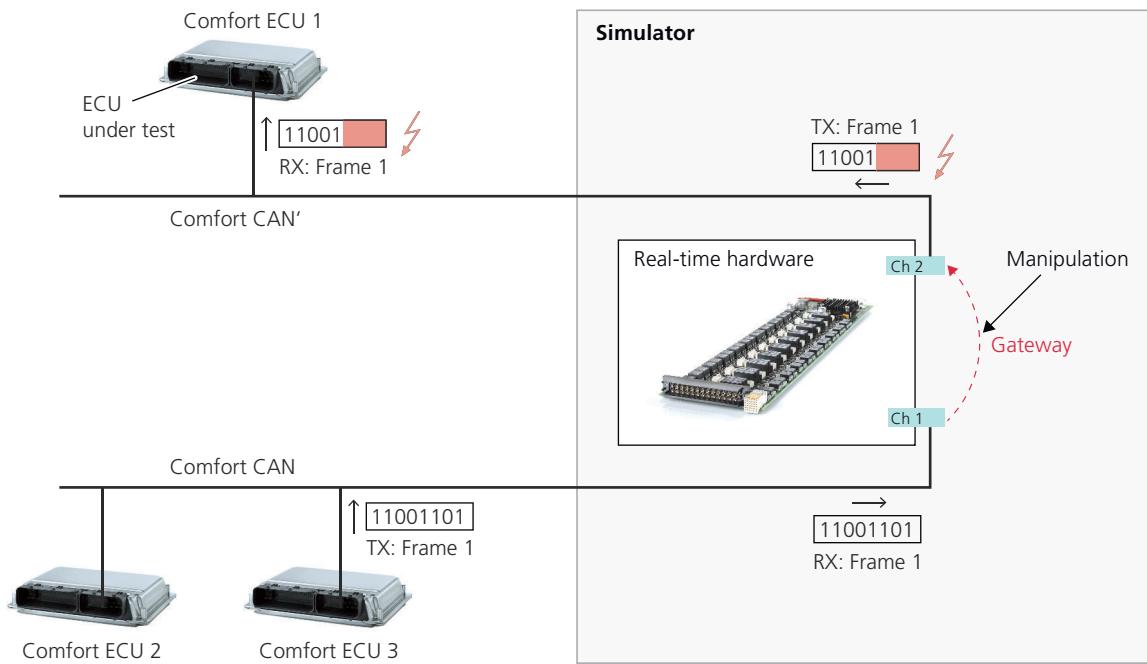
- To inspect the bus communication independently from the ECU development stage. For example, replacing a simulated ECU with a V-ECU or real ECU has no effect on your configurations.
- To reduce the effort for implementing bus communication in a ConfigurationDesk application. For example, three ECUs receive the same PDU but you only want to know that the PDU is available on the bus. In this case, the sender and receiver of the PDU are of no interest. For this, you have to configure the PDU only once for the inspection and not separately for each receiving ECU.

Depending on your use scenario, you can use the inspected bus communication in a mapped behavior model (e.g., to have a simulated ECU react on the inspected bus communication).

#### **Use scenario for gatewaying CAN communication**

The Bus Manager lets you gateway CAN communication, i.e., route the CAN communication between two bus channels. This lets you exchange frames between two CAN communication clusters.

A typical use scenario for gatewaying CAN communication is a failure gateway, i.e., the CAN communication is not only routed from one channel to another but also manipulated. The following illustration is an example for a failure gateway with real ECUs connected to the simulator. Only the ECU under test receives the gatewayed and manipulated frame.



#### Accessing and manipulating bus communication via an experiment software

Depending on your settings, you can access and manipulate the bus communication via an experiment software such as ControlDesk during run time. You can, for example:

- Switch the signal sources between a static signal value specified in ConfigurationDesk and dynamic signal values that are exchanged with the behavior model.
- Manipulate signal values to simulate value changes or faulty signals.

## Basics on Licenses for Working with the Bus Manager (Stand-Alone)

#### Required licenses

To work with the Bus Manager (stand-alone), you need the following licenses:

- **Bus Manager license (BUS\_MANAGER)**  
This license is mandatory.
- **CAN module license (CFD\_I\_CAN)**  
This license is needed only if you configure CAN bus communication.
- **LIN module license (CFD\_I\_LIN)**  
This license is needed only if you configure LIN bus communication.

---

**Enabling licenses**

Two different license types (single-user license, floating network license) are available for working with the Bus Manager (stand-alone) depending on your order:

- Single-user licenses: With single-user licenses, the dSPACE software is protected by a license mechanism based on a dongle. The dongle must be connected to your host PC to use the license-protected features of ConfigurationDesk.
  - Floating network licenses: With floating network licenses, the dSPACE License Client automatically requests the license required by a particular action from the connected dSPACE License Server (automatic activation). If there is at least one instance of each required license available, you can continue working with the dSPACE software as usual. At the same time, the license is blocked for other clients.
- 

**Invalid licenses**

A license becomes invalid when the dongle is removed from the PC or the required license is not available on the dSPACE License Server, for example. If you now try to carry out a license-protected action, an error message is displayed. However, you can save the ConfigurationDesk application.

---

**Using ConfigurationDesk applications without the required licenses**

You can open ConfigurationDesk applications even if not all the licenses required for working with the contained elements are available or enabled. For example, you can open a ConfigurationDesk application containing a CAN function block if the CFD\_I\_CAN license is not enabled, but you cannot work with such an application. If you try to do so, an error message is displayed. However, deleting elements from the application and saving the application is possible.

## Notes for Starting the Bus Manager (Stand-Alone)

---

**Restrictions for starting**

Neither ConfigurationDesk nor the Bus Manager (stand-alone) must be running. If you try to start the Bus Manager (stand-alone) while ConfigurationDesk or another Bus Manager (stand-alone) instance is running, an error occurs.

---

**Prerequisite for starting**

If dSPACE Bus Manager is available in the Windows Start menu, you can start the Bus Manager (stand-alone) as usual. If dSPACE Bus Manager is not available in the Start menu, the reason might be that the Bus Manager (stand-alone) is not part of your active RCP and HIL installation on your host PC. In that case, you have to activate the recommended installation first.

### Handling dSPACE installations

You can have different dSPACE installations on your host PC. A dSPACE installation is all the dSPACE software products installed in the same installation folder:

- If there is more than one RCP and HIL installation on your host PC, only one installation is the active one. You can access only the software belonging to the active installation.
- If there is only one RCP and HIL installation on your host PC, this is always the active installation.

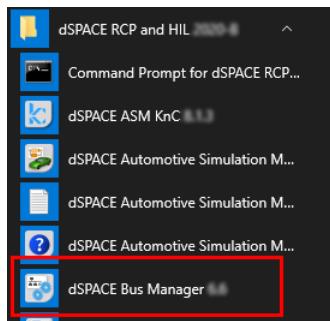
Installations are activated via dSPACE Installation Manager. You need administrator rights to do this. For detailed information, refer to [Activating and Deactivating dSPACE Installations \(Managing dSPACE Software Installations\)](#).

---

### Starting the Bus Manager (stand-alone)

You can start the Bus Manager (stand-alone) as follows:

In the Start menu, click dSPACE RCP and HIL <x.y> — dSPACE Bus Manager <x.y>.



<x.y> is a placeholder for the software version, since your host PC can have multiple installations and multiple items in the Start menu.

---

### First steps

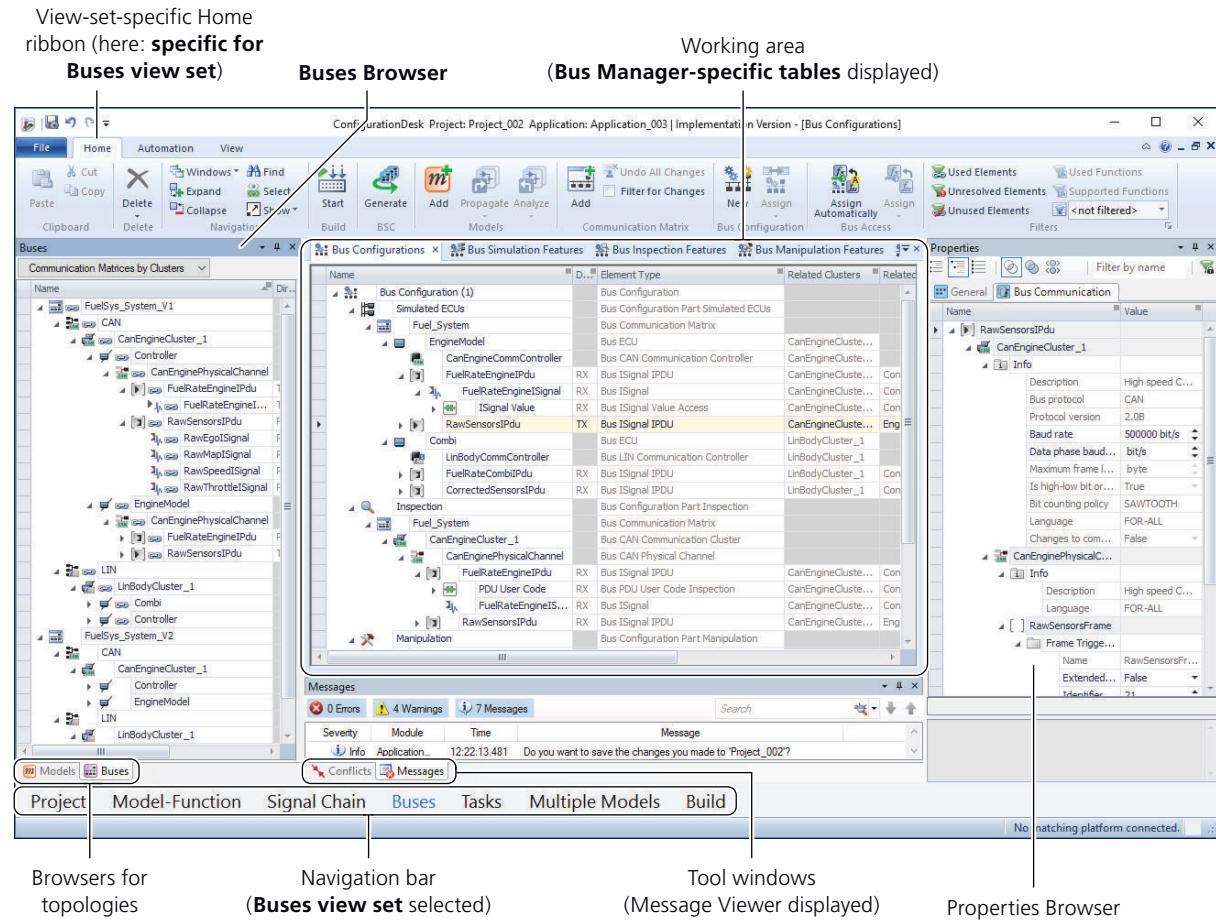
When you have started the Bus Manager (stand-alone), you can do the following:

- Make yourself familiar with the elements of the Bus Manager. Refer to [Elements of the Bus Manager](#) on page 27.
- Make yourself familiar with the workflow for using the Bus Manager (stand-alone) and generating bus simulation containers. Refer to [Typical Workflows for Using the Bus Manager and Generating Bus Simulation Containers](#) on page 31.
- Work through the Bus Manager Tutorial to make yourself familiar with the basic steps of working with the Bus Manager. Refer to [Bus Manager Tutorial](#).

## Elements of the Bus Manager

### User interface

When you work with the Bus Manager, you use Bus Manager elements and standard ConfigurationDesk elements. In the following illustration, the Bus Manager elements are in bold letters.



### Bus Manager elements of the user interface

The following elements of the user interface are specific to the Bus Manager.

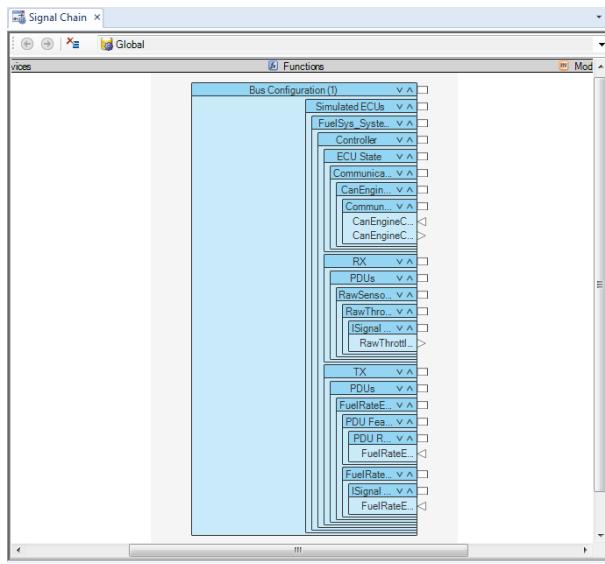
**Buses view set** Provides access to panes that are frequently used when you configure bus communication with the Bus Manager. Additionally, the Home ribbon provides access to frequently used commands when you configure bus communication. Some of these commands are specific to the Bus Manager.

**Buses Browser** Lets you add new communication matrices to the active ConfigurationDesk application and provides access to all the communication

matrices of the active application. You can select communication matrices completely or partly to assign them to bus configurations.

**Bus Manager-specific tables** Provide access to all the bus configurations of the active application. Via the different tables, you can configure the entire bus communication of the signal chain.

**Bus Configuration function block** Provides a graphical view on function ports of a bus configuration. When you configure bus communication in a bus configuration, you can access the related Bus Configuration function block in the Signal Chain Browser (available for the Signal Chain view set), for example.



**Bus Manager-specific elements in the Project Manager** The Project Manager (available in the Project view set) provides the following Bus Manager-specific elements:

- **Communication Matrices folder**

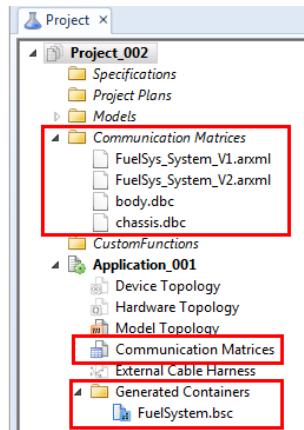
Lets you add communication matrices to ConfigurationDesk's project tree. When you do a backup of your project, the added communication matrices are included in the ZIP archive.

- **Communication Matrices node**

Represents an application component. The node is unavailable until you add the first communication matrix to the Buses Browser.

- **Generated Containers folder**

Represents an application component. This folder is unavailable until you generate bus simulation containers for the first time. Then, the folder displays the generated bus simulation container (BSC) files.



For a detailed description of ConfigurationDesk's user interface, refer to [User Interface of ConfigurationDesk \(ConfigurationDesk Real-Time Implementation Guide\)](#).



# Typical Workflows for Using the Bus Manager and Generating Bus Simulation Containers

---

Where to go from here	Information in this section						
	<table><tr><td>Workflow for Configuring Bus Communication and Generating Bus Simulation Containers.....</td><td>31</td></tr><tr><td>Generating Bus Simulation Containers with a Behavior Model.....</td><td>33</td></tr><tr><td>Generating Bus Simulation Containers Without a Behavior Model.....</td><td>34</td></tr></table>	Workflow for Configuring Bus Communication and Generating Bus Simulation Containers.....	31	Generating Bus Simulation Containers with a Behavior Model.....	33	Generating Bus Simulation Containers Without a Behavior Model.....	34
Workflow for Configuring Bus Communication and Generating Bus Simulation Containers.....	31						
Generating Bus Simulation Containers with a Behavior Model.....	33						
Generating Bus Simulation Containers Without a Behavior Model.....	34						

---

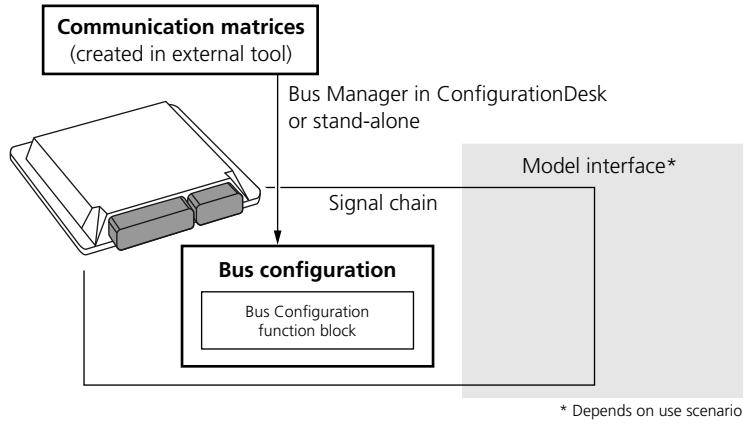
## Workflow for Configuring Bus Communication and Generating Bus Simulation Containers

---

<b>Introduction</b>	You can use the Bus Manager to configure bus communication and generate bus simulation containers that contain the configured bus communication. Bus simulation containers can be used on various dSPACE simulation platforms. For example, you can use bus simulation containers in the VEOS Player or ConfigurationDesk to implement the configured bus communication in offline simulation applications or real-time applications, respectively.
---------------------	---

---

<b>Overview</b>	The following illustration gives you an overview of the workflow for using the Bus Manager to configure bus communication and generate a bus simulation container that contains the configured bus communication. Parts that are specific to this workflow are in bold letters.
-----------------	---



## Workflow

Generating a bus simulation container containing the bus communication configured with the Bus Manager comprises the following workflow steps:

1. Create a new or open an existing ConfigurationDesk project and application.  
For more information on managing projects and applications, refer to [Managing ConfigurationDesk Projects and Applications \(ConfigurationDesk Real-Time Implementation Guide\)](#).
2. Add one or more communication matrices to the application.  
Refer to [How to Add a Communication Matrix to a ConfigurationDesk Application](#) on page 233.
3. Add one or more bus configurations to the application.  
Refer to [How to Add a Bus Configuration to a ConfigurationDesk Application](#) on page 236.
4. Assign the communication matrices completely or partly to the bus configurations.  
Refer to [How to Assign Communication Matrix Elements to a Bus Configuration](#) on page 238.
5. Configure the bus communication, for example:
  - Add bus configuration features to elements of bus configurations. Bus configuration features let you configure ISignals, trigger TX PDUs, or disable communication controllers, for example.  
Refer to [Working with Bus Configuration Features](#) on page 115.
  - Specify user-defined settings for configurable communication matrix elements.  
Refer to [Modifying Communication Matrices](#) on page 267.
6. Generate the bus simulation container. The necessary steps depend on your use scenario. For more information, refer to the following workflows:
  - [Generating Bus Simulation Containers with a Behavior Model](#) on page 33
  - [Generating Bus Simulation Containers Without a Behavior Model](#) on page 34

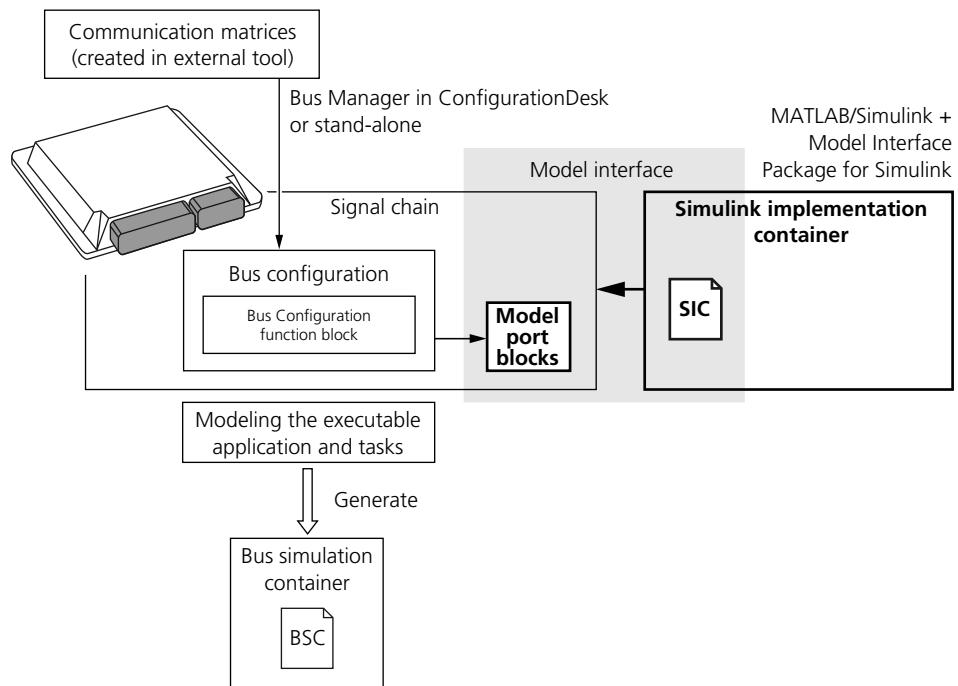
# Generating Bus Simulation Containers with a Behavior Model

## Introduction

When signals that are assigned to a bus configuration must change dynamically during run time (e.g., for dynamic ECU simulation), you need a behavior model and must map the required signals to the model before you generate bus simulation containers. The behavior model must be a Simulink implementation container (SIC) file.

## Overview

The following illustration gives you an overview of working with a behavior model. Parts that are specific to this use scenario are in bold letters.



## Workflow

Generating a bus simulation container that contains the bus communication configured with the Bus Manager and a mapped Simulink implementation container comprises the following workflow steps:

1. Configure the bus communication via the Bus Manager.  
Refer to [Workflow for Configuring Bus Communication and Generating Bus Simulation Containers](#) on page 31.
2. Enable model access for function ports.  
Refer to [How to Enable Model Access for Function Ports](#) on page 242.
3. Add a suitable Simulink implementation container (SIC) file to the ConfigurationDesk application.

Refer to [How to Add Behavior Models to a ConfigurationDesk Application](#) on page 244.

**Tip**

If you do not have a suitable SIC file yet but MATLAB/Simulink and [Model Interface Package for Simulink](#) are installed, ConfigurationDesk can support you by creating the required model interface. When you do this, the model interface is created in ConfigurationDesk and a MATLAB/Simulink model. From the MATLAB/Simulink model, you can then generate an SIC file and add this file to the ConfigurationDesk application.

For more information, refer to [Working with Simulink Behavior Models \(ConfigurationDesk Real-Time Implementation Guide\)](#) and [Generating Simulink Implementation Containers \(Model Interface Package for Simulink - Modeling Guide\)](#).

4. Map model ports of the SIC file to function ports of bus configurations.
  - For an example of mapping the ports via bus configuration tables, refer to [Example of Mapping Model Ports to Bus Configuration Ports via Tables](#) on page 249.
  - For basic information on model port mapping, refer to [Model Port Mapping \(ConfigurationDesk Real-Time Implementation Guide\)](#).
5. Model the executable application and tasks.

If you created a preconfigured application process when you added the SIC file to the ConfigurationDesk application, you can usually skip this step. The settings of the preconfigured application process and the tasks are sufficient in most cases.

For basic information, refer to:

  - [Introduction to Modeling Executable Applications and Tasks \(ConfigurationDesk Real-Time Implementation Guide\)](#).
  - [Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager](#) on page 94
6. Generate the bus simulation container.

Refer to [How to Generate Bus Simulation Containers](#) on page 259.

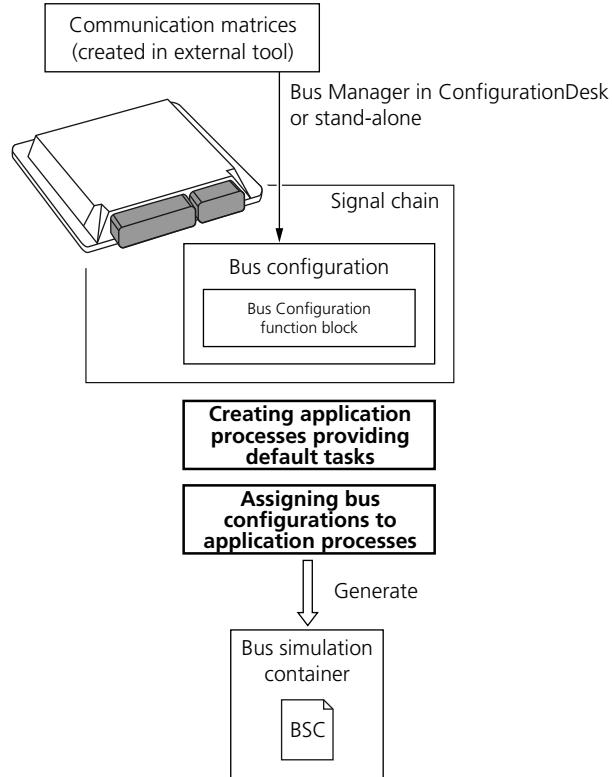
## Generating Bus Simulation Containers Without a Behavior Model

### Introduction

When signals that are assigned to a bus configuration do not have to change dynamically during run time (e.g., in case of static ECU simulation) or you only want to access the signals via an experiment software, you can work without a behavior model.

## Overview

The following illustration gives you an overview of working without a behavior model. Parts that are specific to this use scenario are in bold letters.



## Workflow

Generating bus simulation containers that contain the bus communication configured with the Bus Manager but no behavior model comprises the following workflow steps:

1. Configure the bus communication via the Bus Manager.  
Refer to [Workflow for Configuring Bus Communication and Generating Bus Simulation Containers](#) on page 31.
2. Create one or more application processes that provide default tasks.  
Refer to [How to Create Application Processes that Provide Default Tasks](#) on page 264.
3. Assign the bus configurations to the available application processes manually.  
Refer to [How to Manually Assign Bus Configurations to Application Processes](#) on page 265.
4. Generate the bus simulation containers.  
Refer to [How to Generate Bus Simulation Containers](#) on page 259.



# Aspects of Supported Bus Communication

## Where to go from here

## Information in this section

Supported PDU Types and Signal Data Types.....	37
Aspects of Supported AUTOSAR Features.....	39
Aspects of Supported CAN Bus Features.....	47
Aspects of Supported LIN Bus Features.....	52
Signal Conversion by the Bus Manager.....	54

## Supported PDU Types and Signal Data Types

### Supported PDU types

The Bus Manager supports various PDU types according to AUTOSAR, FIBEX, DBC, and LDF. To provide a consistent representation of the supported PDU types, the Bus Manager uses PDU-specific element types that are derived from the AUTOSAR PDU entities. PDUs of other communication standards are mapped to these element types. The following PDU-specific element types are available:

- Bus ISignal IPDU
- Bus General-Purpose IPDU
- Bus General-Purpose PDU
- Bus DCM IPDU
- Bus NMPDU
- Bus NPDU
- Bus Secured IPDU
- Bus User-Defined IPDU
- Bus User-Defined PDU
- Bus Multiplexed IPDU

- Bus Extended Multiplexed IPDU
- Bus Container IPDU

**Note**

The Bus Manager does not provide access to all the features and functions that are defined in the communication standards for the supported PDU types.

Except for multiplexed IPDUs, container IPDUs, and secured IPDUs, the Bus Manager provides the same functionalities for the supported PDU types. These PDU types are named *basic PDUs* in the documentation. In the ConfigurationDesk application, basic PDUs are represented by the or symbol in tables and browsers. However, basic PDUs can be distinguished by their PDU element type in tables, browsers, and via the automation interface. Therefore, you can filter for specific PDU types in tables or access them directly via XPath expressions in automation scripts, for example.

**Further information**

- For more information on multiplexed, container, and secured IPDUs, refer to [Aspects of Supported AUTOSAR Features](#) on page 39.
- For more information on extended multiplexed IPDUs, refer to [Aspects of Supported CAN Bus Features](#) on page 47.
- For more information on the mapping of the Bus Manager PDU types to the AUTOSAR, FIBEX, DBC, and LDF PDU types, refer to [Mapping of Bus Manager PDU Types to Communication Standard PDU Types](#) on page 335.
- For more information on accessing Bus Manager elements via XPath expressions, refer to [Accessing Bus Manager Features via ConfigurationDesk's Automation Interface](#) on page 97.
- For more information on limitations regarding the supported PDU types, refer to:
  - [Limitations for Communication Matrices and Communication Standards](#) on page 319
  - [Limitations for CAN Communication](#) on page 323
  - [Limitations for LIN Communication](#) on page 325

---

**Supported signal data types**

The Bus Manager supports signals of the following data types:

- Int8
- Int16
- Int32
- Int64
- UInt8
- UInt16
- UInt32
- UInt64
- UInt8[] (array signals)

- Float (Float32)
- Double (Float64)
- Boolean

Other data types (e.g., ASCII string, char, bitfield) are not supported. Signals whose signal length is a multiple of 8 and exceeds 64 bits are imported as UInt8[] array signals with static length. For the support of array signals, some limitations apply. Refer to [Limitations for array signals](#) on page 322.

## Aspects of Supported AUTOSAR Features

### Multiplexed IPDUs

For CAN communication, the Bus Manager supports multiplexed IPDUs. A multiplexed IPDU 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 ISignal IPDU that is configured as a dynamic part alternative, there must be a unique selector field value. The selector field value is evaluated by the receiver of the multiplexed IPDU.
- The static part is one ISignal IPDU that is always transmitted.

#### Tip

When you select a multiplexed IPDU, e.g., in the Buses Browser or Bus Configurations table, the Properties Browser might aggregate property values and display them in a comma-separated list for only one property. In this case, it can be useful to select the  intersection mode in the Properties Browser to display the properties for each related element, e.g., for each dynamic part segment.

### Container IPDUs and contained IPDUs

For CAN FD communication, the Bus Manager supports container IPDUs. A container IPDU consists of several smaller IPDUs (i.e., contained IPDUs). The contained IPDUs can be ISignal IPDUs or multiplexed IPDUs, for example. A container IPDU can be used to transmit several classic CAN-compliant IPDUs in one CAN-FD-compliant IPDU on a CAN FD bus, for example.

**Dynamic and static container layout** The container layout of container IPDUs can be organized dynamically or statically:

Container Layout	Description
Dynamic container layout	<p>The contained IPDUs do not have a predefined position in the container IPDU. The receiving ECU identifies the contained IPDUs via their header IDs.</p> <p>For this purpose, two header types are available (<b>SHORT-HEADER</b>, <b>LONG-HEADER</b>). A contained IPDU can provide a separate header ID for each of these header types. The container IPDU defines the required header type: The ID of the required header type must be specified for each contained IPDU and the receiving ECU evaluates only these IDs.</p>
Static container layout	<p>The contained IPDUs have a fixed position within the container IPDU. The receiving ECU identifies the contained IPDUs unambiguously by their position within the container IPDU.</p> <p>For this purpose, the header type of the container IPDU must be set to <b>NO-HEADER</b> and an offset value must be specified for each contained IPDU. The offset value determines the position of the contained IPDU within the container IPDU.</p>

**Triggering of container IPDUs** The triggering of container IPDUs depends on various timing and triggering conditions that can be specified for container and/or contained IPDUs in the communication matrix. For example, a container IPDU can be transmitted:

- When a specified timeout value elapses
- Immediately after the first contained IPDU is added
- When a specific contained IPDU triggers its transmission

The Bus Manager supports various AUTOSAR-compliant timing and triggering conditions. Additionally, the Bus Manager also triggers the transmission of container IPDUs with a static container layout if all of the following conditions are met:

- The **CONTAINER-TRIGGER** attribute of a container IPDU with a static container layout is set to **DEFAULT-TRIGGER**.
- The data of a contained IPDU has changed or its transmission is triggered but the **TRIGGER** attribute of the contained IPDU is not specified.

**Queuing of contained IPDUs** For each contained IPDU that is transmitted on the bus, the communication matrix must specify the collection semantics:

- With a queued semantics, several instances of the contained IPDU can be added to one container IPDU.
- A queued semantics is supported only for contained IPDUs that are included in container IPDUs with a dynamic container layout.
- With a last-is-best semantics, only one instance of the contained IPDU can be added to one container IPDU. In this case, the data of the contained IPDU is buffered and only the latest data is added to the container IPDU just before it is transmitted.

For contained IPDUs that are included in container IPDUs with a static container layout, only the last-is-best semantics is supported.

The Bus Manager supports both semantics. However, when a container IPDU is received that contains several instances of a contained IPDU, only the last received instance can be displayed.

**Identifying updated contained IPDUs in container IPDUs with static container layout** When a container IPDU with a static container layout is transmitted, some of its contained IPDUs might not have been updated since the previous transmission and therefore provide old data. There are two ways to identify the update state:

Setting	Description
Update bit of contained IPDU	Contained IPDUs can provide an update bit. The bit is set if a contained IPDU has been updated between two transmissions of the related container IPDU.
Unused bit pattern of container IPDU	Container IPDUs can provide an unused bit pattern. If a contained IPDU of the container IPDU is not updated, the content of the contained IPDU is cleared and replaced with the unused bit pattern.

**Accepting contained IPDUs received with container IPDUs with dynamic container layout** For container IPDUs with a dynamic container layout, the setting of the RX-ACCEPT-CONTAINED-IPDU attribute determines which contained IPDUs are accepted by the receiving ECU:

Setting	Description
ACCEPT-ALL	In general, the receiving ECU accepts any contained IPDU, regardless of whether it is included in the container IPDU according to the specifications in the communication matrix. However, to accept a contained IPDU that is not included in the container IPDU according to the specifications in the communication matrix, the following conditions must be met: <ul style="list-style-type: none"> <li>▪ The communication matrix includes the IPDU in another container IPDU whose RX-ACCEPT-CONTAINED-IPDU attribute is set to ACCEPT-ALL.</li> <li>▪ The header ID of the header type that is required by the container IPDU is specified for the contained IPDU.</li> </ul>
ACCEPT-CONFIGURED	The receiving ECU accepts only contained IPDUs that are included in the container IPDU according to the specifications in the communication matrix.

Contained IPDUs that are not accepted are ignored by the receiving ECU, i.e., they are not extracted from the container IPDU and their data is not processed.

#### Secure onboard communication for PDUs

For CAN communication, the Bus Manager supports secure onboard communication (SecOC) according to AUTOSAR. Secure onboard communication provides authentication mechanisms to identify PDU data that was modified or transmitted without authorization, e.g., due to a replay attack.

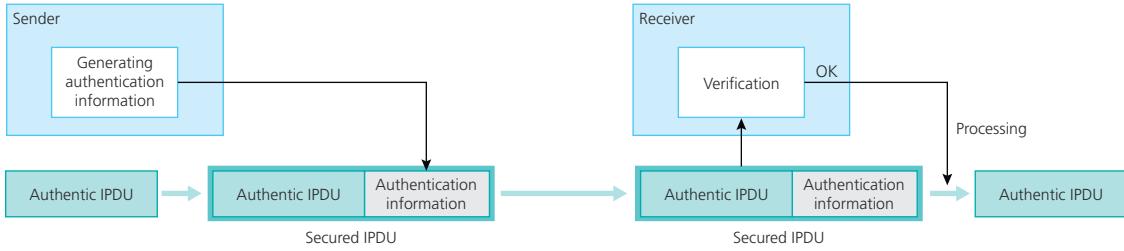
**PDUs for SecOC** In secure onboard communication scenarios, the following PDUs are used:

- Authentic IPDUs

Authentic IPDUs contain the payload, which will be secured by the authentication mechanisms of SecOC. The required authentication information itself is not included in authentic IPDUs but in secured IPDUs. According to AUTOSAR, authentic IPDUs can be ISignal IPDUs, container IPDUs, or multiplexed IPDUs, for example.

- Secured IPDUs and cryptographic IPDUs

Secured IPDUs are used to secure the payload of authentic IPDUs, i.e., they contain the required authentication information. The sender includes the authentication information in the secured IPDU and the receiver verifies the received authentication information, as shown in the following example.



The handling of the related authentic IPDUs depends on whether secured IPDUs are configured as cryptographic IPDUs:

- If a secured IPDU is configured as a cryptographic IPDU, the related authentic IPDU is not included in the secured IPDU. In this case, the authentic IPDU and the cryptographic IPDU are separately transmitted on the bus. The receiver verifies the authentication information each time the authentic IPDU or cryptographic IPDU is received.
- If a secured IPDU is not configured as cryptographic IPDU, the related authentic IPDU is directly included in the secured IPDU. In this case, only one PDU is exchanged on the bus.

**Authentication information** According to AUTOSAR, there are various ways for generating and verifying authentication information. To generate and verify specific authentication information, OEM-specific implementations are therefore required. In general, authentication information that is included in a secured IPDU consists of a freshness value and an authenticator:

- Freshness value

The freshness value is a monotonous increasing value. Depending on the OEM-specific implementation, the freshness value can be a counter value or a time stamp value. The freshness value is required for calculating the authenticator. Additionally, the freshness value can directly be included in the secured IPDU. If it is, it can be included completely or in part, i.e., as truncated freshness value.

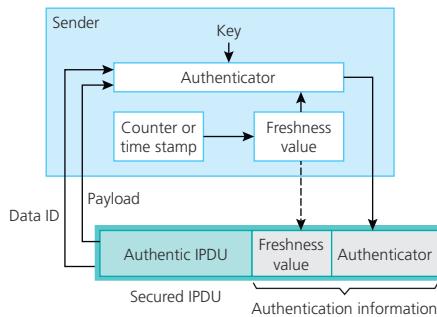
- Authenticator

The authenticator is calculated according to OEM-specific algorithms and keys. For calculating the authenticator, the following data is required:

- Data identifier of the secured IPDU
- Payload of the authentic IPDU
- Freshness value

The authenticator is either completely included in the secured IPDU or in part, i.e., as truncated authenticator.

The following illustration is an example of authentication information in a secured IPDU.



To implement secure onboard communication in [executable applications](#), you must enable SecOC support and provide the OEM-specific implementation for generating and/or verifying authentication information via user code. Refer to [Implementing Secure Onboard Communication in Executable Applications](#) on page 87.

#### End-to-end protection for ISignal groups

The Bus Manager supports end-to-end (E2E) protection for ISignal groups according to the following AUTOSAR end-to-end protection profiles (E2E profiles):

- Profile 01 (including the profile variants 1A, 1B, 1C)
- Profile 02
- Profile 05
- Profile 11 (including the profile variants 11A, 11C)
- Profile 22

The Bus Manager supports COM E2E callouts (only for profile 01 and profile 02) and E2E transformers (for all supported profiles) for calling the end-to-end communication protection library (E2E library).

The end-to-end protection of an ISignal group must be specified in the communication matrix. Each end-to-end-protected ISignal group must be mapped to exactly one ISignal IPDU. An ISignal IPDU can contain one or more end-to-end-protected ISignal groups.

The Bus Manager can transmit and receive end-to-end-protected ISignals groups:

- TX ISignal groups are transmitted with the end-to-end protection information that is required according to the specified profile.
- RX ISignal groups are received but the end-to-end protection information is not evaluated by the Bus Manager, i.e., a received IPDU is decoded regardless of whether the end-to-end protection of a contained ISignal group indicates failures.

#### Global time synchronization

The Bus Manager supports global time synchronization (GTS) according to AUTOSAR. Global time synchronization means providing and distributing synchronized times among all ECUs in a bus network.

**Time bases** A synchronized time is called a time base. A time base is a unique time entity characterized by the progression of time, ownership, and reference to the physical world. There are two types of time bases:

Time Base Type	Description
Synchronized time base	A synchronized time base is a time base that is synchronized with other time bases of different bus network participants. These synchronized time bases constitute a global time.
Offset time base	An offset time base is a time base that depends on a particular synchronized time base and holds an offset value with respect to that time base.

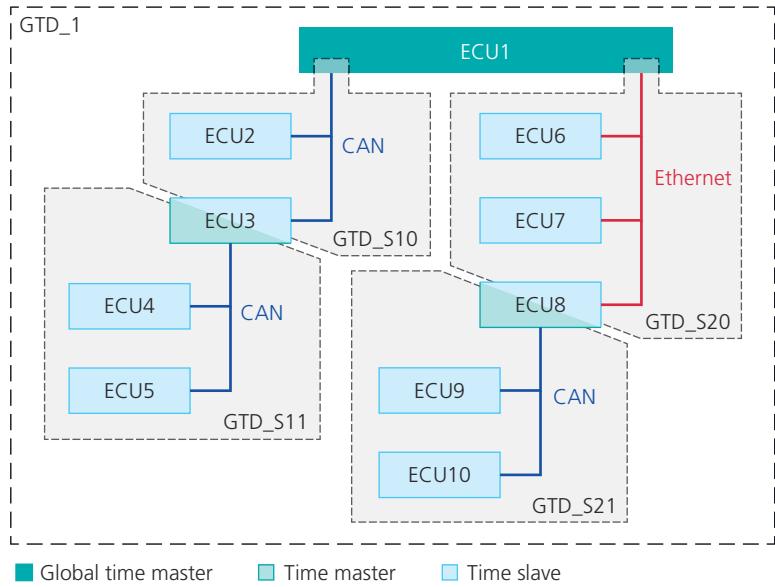
Time bases can provide absolute times (e.g., a UTC time) or relative times (e.g., time after power-up of an ECU or an operating hours counter). Because there can be more than just one time in a bus network, multiple time bases can exist in parallel. Per ECU, there can be up to 16 synchronized time bases and offset time bases each.

**Global time domains** A global time domain contains components (e.g., [network nodes](#)) that are linked to the same synchronized time base. There can be several global time domains in a bus network. In general, global time domains can be distinguished by domain identifiers as follows:

- Global time domains that use different synchronized time bases have different domain identifiers.
- Global time domains that use the same synchronized time base have the same domain identifier.

Global time domains that use the same synchronized time base are connected by the means of subdomains. A subdomain denotes which components are linked to the related time base. Each subdomain is limited to exactly one [communication cluster](#). A global time domain can declare several other global time domains as its subdomains.

The following illustration is an example for the distribution of a global time in a bus network with four communication clusters and five global time domains.  $GTD_{S10}$  and  $GTD_{S20}$  are subdomains of  $GTD_1$ .  $GTD_{S11}$  and  $GTD_{S21}$  are subdomains of  $GTD_{S10}$  and  $GTD_{S20}$ , respectively. All global time domains use the same domain identifier.



Depending on its role in the network, an ECU can be a time master and/or a time slave.

**Time masters, time slaves, and time gateways** Global time synchronization is based on a master-slave system. An ECU can be both master and slave at the same time. Additionally, if synchronized times are processed via multiple communication clusters, an ECU can act as a time gateway. To distinguish between the different roles, the following terms are used:

Role	Description	ECUs in the Previous Illustration
Time master	An entity that is the master for a certain time base and distributes this time base to a set of time slaves within a certain cluster of a bus network.	ECU3 (only for GTD_S11), ECU8 (only for GTD_S21)
Global time master	A time master that is also the global owner and origin of the time base.	ECU1
Time slave	An entity that is the recipient of a certain time base within a cluster of a bus network. Each time slave has its own time base instance, i.e., a local instance of the time. Whenever a time slave receives a new time from the time master, the time slave interpolates the time until the next synchronization with the time master.	e.g., ECU2, ECU3 (only for GTD_S10), ECU4, ECU7, ECU8 (only for GTD_S20)
Time gateway	A time base instance that receives the synchronized time as a time slave from a time master on one cluster and then forwards it to another cluster as a time master. A time gateway typically consists of one time slave and one or more time masters.	ECU3, ECU8

**Time synchronization messages** The messages that distribute the time information are called time synchronization messages.

With time synchronization over CAN, each time synchronization message consists of two messages that are transmitted separately: a synchronization

(SYNC) message and a follow-up (FUP) message. The latter completes the synchronization process. SYNC and FUP messages are transmitted using the same CAN identifier.

SYNC and FUP messages are identified by their message type, and their message layout depends on whether they are CRC-secured:

CRC-Secured	Message Layout																								
CRC-secured	<p><b>SYNC:</b></p> <table border="1"> <tr> <td>Message type 8 bits</td> <td>CRC 8 bits</td> <td>Time domain 4 bits</td> <td>E2E sequence counter 4 bits</td> <td>User byte 0 8 bits</td> <td>Seconds 32 bits</td> </tr> <tr> <td>Byte 0 MSB</td> <td>Byte 1</td> <td>Byte 2</td> <td></td> <td>Byte 3</td> <td>Byte 4 ... Byte 7 LSB</td> </tr> </table> <p><b>FUP:</b></p> <table border="1"> <tr> <td>Message type 8 bits</td> <td>CRC 8 bits</td> <td>Time domain 4 bits</td> <td>E2E sequence counter 4 bits</td> <td>Reserved 5 bits</td> <td>SGW (1 bit) Overflow of seconds (2 bits) Nanoseconds 32 bits</td> </tr> <tr> <td>Byte 0 MSB</td> <td>Byte 1</td> <td>Byte 2</td> <td></td> <td>Byte 3</td> <td>Byte 4 ... Byte 7 LSB</td> </tr> </table> <ul style="list-style-type: none"> <li>▪ SYNC message type: 0x20</li> <li>▪ FUP message type: 0x28</li> </ul>	Message type 8 bits	CRC 8 bits	Time domain 4 bits	E2E sequence counter 4 bits	User byte 0 8 bits	Seconds 32 bits	Byte 0 MSB	Byte 1	Byte 2		Byte 3	Byte 4 ... Byte 7 LSB	Message type 8 bits	CRC 8 bits	Time domain 4 bits	E2E sequence counter 4 bits	Reserved 5 bits	SGW (1 bit) Overflow of seconds (2 bits) Nanoseconds 32 bits	Byte 0 MSB	Byte 1	Byte 2		Byte 3	Byte 4 ... Byte 7 LSB
Message type 8 bits	CRC 8 bits	Time domain 4 bits	E2E sequence counter 4 bits	User byte 0 8 bits	Seconds 32 bits																				
Byte 0 MSB	Byte 1	Byte 2		Byte 3	Byte 4 ... Byte 7 LSB																				
Message type 8 bits	CRC 8 bits	Time domain 4 bits	E2E sequence counter 4 bits	Reserved 5 bits	SGW (1 bit) Overflow of seconds (2 bits) Nanoseconds 32 bits																				
Byte 0 MSB	Byte 1	Byte 2		Byte 3	Byte 4 ... Byte 7 LSB																				
Not secured	<p><b>SYNC:</b></p> <table border="1"> <tr> <td>Message type 8 bits</td> <td>User byte 1 8 bits</td> <td>Time domain 4 bits</td> <td>E2E sequence counter 4 bits</td> <td>User byte 0 8 bits</td> <td>Seconds 32 bits</td> </tr> <tr> <td>Byte 0 MSB</td> <td>Byte 1</td> <td>Byte 2</td> <td></td> <td>Byte 3</td> <td>Byte 4 ... Byte 7 LSB</td> </tr> </table> <p><b>FUP:</b></p> <table border="1"> <tr> <td>Message type 8 bits</td> <td>User byte 2 8 bits</td> <td>Time domain 4 bits</td> <td>E2E sequence counter 4 bits</td> <td>Reserved 5 bits</td> <td>SGW (1 bit) Overflow of seconds (2 bits) Nanoseconds 32 bits</td> </tr> <tr> <td>Byte 0 MSB</td> <td>Byte 1</td> <td>Byte 2</td> <td></td> <td>Byte 3</td> <td>Byte 4 ... Byte 7 LSB</td> </tr> </table> <ul style="list-style-type: none"> <li>▪ SYNC message type: 0x10</li> <li>▪ FUP message type: 0x18</li> </ul>	Message type 8 bits	User byte 1 8 bits	Time domain 4 bits	E2E sequence counter 4 bits	User byte 0 8 bits	Seconds 32 bits	Byte 0 MSB	Byte 1	Byte 2		Byte 3	Byte 4 ... Byte 7 LSB	Message type 8 bits	User byte 2 8 bits	Time domain 4 bits	E2E sequence counter 4 bits	Reserved 5 bits	SGW (1 bit) Overflow of seconds (2 bits) Nanoseconds 32 bits	Byte 0 MSB	Byte 1	Byte 2		Byte 3	Byte 4 ... Byte 7 LSB
Message type 8 bits	User byte 1 8 bits	Time domain 4 bits	E2E sequence counter 4 bits	User byte 0 8 bits	Seconds 32 bits																				
Byte 0 MSB	Byte 1	Byte 2		Byte 3	Byte 4 ... Byte 7 LSB																				
Message type 8 bits	User byte 2 8 bits	Time domain 4 bits	E2E sequence counter 4 bits	Reserved 5 bits	SGW (1 bit) Overflow of seconds (2 bits) Nanoseconds 32 bits																				
Byte 0 MSB	Byte 1	Byte 2		Byte 3	Byte 4 ... Byte 7 LSB																				

If global time synchronization is specified in the communication matrix, the Bus Manager lets you implement global time synchronization in [executable applications](#). Refer to [Implementing Global Time Synchronization in Executable Applications](#) on page 88.

## Limitations

When you work with bus communication according to AUTOSAR, some limitations apply. Refer to [Limitations for Communication Matrices and Communication Standards](#) on page 319.

**Related topics****Basics**

[Introduction to the Properties Browser \(ConfigurationDesk Real-Time Implementation Guide\)](#)

## Aspects of Supported CAN Bus Features

**Identifier format**

The Bus Manager supports [frames](#) with the standard (11-bit) and extended (29-bit) identifier format.

The identifier contains address information. It does not address individual bus members, but refers to the information the frame contains. As a result, a frame can be received by any number of bus members. Each bus member can filter out frames of interest. The identifier must be defined for each CAN frame.

For each CAN frame, you can change the specified identifier format and specify a user-defined identifier value.

**CAN FD protocol**

The Bus Manager supports the CAN FD protocol.

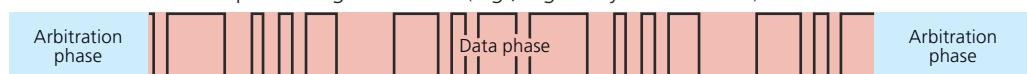
CAN FD stands for *CAN with Flexible Data Rate*. Compared with the classic CAN protocol, CAN FD comes with an increased bandwidth for serial communication. The improvement is based on two factors:

- Longer payload length

The data fields (payload length) of CAN FD frames can be up to 64 bytes long (classic CAN: 8 bytes). Valid CAN FD payload lengths are: 0 ... 8, 12, 16, 20, 24, 32, 48, and 64 bytes. If a CAN FD frame has an invalid payload length, the Bus Manager adds padding bytes to the frame and extends the payload length to the next higher valid length (e.g., to a payload length of 14 bytes, the Bus Manager adds 2 padding bytes). The value of each padding byte is 255.

- Higher bit rate for the data phase

CAN frames consist of an arbitration phase and a data phase. The data phase spans the phase in which the data fields, CRC, and length information are transferred. The data phase of CAN FD frames can be transmitted with an optional higher bit rate (e.g., higher by a factor of 8).



Currently, there are two CAN FD protocols on the market which are not compatible with each other.

- The *non-ISO CAN FD protocol* is the original CAN FD protocol from Bosch.
- The *ISO CAN FD protocol* is the CAN FD protocol according to ISO 11898-1:2015.

The ISO CAN FD protocol comes with an improved failure detection capability.

The Bus Manager supports both CAN FD protocols and lets you use classic CAN frames and CAN FD frames in parallel. For CAN FD frames, you can define a separate bit rate for the data phase. Additionally, the Bus Manager supports container IPDUs for CAN FD communication. For more information, refer to [Aspects of Supported AUTOSAR Features](#) on page 39.

## J1939 protocol

The Bus Manager supports the J1939 protocol as specified by the Society of Automotive Engineers (SAE) J1939 standard.

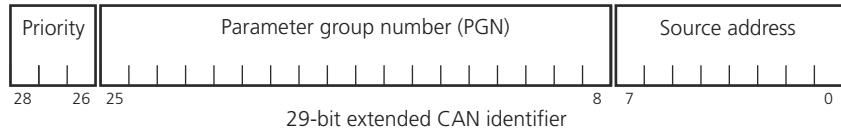
The J1939 protocol is used, for example, for the communication in heavy-duty vehicles, such as between a tractor and its trailer. It allows peer-to-peer and broadcast communication, and supports messages with up to 1,785 data bytes.

In the Bus Manager, J1939 messages are represented by J1939-compliant PDUs<sup>2</sup>, for example, by J1939-compliant ISignal IPDUs<sup>2</sup> or multiplexed IPDUs<sup>2</sup>.

**J1939-compliant PDUs** J1939-compliant PDUs can have a payload length with up to 1,785 data bytes. The transmission of such PDUs depends on the actual number of data bytes:

- 0 ... 8 data bytes: The PDU can be directly transmitted on the bus, i.e., as a J1939 direct PDU.
- 9 ... 1,785 data bytes: The PDU is segmented into multiple packets that are separately transmitted on the bus. For segmenting and transmitting the packets on the bus, two transport protocols are available. Refer to [Handling J1939-compliant PDUs with up to 1,785 data bytes](#) on page 50.

**Extended CAN identifier** The J1939 protocol uses a 29-bit extended CAN identifier. The identifier is segmented as follows:

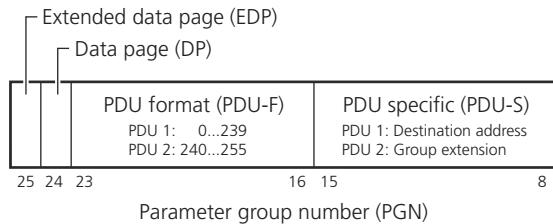


Segment	Description
Priority	The priority of the PDU. A value of 0 represents the highest priority, a value of 7 represents the lowest priority.
Parameter group number (PGN)	A unique number that unambiguously references a parameter group, i.e., the content of the PDU. The PGN itself consists of four segments. Refer to <a href="#">Parameter group number (PGN), parameter groups, and parameters</a> on page 48.
Source address	The transport protocol address of the network node that transmits the PDU.

**Parameter group number (PGN), parameter groups, and parameters** The *parameter group number (PGN)* is an 18-bit number that unambiguously references a parameter group. A *parameter group* is a group of parameters that are included in the data field of a PDU. For example, an engine temperature parameter group can group parameters that provide the coolant temperature, oil temperature, etc. Each parameter is unambiguously identified by

its *suspect parameter number (SPN)*. The J1939 standard defines PGNs, the referenced parameter groups, and the grouped parameters.

The PGN uses the bits 25 ... 8 of the CAN identifier, which are segmented as follows:



Segment	Description
Extended data page (EDP)	Reserved for future use according to the J1939 standard. Regardless of whether the EDP of a PDU is set to 0 or 1, the Bus Manager handles the PDU as J1939-compliant.
Data page (DP)	Specifies the used data page (data page 0 or data page 1). The data pages are used to expand the PGN value range.
PDU format (PDU-F)	Specifies the PDU format. Either PDU 1 or PDU 2. <ul style="list-style-type: none"> <li>▪ PDU 1: <ul style="list-style-type: none"> <li>▪ PDU-F value: 0 ... 239 (0x00 ... 0xEF)</li> <li>▪ Communication mode: Peer-to-peer, i.e., the PDU is transmitted to one destination network node.</li> </ul> </li> <li>▪ PDU 2: <ul style="list-style-type: none"> <li>▪ PDU-F value: 240 ... 255 (0xF0 ... 0xFF)</li> <li>▪ Communication mode: Broadcast, i.e., the PDU is transmitted to all network nodes in the communication cluster.</li> </ul> </li> </ul>
PDU specific (PDU-S)	The usage depends on the specified PDU format: <ul style="list-style-type: none"> <li>▪ PDU 1: <ul style="list-style-type: none"> <li>Provides the transport protocol address of the destination network node.</li> </ul> </li> <li>▪ PDU 2: <ul style="list-style-type: none"> <li>Provides the group extension. The group extension is used to increase the number of PDUs that can be broadcast in the communication cluster.</li> </ul> </li> </ul>

**J1939 name and transport protocol address** Each [network node](#) that participates in J1939 communication can be identified by its J1939 name and its transport protocol address. Regarding the J1939 name and transport protocol address, the following terms are used to address a network node:

Node Name	Description										
J1939 network management node (J1939 NM node)	Provides the J1939 name. The J1939 name is a 64-bit value that unambiguously identifies each J1939 network node worldwide. Therefore, the J1939 name value must be unique worldwide. The J1939 name indicates the main function of the network node and provides information on the manufacturer. For this purpose, the 64 bits are segmented as follows: <table border="1"> <tr> <td>Arbitrary Address Capable 1 bit</td> <td>Industry Group 3 bit</td> <td>Vehicle System Instance 4 bit</td> <td>Vehicle System 7 bit</td> <td>Reserved 1 bit</td> <td>Function 8 bit</td> <td>Function Instance 5 bit</td> <td>ECU Instance 3 bit</td> <td>Manufacturer Code 11 bit</td> <td>Identity Number 21 bit</td> </tr> </table>	Arbitrary Address Capable 1 bit	Industry Group 3 bit	Vehicle System Instance 4 bit	Vehicle System 7 bit	Reserved 1 bit	Function 8 bit	Function Instance 5 bit	ECU Instance 3 bit	Manufacturer Code 11 bit	Identity Number 21 bit
Arbitrary Address Capable 1 bit	Industry Group 3 bit	Vehicle System Instance 4 bit	Vehicle System 7 bit	Reserved 1 bit	Function 8 bit	Function Instance 5 bit	ECU Instance 3 bit	Manufacturer Code 11 bit	Identity Number 21 bit		

Node Name	Description
J1939 transport protocol node (J1939 TP node)	<p>Provides the transport protocol address.</p> <p>The transport protocol address is an 8-bit value that specifies the source or destination of a J1939-compliant PDU in the communication cluster. For this purpose, the value must be unique in the communication cluster. If there is an address conflict, the network node can dynamically claim an address to ensure unique addresses, if this is enabled for the network node.</p> <p>According to the J1939 standard, the following addresses are reserved:</p> <ul style="list-style-type: none"> <li>▪ Null address: <ul style="list-style-type: none"> <li>▪ Address value: 254 (0xFE)</li> <li>▪ Used as the source address by network nodes that have not yet claimed an address or have failed to claim an address.</li> </ul> </li> <li>▪ Global address: <ul style="list-style-type: none"> <li>▪ Address value: 255 (0xFF)</li> <li>▪ Used as a destination address that addresses all network nodes in a communication cluster, e.g., to broadcast PDUs of PDU 1 format or for address claims.</li> </ul> </li> </ul>

If a network node participates in J1939 communication but is not available in a DBC file, the Bus Manager generates a suitable ECU during the import of the DBC file. Refer to [Generating missing ECUs](#) on page 60.

**Address claiming** The J1939 standard defines an address claiming procedure in which transport protocol addresses are assigned to network nodes during communication cluster initialization. This procedure ensures that each address is unique.

Each network node can send an address claim to the CAN bus. The nodes that receive an address claim verify the claimed address. If there is an address conflict, the network node with the lowest 64-bit name value (i.e., the highest priority) gets the claimed address. The other network nodes must either claim different addresses or stop the communication. However, the **arbitrary address capable** parameter of a network node's J1939 name determines whether the node can claim a different address:

- **Arbitrary address capable** set to **1**: The network node can claim a different address.
- **Arbitrary address capable** set to **0**: The network node cannot claim a different address. In this case, it sends a *cannot claim address* message on the bus and stops its J1939 communication.

The Bus Manager supports the address claiming procedure as follows:

- A J1939-compliant network node that is simulated by the Bus Manager can participate in the address claiming procedure only if you enable J1939 network management for the network node. Refer to [Enabling and Disabling J1939 Network Management](#) on page 211.
- For PDUs that are manipulated by the Bus Manager, the source and/or destination addresses are adjusted at run time if the related network nodes have claimed a new transport protocol address.

**Handling J1939-compliant PDUs with up to 1,785 data bytes** To exchange J1939-compliant PDUs with up to 1,785 data bytes on the bus, transport protocols are used that segment the data of the PDU into multiple packets (*Data Transfer (DT)* messages). Each packet has an 8-byte data field, whose first byte is reserved for the sequence number of the packet. Thus, the data of a PDU can be segmented into up to 255 packets (1,785 / 7 bytes).

The exact behavior for segmenting and exchanging the PDUs on the bus depends on the used transport protocol. Two transport protocols are available:

Transport Protocol	Characteristics	Transmission Sequence
Broadcast Announce protocol	<ul style="list-style-type: none"> <li>▪ Communication mode: Broadcast</li> <li>▪ Transfer not acknowledged</li> </ul> <p>The sending network node does not know which network node receives the PDU and the receiving network node cannot influence the transfer.</p> <ul style="list-style-type: none"> <li>▪ Transfer duration: Up to 51 seconds (with a gap of up to 200 ms before each packet)</li> </ul>	<p>The sending network node first sends a <i>Broadcast Announce Message (BAM)</i>. The data field of the BAM contains the following information:</p> <ul style="list-style-type: none"> <li>▪ The PGN of the PDU.</li> <li>▪ The number of data bytes of the PDU.</li> <li>▪ The number of packets that are used to transmit the data bytes of the PDU on the bus.</li> </ul> <p>The BAM allows all receiving network nodes to prepare for reception. After sending the BAM, the sending network node starts the actual data transfer.</p>
Connection Mode Data Transfer protocol	<ul style="list-style-type: none"> <li>▪ Communication mode: Peer-to-peer</li> <li>▪ Transfer is acknowledged</li> </ul> <p>The sending network knows whether the receiving network node has received the PDU.</p> <ul style="list-style-type: none"> <li>▪ The receiving network node can request individual packets again.</li> <li>▪ The sending and the receiving network node can abort the transmission of an PDU at any time.</li> </ul>	<p>The sending network node first sends a <i>Request to Send (RTS)</i>. The receiving network node responds with either a <i>Clear to Send (CTS)</i> or a <i>Connection Abort</i> message if the connection cannot be established. When the sending network node receives the CTS, it starts the actual data transfer. The data transfer might be segmented, i.e., the receiving network sends further CTS to request data packets. After the receiving network node has successfully received the complete data, it sends an <i>End of Message Acknowledgment (EOM)</i>.</p>

## Multiplexed IPDUs

The Bus Manager supports CAN multiplexed IPDUs. For more information, refer to [Aspects of Supported AUTOSAR Features](#) on page 39.

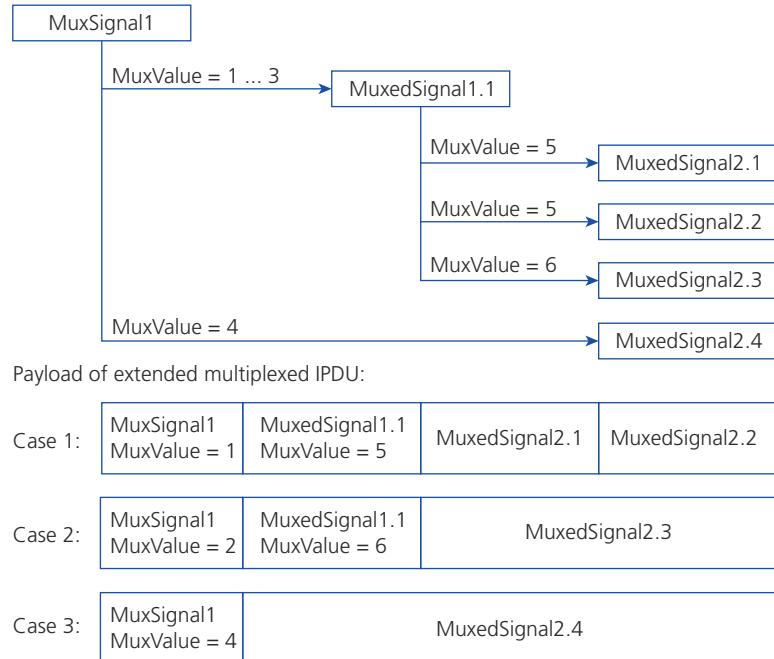
## Extended signal multiplexing

The Bus Manager supports extended signal multiplexing. Extended signal multiplexing is possible only for bus communication that is specified in DBC files.

In general, the value of a *multiplexer signal* can determine that a *multiplexed signal* is included in an IPDU. With extended signal multiplexing, a multiplexed signal itself can serve as a multiplexer signal, i.e., its value can determine other multiplexed signals to be included in the IPDU. The value that determines a specific multiplexed signal to be included in an IPDU is called *multiplexer value*. A multiplexed signal can be included in an IPDU due to one or more values and/or value ranges of the multiplexer value. However, a multiplexed signal is included in an IPDU only if the related multiplexer signal is also included in the IPDU.

In the Bus Manager, the multiplexer signals and multiplexed signals that are used for extended signal multiplexing are included in *extended multiplexed IPDUs*.

The following illustration is an example of a multiplexer signal (*MuxSignal1*) and multiplexed signals, where *MuxedSignal1.1* serves as a multiplexer signal itself. The signals that are included in the extended multiplexed IPDU depend on the multiplexer values (*MuxValue*) of *MuxSignal1* and *MuxedSignal1.1*.



If a multiplexed signal is not included in the extended multiplexed IPDU and this results in unused bits, these bits are filled with a default unused bit pattern, which is 255 for J1939-compliant PDUs and 0 for all other PDUs. If required, you can modify the communication matrix in the ConfigurationDesk application to specify a user-defined unused bit pattern. Refer to [Basics on Modifying Communication Matrices](#) on page 267.

## Limitations

When you work with a CAN bus system, some limitations apply. Refer to [Limitations for CAN Communication](#) on page 323.

## Aspects of Supported LIN Bus Features

### LIN specifications

The Bus Manager lets you work with LIN communication that complies with the following LIN specifications and SAE standards:

- LIN specification 1.3
- LIN specifications 2.0, 2.1, and 2.2
- SAE J2602 standard (protocol version: J2602\_1\_1.0, language version: J2602\_3\_1.0)

SAE J2602 is a vehicle LIN bus standard based on LIN 2.0 and defined by the Society of Automotive Engineers (SAE).

---

**Frame types**

The Bus Manager supports the following [frame](#) types:

- Unconditional frames

An unconditional frame is a standard LIN frame. The frame header sent by the [LIN master](#) is explicitly assigned to one frame response of a LIN node.

- Event-triggered frames

An event-triggered frame consists of a frame header that is assigned to several frame responses of different LIN nodes. A frame response is sent only if its data changed since its last transmission. This allows frame transmission of different LIN nodes via the same frame slot. If more than one LIN node tries to send a frame response, a collision occurs. Collisions have to be resolved by the LIN master (e.g., via collision resolver schedules).

- Sporadic frames

A sporadic frame references a list of frames. The position of a frame in the list determines the frame's priority. When a sporadic frame is scheduled, the referenced list is checked for frames containing at least one changed signal. The frame that contains a changed signal and has the highest priority is transmitted.

- Diagnostic frames

A diagnostic frame has eight data bytes and is either a master request frame (ID 60) or a slave response frame (ID 61). Diagnostic frames are used for slave node configuration and identification or for implementing diagnostic data transfer between a LIN master and LIN slaves.

---

**Checksum types**

The Bus Manager supports LIN frames with classic and enhanced checksum calculation types.

With the classic checksum calculation type, only the data bytes of a LIN frame are used to calculate the checksum. With the enhanced checksum calculation type, the data bytes and the protected identifier (PID) byte are used to calculate the checksum.

---

**LIN masters and slaves**

The Bus Manager supports the simulation of [LIN masters](#) and [LIN slaves](#). Simulating a LIN master supports the following LIN master tasks at run time:

- Send frame headers according to a schedule table that is defined in the communication matrix.
- Specify an initial schedule table that is started automatically after the executable application was started, and change the active schedule table during run time. For details, refer to [Working with LIN Schedule Tables](#) on page 207.
- Configure LIN slave nodes by transmitting master request frames with the following configuration commands:
  - AssignNAD
  - AssignFrameId
  - UnassignFrameId
  - AssignFrameIdRange

- ConditionalChangeNAD
- FreeFormat
- SaveConfiguration
- DataDump

To transmit the master request frames, they must be elements of the active schedule table. The configuration commands including their required parameters must be specified in the communication matrix and cannot be changed by the Bus Manager.

- Resolve collisions of event-triggered frames by automatically executing a related collision resolver schedule table. The collision resolver tables must be specified in the communication matrix for each event-triggered frame.

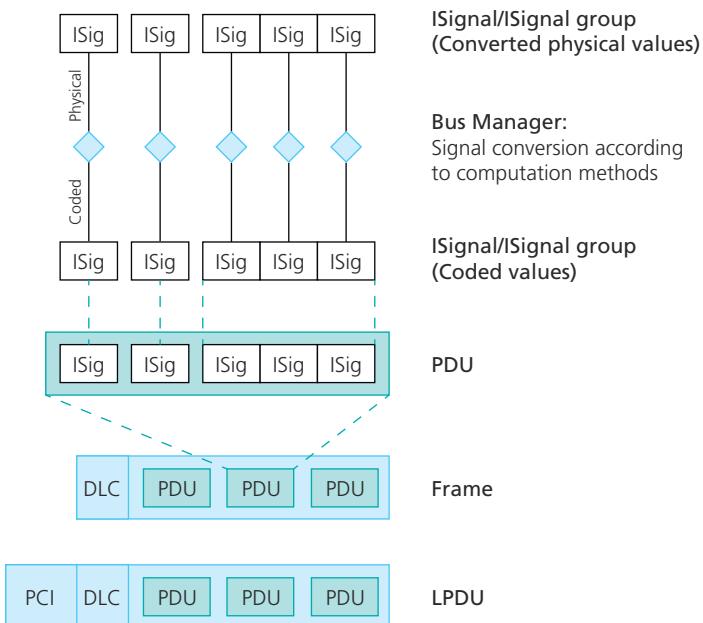
## Limitations

When you work with a LIN bus system, some limitations apply. Refer to [Limitations for LIN Communication](#) on page 325.

## Signal Conversion by the Bus Manager

### Transmitting and receiving ISignals on the bus

The Bus Manager accesses [ISignals](#) via their converted physical signal values (e.g., `EngineTempSig = 85 °C`). Because physical signal values cannot be transmitted via a bus, the Bus Manager converts physical signal values into coded signal values (e.g., `EngineTempSig = 0x17`) and includes the ISignals in a [PDU](#) (TX ISignals). When a PDU is received on the bus, the contained coded ISignals are extracted and their signal values are converted to physical values (RX ISignals).



## Converting ISignal values

Computation methods define the algorithm for converting coded ISignal values into physical values and vice versa. Normally, the [communication matrix](#) defines a computation method for each ISignal. If the communication matrix does not define a computation method for an ISignal, the Bus Manager uses a default computation method.

**Supported computation methods** Generally, computation methods can be specified in different ways, for example, as linear conversion or as a rational function of higher order:

$$y = \frac{Num\_V[0] + Num\_V[1] \cdot x + Num\_V[2] \cdot x^2 + Num\_V[3] \cdot x^3 + \dots + Num\_V[n] \cdot x^n}{Denom\_V[0] + Denom\_V[1] \cdot x + \dots + Denom\_V[n] \cdot x^n}$$

The Bus Manager only supports computation methods that define a linear computation scale, i.e.:

`y = Factor * x + Offset` (with `Factor ≠ 0`)

This also includes identical computation methods with `Factor = 1`, `Offset = 0`.

If a rational function is used as the computation method, the numerator and denominator arrays must be defined as follows:

- The numerator array must contain at least two elements and the second element must not be zero.
- If the numerator array contains a third element, it and all following elements must be zero.
- If a denominator array is defined, the first element must not be zero. If the array contains more elements, they must all be zero.

If no denominator array is defined, the default denominator value 1 is used.

Examples of supported computation scales:

```
Numerator_V[] = {0,1}
Denominator_V[] = {}
Numerator_V[] = {0,2.7,0,0,0}
Denominator_V[] = {3,0,0}
Numerator_V[] = {2,5.6,0,0}
Denominator_V[] = {4,0}
```

**Default computation method** If the communication matrix does not define a computation method for a signal, the Bus Manager uses an identical computation method by default.

Regarding the computation methods, some limitations apply. For details, refer to [Limitations for Communication Matrices and Communication Standards](#) on page 319.

## Determining the data type for physical signal values

If the communication matrix specifies data types for physical signal values, the Bus Manager uses these data types. If the communication matrix does not specify data types for physical signal values but only for coded ISignals, the Bus Manager determines the physical data type by using the computation method definitions of the communication matrix:

Definition in Communication Matrix	Determined Data Type of Physical Signal
No computation method defined	Same data type as coded data type
Linear computation method with <b>Factor</b> = 1 and <b>Offset</b> = 0	Same data type as coded data type
<ul style="list-style-type: none"> <li>▪ Supported computation method with integer values for <b>Factor</b> and <b>Offset</b></li> <li>▪ Data type for coded ISignals = integer</li> </ul>	<p>Smallest integer data type that is able to cover all possible physical signal values.</p> <p>The Bus Manager calculates the possible physical signal values according to the computation method and the signal length specified in the communication matrix.</p>
<ul style="list-style-type: none"> <li>▪ Supported computation method with floating-point values for <b>Factor</b> and/or <b>Offset</b></li> <li>▪ Data type for coded ISignals = integer</li> </ul>	Double

In case of an array signal, the Bus Manager does not determine the physical data type on the basis of computation methods. Instead, the Bus Manager uses the same data type as specified for the coded data type of the array signal.

Regarding the determination of data types for physical signal values, some limitations apply. For details, refer to [Limitations for Communication Matrices and Communication Standards](#) on page 319.

#### Tip

- When you select an ISignal (e.g., in the Buses Browser), the Properties Browser provides a Scale and Offset property that let you access the **Factor** and **Offset** value, respectively.
- Depending on the specifications in the communication matrix, you can specify user-defined settings for the numerator and denominator arrays. Refer to [Configurable Settings of ISignals](#) on page 284.

# Basics on the Bus Manager

## Where to go from here

## Information in this section

Working with Communication Matrices.....	58
Basics on Bus Configurations.....	63
Bus Configuration Tables.....	66
Bus Configuration Function Block.....	71
Assigning Communication Matrix Elements to Bus Configurations.....	75
Removing Communication Matrix Elements from Bus Configurations.....	83
Working with User Code.....	84
Implementing Secure Onboard Communication in Executable Applications.....	87
Implementing Global Time Synchronization in Executable Applications.....	88
Basics on Bus Access Requests.....	91
Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager.....	94
Accessing Bus Manager Features via ConfigurationDesk's Automation Interface.....	97

## Working with Communication Matrices

### Basics on communication matrices

Communication matrices define the communication of bus systems according to various standards. The design and file format of a communication matrix depend on the standard that is used to define the bus communication. Depending on the standard, communication matrices can be generated by various commercial off-the-shelf tools, by user-specific tools (e.g., OEM-specific database export tool) or other methods.

For an example of bus communication defined in a communication matrix, refer to [Introduction to Bus Communication Defined in Communication Matrices](#) on page 19.

#### Tip

If you do not have a communication matrix, you can use the `Basic_ComMatrix.arxml` communication matrix as a starting point to set up CAN communication. However, to use the communication matrix, you must modify it. For more information, refer to [Modifying Communication Matrices](#) on page 267.

### Supported file formats

The Bus Manager supports the following communication matrix file formats:

**AUTOSAR system description file** AUTOSAR (AUTomotive Open System ARchitecture) is an industry partnership that aims to develop and establish an open standard for automotive electrics/electronics (E/E) architectures.

AUTOSAR system description files are files of AUTOSAR XML type that describe 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. AUTOSAR system description files are instances of the AUTOSAR System Template. An AUTOSAR system description file usually describes more than one bus system (e.g., CAN and LIN).

The Bus Manager supports AUTOSAR system description files based on the following AUTOSAR Releases:

- AUTOSAR Release 3.2.1, 4.0.3, 4.2.1, 4.2.2, 4.3.0, 4.3.1, and 4.4.0
- AUTOSAR Classic Platform Release R19-11 and R20-11

**FIBEX file** The Field Bus Exchange (FIBEX) format is an XML exchange file format. It is used for data exchange between different tools that work with message-oriented bus communication. A FIBEX file usually describes more than one bus system.

The Bus Manager supports FIBEX files based on FIBEX 3.1.0, 4.1.1, and 4.1.2.

**DBC file** The Data Base Container (DBC) file format was primarily developed to describe a CAN bus system but it can also describe a LIN bus system. However, a DBC file can describe the communication of only one [communication cluster](#).

The Bus Manager does not support DBC files that describe a LIN bus system. For more information, refer to [Limitations for LIN Communication](#) on page 325.

**LDF file** The LIN Description File (LDF) file format was specially developed for LIN networks. An LDF file describes one LIN communication cluster and contains all the information necessary to configure it.

The Bus Manager supports LDF files based on LDF versions 1.3 to 2.2.

### Handling communication matrices

The Bus Manager lets you add communication matrices to the active [ConfigurationDesk application](#). There, you can assign them to [bus configurations](#), specify user-defined settings for configurable elements, and delete the communication matrices from the active application.

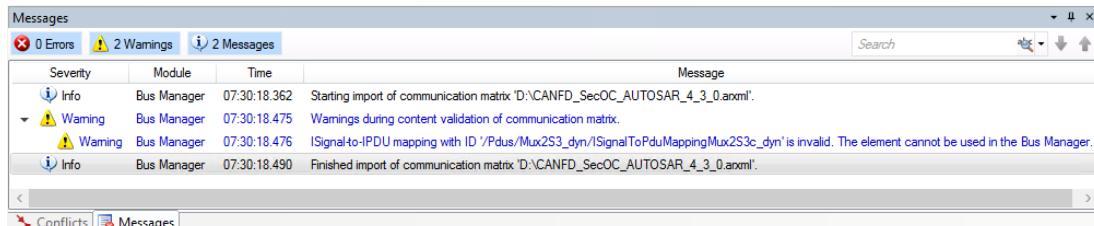
**Adding communication matrices** You can add one or more communication matrices to an active application. The communication matrices can have identical names but they must differ in content.

#### Tip

The Message Viewer provides information on the import process, e.g., whether the communication matrix is imported successfully, the import failed, or inconsistent settings are detected during the import.

The Bus Manager lets you add communication matrices to the active application without strict constraints. This means that you can even add communication matrices with inconsistent, conflicting, or unsupported settings. Depending on the affected communication matrix elements, this results in one of the following:

- The communication matrix is added to the active application but the affected communication matrix elements are discarded and not added. In this case, the [Message Viewer](#) provides information on the communication matrix elements that are not added, as shown in the following example.



- The affected communication matrix elements are added to the application but conflicts occur. For more information, refer to [Basics on Bus Manager-Related Conflicts](#) on page 99.

**Tip**

During the import, the Bus Manager checks whether all required ECUs that participate in CAN communication are part of the communication matrix. If a required ECU is missing, it is generated during the import. Refer to [Generating missing ECUs](#) on page 60.

**Assigning communication matrix elements to bus configurations** To be able to simulate, inspect, or manipulate bus communication during run time, you must implement it in the [signal chain](#). To do so, you must assign elements of an added communication matrix to a bus configuration. For more information, refer to [Assigning Communication Matrix Elements to Bus Configurations](#) on page 75.

**Specifying user-defined settings for configurable communication matrix elements** You can specify user-defined settings for configurable communication matrix elements, for example, to resolve conflicts. For details, refer to [Modifying Communication Matrices](#) on page 267.

**Deleting communication matrices** When you delete a communication matrix from the active application, you can decide whether to delete only the matrix or also matrix elements that are assigned to bus configurations.

- Deleting only the communication matrix

When you delete the communication matrix via the **Delete from Topology** command, only the communication matrix is deleted. If communication matrix elements are assigned to bus configurations, the assigned elements become unresolved. These elements and the associated higher-level elements are marked with a  symbol and are still visible in the **Buses Browser**.

If you generate bus simulation containers, conflicts concerning the unresolved elements occur and no code is generated for the affected bus configurations.

- Deleting the communication matrix and matrix elements that are assigned to bus configurations

When you delete the communication matrix via the **Delete Completely** command, matrix elements that are assigned to bus configurations are deleted as well, i.e., all the related bus configuration elements are removed. Because there are no unresolved elements, the communication matrix is no longer visible in the **Buses Browser** and no conflicts concerning the removed communication matrix occur.

**Generating missing ECUs**

If a communication matrix contains CAN PDUs for which no sending or receiving ECU is specified and/or a specified ECU is not part of the communication matrix, the required ECUs are generated during the import.

**ECUs generated for classic CAN and CAN FD PDUs** For classic CAN PDUs that are not J1939-compliant and for CAN FD PDUs, the generated ECUs are named `DS_UnknownSender` or `DS_UnknownReceiver`. The affected PDUs and their ISignals are assigned to the related sending or receiving ECU.

**ECUs generated for J1939-compliant PDUs** For J1939-compliant PDUs, the generated ECUs are named `DS_UnknownJ1939ECU_<transport protocol address in hex>`. The affected PDUs and ISignals are assigned to the generated ECUs as follows:

- TX PDUs and all their ISignals are assigned to a generated ECU if it is the sending ECU, i.e., the transport protocol address of the generated ECU is the transport protocol address that is specified as the source address of an affected PDU.
- Peer-to-peer RX PDUs and all their ISignals are assigned to a generated ECU if it is the receiving ECU, i.e., the transport protocol address of the generated ECU is the transport protocol address that is specified as the destination address of an affected peer-to-peer PDU.
- Broadcast RX PDUs can be received by any [network node](#) in a communication cluster. Therefore, no specific receiving ECU is specified for broadcast PDUs. However, to handle broadcast PDUs, the Bus Manager generates a receiving ECU in the following cases:
  - The communication matrix does not assign any RX ISignals to a broadcast PDU.
  - For at least one RX ISignal of a broadcast PDU, the communication matrix does not specify a receiving ECU, or the receiving ECU is not part of the communication matrix.

In these cases, the generated ECU is named `DS_UnknownJ1939ECU_0xFF` and all affected broadcast RX PDUs are assigned to this ECU. However, related ISignals are assigned to this ECU only if the communication matrix does not specify a receiving ECU for an ISignal, or the receiving ECU is not part of the communication matrix.

#### Tip

J1939 bridge ECUs are a typical use scenario in which ECUs are missing in DBC communication matrices. Bridge ECUs route J1939-compliant PDUs from one communication cluster to another by using the transport protocol address of the original sending and/or receiving ECUs. Because a DBC file specifies the communication of only one communication cluster, the original sending and/or receiving ECUs are not available in the DBC file.

For more information on the J1939 protocol, refer to [J1939 protocol](#) on page 48.

---

#### Clusters and ECUs view

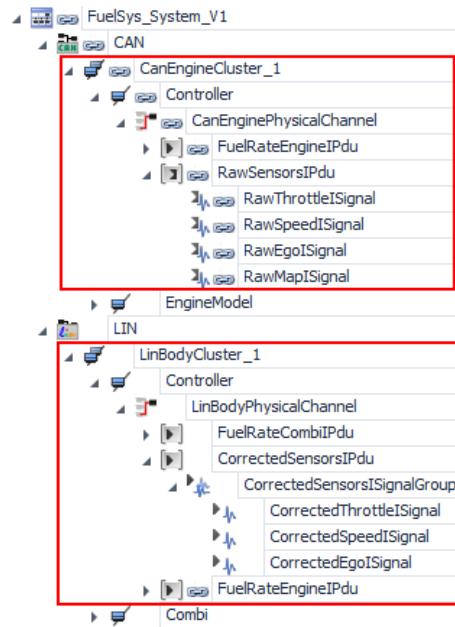
The Bus Manager provides different views on communication matrix elements. You can switch between the following views:

- Clusters view
- ECUs view

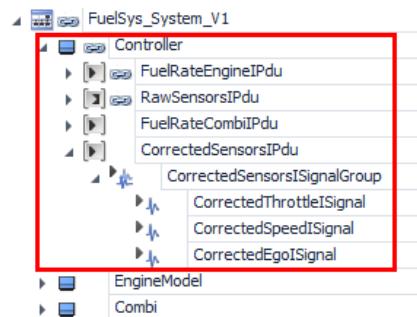
By default, the communication matrix elements are sorted by clusters.

**Clusters view** In the clusters view, all the [communication clusters](#) defined in the communication matrix are displayed and sorted by the bus system (CAN, LIN). For each cluster, the [network nodes](#) with the relevant physical channels, [PDUs](#), and [ISignals](#) are displayed.

The following illustration is an example of the clusters view. The 'Controller' ECU is a member of the 'CanEngineCluster\_1' and 'LinBodyCluster\_1' clusters. For both clusters, the 'Controller' network node is displayed with the cluster-relevant parts of the ECU communication. PDUs that are transmitted and received via both clusters are displayed for each of the clusters.



**ECUs view** In the ECUs view, the bus communication defined in the communication matrix is sorted by [ECUs](#). All the ECUs with all their PDUs and ISignals are displayed, regardless of which bus system or cluster they belong to. PDUs that are transmitted or received via multiple clusters are displayed only once for the transmitting or receiving ECU.



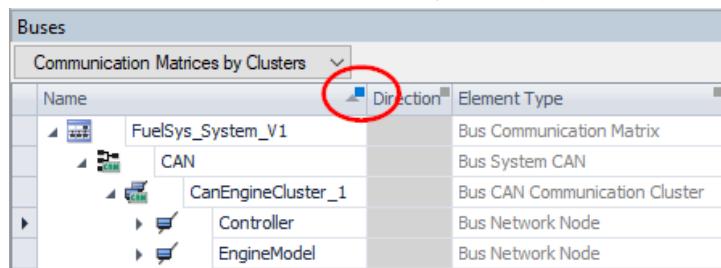
## Filtering communication matrix elements

For a better overview, you can filter the communication matrix elements in the Buses Browser. You can do so via context menu, for example. You can filter for:

- Used elements
- Unused elements
- Unresolved elements

- Changed communication matrix elements
- Names or regular expressions (by specifying column filters)

You can use several filters in parallel. The selected filters are active until you deactivate them explicitly. Active element filters are highlighted in the Home ribbon, active column filters are marked by a blue square.



Name	Direction	Element Type
FuelSys_System_V1		Bus Communication Matrix
CAN		Bus System CAN
CanEngineCluster_1		Bus CAN Communication Cluster
Controller		Bus Network Node
EngineModel		Bus Network Node

### Tip

- If an added communication matrix is not displayed or you miss communication matrix elements, make sure that all the filters are deactivated.
- You can use the filters and the Select Elements by Type command to easily select multiple similar communication matrix elements at once. For an example, refer to [Example of Selecting Multiple Similar Communication Matrix Elements at Once](#) on page 247.

For details on using the filters, refer to [Using Display Filters \(ConfigurationDesk Real-Time Implementation Guide\)](#).

## Limitations

Regarding the specifications of communication matrices, some limitations apply. Refer to [Limitations for Communication Matrices and Communication Standards](#) on page 319.

## Basics on Bus Configurations

### Introduction

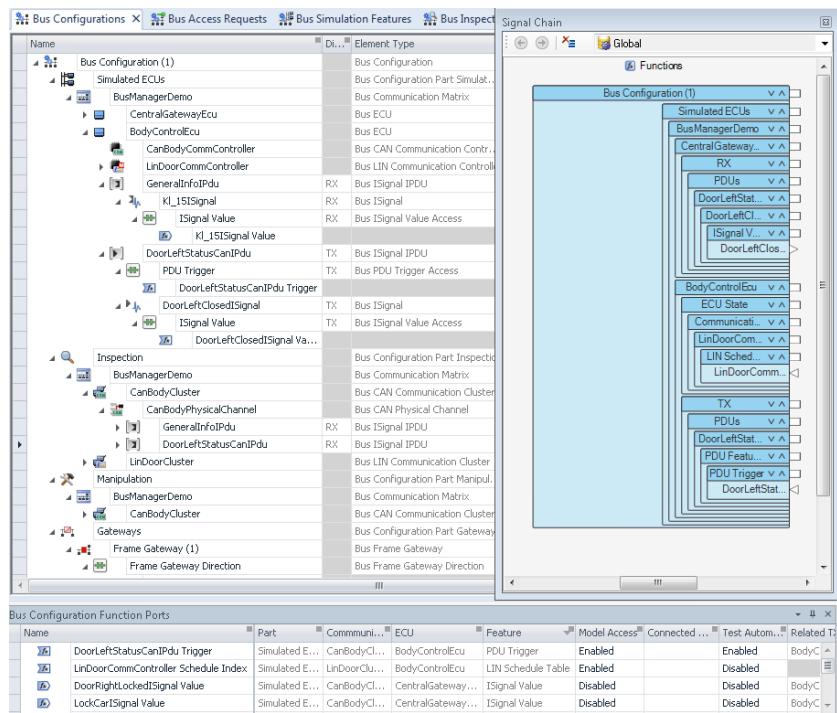
Bus configurations let you implement bus communication in the [signal chain](#) of ConfigurationDesk and configure it for simulation, inspection, and/or manipulation purposes. Additionally, you can specify gateways for CAN communication between [communication clusters](#).

### Using multiple bus configurations

You can work with multiple bus configurations at the same time. This makes it possible for you to use a separate bus configuration, e.g., for each communication cluster or each [ECU](#) you want to simulate.

## Accessing bus configurations

When you add a bus configuration to the signal chain, you can access it via specific tables and its Bus Configuration function block. The tables provide access to the complete structure of the bus configurations (e.g., to the [ISignals](#) you want to simulate, their function ports, the bus access requests, etc.) and they let you configure the bus communication for simulation, inspection, and/or manipulation purposes, and specify gateways. The Bus Configuration function block can be accessed via working views and provides a structured overview of the ports of a bus configuration.



For more information, refer to:

- [Bus Configuration Tables](#) on page 66
- [Bus Configuration Function Block](#) on page 71

## Assigning communication matrix elements

To simulate, inspect, and/or manipulate bus communication, you must assign communication matrix elements to different parts of one or more bus configurations. Refer to [Assigning Communication Matrix Elements to Bus Configurations](#) on page 75.

## Removing communication matrix elements from bus configurations

You can remove an assigned communication matrix element from a bus configuration by simply deleting it from the bus configuration. Refer to [Removing Communication Matrix Elements from Bus Configurations](#) on page 83.

**Specifying gateways**

To exchange CAN communication between CAN communication clusters, you can specify gateways. Refer to [Specifying CAN Frame Captures and Gateways](#) on page 107.

**Element identification via node names**

The node names in a bus configuration structure are used to identify bus communication elements unambiguously within the structure. The names are used in the variable description file (TRC file) and can, for example, be used to access ISignals via automation scripts.

The node names of the bus configuration and the assigned communication matrices are configurable. If you work with DBC or LDF communication matrices, you can also configure the names of communication cluster nodes in the bus configuration.

It is recommended that you specify meaningful names without version or date information for these nodes. This lets you, for example, replace an assigned communication matrix without having to adapt test automation scripts later on.

**Tip**

- DBC and LDF communication matrices specify the communication of only one communication cluster. Because the matrices cannot specify a name for this cluster, the cluster name is derived from the communication matrix name when you add the matrix to the ConfigurationDesk application. Therefore, the cluster name might contain version or date information.
- Communication cluster nodes are available in a bus configuration when you configure bus communication for inspection or manipulation purposes. Nevertheless, you can access the cluster name via the Properties Browser when you select a PDU that is assigned to any part of a bus configuration.

**Configuring bus communication**

You can configure the bus communication of each bus configuration via various bus configuration features. Refer to [Working with Bus Configuration Features](#) on page 115.

**Updating bus configurations**

Typically, the bus communication defined in a communication matrix is frequently changed during the development process. Communication matrices cannot be updated in the ConfigurationDesk application but you can import the modified communication matrix regardless of whether its name has changed. The Bus Manager updates a bus configuration when you replace the assigned communication matrix by the modified communication matrix. Refer to [Replacing Assigned Communication Matrices](#) on page 291.

<b>Including bus communication in offline or real-time simulation</b>	To include bus communication in offline or real-time simulation, you can generate bus simulation containers. You can import bus simulation containers to the VEOS Player or ConfigurationDesk and implement the contained bus communication in an offline simulation application or a real-time application, respectively. Refer to <a href="#">Basics on Bus Simulation Containers</a> on page 253.
---	--

**Related topics****Basics**

[Limitations for Bus Configuration Handling](#)..... 328

## Bus Configuration Tables

**Introduction**

The bus configuration tables are the main elements for accessing and working with bus configurations:

- They let you assign communication matrix elements to bus configurations.
- They let you configure the bus communication for simulation, inspection, and/or manipulation purposes.
- They let you specify gateways.
- They provide access to all the elements and properties of bus configurations.

You can customize the display of the tables according to your requirements (e.g., by using display filters, selecting elements by type, or sorting or removing columns). Some of the changes, such as removed columns, apply to all ConfigurationDesk projects and are stored, even if you close the tables or the Bus Manager. For more information, refer to [Customizing View Sets \(ConfigurationDesk Real-Time Implementation Guide\)](#).

The following tables are specific for bus configurations:

- [Bus Configurations table](#) on page 66
- [Bus Access Requests table](#) on page 69
- [Bus Simulation Features table, Bus Inspection Features table, and Bus Manipulation Features table](#) on page 70
- [Bus Configuration Ports table](#) on page 71

**Tip**

If a table is closed, you can open it via Switch Controlbars on the View ribbon, for example.

**Bus Configurations table**

This table provides access to all the bus configurations of the active ConfigurationDesk application. The following illustration is an example of the

Bus Configurations table structure for an application with two bus configurations named 'Combi\_BusConfiguration' and 'Controller\_BusConfiguration'.

Name	Dir...	Element Type	Related Clusters	Related TX ECUs	Related RX ECUs	Connected Model Ports
Combi_BusConfiguration		Bus Configuration				
Simulated ECUs		Bus Configuration Part Simulated ECUs				
Inspection		Bus Configuration Part Inspection				
Manipulation		Bus Configuration Part Manipulation				
Gateways		Bus Configuration Part Gateways				
Controller_BusConfiguration		Bus Configuration				
Simulated ECUs		Bus Configuration Part Simulated ECUs				
BusManagerDemo		Bus Communication Matrix				
BodyControlEcu		Bus ECU	CanBodyCluster; LinD...			
CanBodyCommController		Bus CAN Communication Controller	CanBodyCluster			
LinDoorCommController		Bus LIN Communication Controller	LinDoorCluster			
DoorLeftStatusCanIPdu	TX	Bus ISignal IPDU	CanBodyCluster	BodyControlEcu	CentralGatewayEcu	
DoorLeftClosedISignal	TX	Bus ISignal	CanBodyCluster	BodyControlEcu	CentralGatewayEcu	
ISignal Value	TX	Bus ISignal Value Access	CanBodyCluster	BodyControlEcu	CentralGatewayEcu	
DoorRightStatusIPdu	RX	Bus ISignal IPDU	LinDoorCluster	DoorEcuRight	BodyControlEcu	
Inspection		Bus Configuration Part Inspection				
BusManagerDemo	RX	Bus Communication Matrix				
CanBodyCluster	RX	Bus CAN Communication Cluster				
CanBodyPhysicalChannel	RX	Bus CAN Physical Channel				
DoorLeftStatusCanIPdu	RX	Bus ISignal IPDU	CanBodyCluster	BodyControlEcu	CentralGatewayEcu	
LinDoorCluster	RX	Bus LIN Communication Cluster				
Frame Capture (1)	RX	Bus Frame Capture				
Frame Capture Data	RX	Bus Frame Capture Data Inspection				
CAN Cluster Filter	RX	Bus Frame Capture Filter				
Manipulation		Bus Configuration Part Manipulation				
BusManagerDemo	TX	Bus Communication Matrix				
CanBodyCluster	TX	Bus CAN Communication Cluster				
CanBodyPhysicalChannel	TX	Bus CAN Physical Channel				
DoorLeftStatusCanIPdu	TX	Bus ISignal IPDU	CanBodyCluster	BodyControlEcu	CentralGatewayEcu	
LinDoorCluster	TX	Bus LIN Communication Cluster				
Gateways		Bus Configuration Part Gateways				
Frame Gateway (1)		Bus Frame Gateway				
Frame Gateway Direction		Bus Frame Gateway Direction				
CAN Cluster 1 Filter		Bus Frame Gateway Filter				
CAN Cluster 2 Filter		Bus Frame Gateway Filter				

**Bus configuration parts** Each bus configuration consists of five bus configuration parts. The Bus Configurations table provides access to the following four parts:

Bus Configuration Part	Description
Simulated ECUs	Lets you assign communication matrix elements to the bus configuration for simulation purposes. Simulating bus communication is ECU-dependent. Therefore, elements are assigned in the context of their sending or receiving ECUs. You can select individual elements to access their properties in the Properties Browser and add available bus configuration features.
Inspection	<ul style="list-style-type: none"> <li>Lets you assign communication matrix elements to the bus configuration for inspection purposes. Inspecting bus communication is cluster-dependent, i.e., ECU-independent. Therefore, elements are assigned in the context of their communication cluster <a href="#">?</a>.</li> <li>Lets you add frame captures to capture CAN frames that are received on the bus. Frame captures are independent from communication matrices, i.e., frames are captured on the basis of specified frame parameters and not on the basis of communication matrix elements.</li> </ul> <p>You can select individual elements to access their properties in the Properties Browser and add available bus configuration features.</p>

Bus Configuration Part	Description
Manipulation	Lets you assign communication matrix elements to the bus configuration for manipulation purposes. Manipulating bus communication is cluster-dependent, i.e., ECU-independent. Therefore, elements are assigned in the context of their communication cluster. You can select individual elements to access their properties in the Properties Browser and add available bus configuration features.
Gateways	Lets you specify gateways. Via a gateway, you can exchange CAN communication between two communication clusters. Specifying gateways is independent from communication matrices, i.e., you cannot assign communication matrix elements to this part. Instead, you must add a frame gateway for each gateway you want to specify.

**Tip**

- The fifth part of each bus configuration is the Bus Access Requests part. You can access this part via the Bus Access Requests table.
- All the communication matrix elements that are assigned to a bus configuration are displayed hierarchically, starting with the communication matrix they are part of. Therefore, a communication matrix can be represented by up to four communication matrix nodes, one below each Simulated ECUs, Inspection, Manipulation, and Bus Access Requests node. Changes you make to one communication matrix node automatically apply to the other nodes.
- When you drag communication matrix elements to a bus configuration node, the elements are automatically assigned to the Simulated ECUs part of the related bus configuration.

**Further information**

- For an overview of the use scenarios for simulating, inspecting, manipulating, and gatewaying bus communication, refer to [Use Scenarios for Configuring Bus Communication with the Bus Manager](#) on page 20.
- For basic information on assigning communication matrix elements for simulation, inspection, and manipulation purposes to bus configurations, refer to [Assigning Communication Matrix Elements to Bus Configurations](#) on page 75.
- For basic information on capturing CAN frames, refer to [Capturing CAN Frames](#) on page 107.
- For basic information on specifying gateways, refer to [Specifying CAN Gateways](#) on page 109.
- For more information on this table, refer to [Bus Configurations Table \(ConfigurationDesk User Interface Reference\)](#).
- For an overview of all the available symbols, refer to [Elements and Symbols of Bus Configurations and Communication Matrices \(ConfigurationDesk User Interface Reference\)](#).

**Bus Access Requests table**

This table provides access to all the [bus access requests](#) of all bus configurations of the active ConfigurationDesk application. Bus access requests are generated for the following elements:

- One bus access is generated for each pair of a [communication cluster](#) and the Simulated ECUs, Inspection, or Manipulation part of a bus configuration if at least one cluster element is assigned to the related part.
- One bus access request is generated for each frame capture element that is added to the Inspection part of a bus configuration.
- Two bus access requests are generated for each frame gateway that is added to the Gateways part of a bus configuration.

You can specify user-defined names for the bus access requests. However, with the Bus Manager (stand-alone), you cannot use the full functionality of this table. In general, this table can be used to assign bus access requests to [bus accesses](#), i.e., to bus function blocks (CAN, LIN) and real-time hardware. This is not possible with the Bus Manager (stand-alone). Nevertheless, bus access requests are included in [bus simulation containers](#).

The following illustration is an example of the Bus Access Requests table structure for an application with two bus configurations named 'Combi\_BusConfiguration' and 'Controller\_BusConfiguration'.

The screenshot shows the 'Bus Access Requests' table interface. The left pane displays a hierarchical tree of bus configurations and their components. The right pane shows a detailed properties view for a selected 'Bus Access Request' item, specifically for 'Controller\_BusConfiguration'.

Name	Element Type	Bus Configuration	Bus Configuration Part
Combi_BusConfiguration	Bus Configuration	Combi_BusConfiguration	Bus Configuration
Bus Access Requests	Bus Configuration Part Bus Access Requests	Combi_BusConfiguration	Bus Configuration
Controller_BusConfiguration	Bus Configuration	Controller_BusConfiguration	Bus Configuration
Bus Access Requests	Bus Configuration Part Bus Access Requests	Controller_BusConfiguration	Controller_BusConfiguration
BusManagerDemo	Bus Communication Matrix	Controller_BusConfiguration	Controller_BusConfiguration
CAN	Bus System CAN	Controller_BusConfiguration	Controller_BusConfiguration
CanBodyCluster	Bus CAN Communication Cluster	Controller_BusConfiguration	Controller_BusConfiguration
Bus Access Request [Controller_BusConfiguration]Simulated ECU[CanBodyCluster]	Bus Access Request Simulated ECUs	Controller_BusConfiguration	Simulated ECUs
CanBodyCluster (1)			
Bus Access Request [Controller_BusConfiguration]Inspection[CanBodyCluster (1)]	Bus Access Request Inspection	Controller_BusConfiguration	Inspection
Bus Access Request [Controller_BusConfiguration]Manipulation[CanBodyCluster (1)]	Bus Access Request Manipulation	Controller_BusConfiguration	Manipulation
LIN	Bus System LIN	Controller_BusConfiguration	Controller_BusConfiguration
LinDoorCluster	Bus LIN Communication Cluster	Controller_BusConfiguration	Controller_BusConfiguration
Bus Access Request [Controller_BusConfiguration]Simulated ECU[LinDoorCluster]	Bus Access Request Simulated ECUs	Controller_BusConfiguration	Simulated ECUs
LinDoorCluster (1)			
Bus Access Request [Controller_BusConfiguration]Inspection[LinDoorCluster (1)]	Bus Access Request Inspection	Controller_BusConfiguration	Inspection
Bus Access Request [Controller_BusConfiguration]Manipulation[LinDoorCluster (1)]	Bus Access Request Manipulation	Controller_BusConfiguration	Manipulation
Frame Capture (1)	Bus Frame Capture	Controller_BusConfiguration	Controller_BusConfiguration
CAN	Bus System CAN	Controller_BusConfiguration	Controller_BusConfiguration
CAN Cluster	Bus CAN Frame Capture Cluster	Controller_BusConfiguration	Controller_BusConfiguration
Bus Access Request [Controller_BusConfiguration]Frame Capture (1)[CAN Cluster]	Bus Access Request Inspection	Controller_BusConfiguration	Inspection
Frame Gateway (1)	Bus Frame Gateway	Controller_BusConfiguration	Controller_BusConfiguration
CAN	Bus System CAN	Controller_BusConfiguration	Controller_BusConfiguration
CAN Cluster 1	Bus CAN Frame Gateway Cluster	Controller_BusConfiguration	Controller_BusConfiguration
Bus Access Request [Controller_BusConfiguration]Frame Gateway (1)[CAN Cluster 1]	Bus Access Request Gateways	Controller_BusConfiguration	Gateways
CAN Cluster 2	Bus CAN Frame Gateway Cluster	Controller_BusConfiguration	Controller_BusConfiguration

**Properties**

**General**

Name	Value
Bus Access Request [Controller_BusConfiguration]	
Identification	
Manual naming	<input type="checkbox"/>
Bus Access Assignment	
Bus access	CAN [CanBodyCluster (1)]
Bus Access Request	
Required baud rate	500E+03 Baud
Requires CAN FD	<input type="checkbox"/>
Required data phase baud rate	Baud

**Further information**

- For more information on bus access requests and bus accesses, refer to [Basics on Bus Access Requests](#) on page 91.
- For more information on this table, refer to [Bus Access Requests Table](#) ([ConfigurationDesk User Interface Reference](#)).

- For an overview of all the available symbols, refer to [Elements and Symbols of Bus Configurations and Communication Matrices \(ConfigurationDesk User Interface Reference\)](#).

**Bus Simulation Features table, Bus Inspection Features table, and Bus Manipulation Features table**

Bus configurations provide various bus configuration features, which are available for specific elements and for simulation, inspection, and/or manipulation purposes.

The Bus Simulation Features table, Bus Inspection Features table, and Bus Manipulation Features table provide access to the bus configuration features that can be added to PDUs and ISignals for simulation, inspection, and manipulation purposes, respectively. Each table lets you enable and disable PDU-related and signal-related bus configuration features for all the bus configurations of the active ConfigurationDesk application. Depending on the enabled bus configuration features, function ports and/or event ports are added to the related bus configuration. You can access the ports via the Bus Configurations and Bus Configuration Ports table, for example.

**Tip**

The Bus Simulation Features, Bus Inspection Features, and Bus Manipulation Features table provide access only to bus configuration features that are available for PDUs and ISignals. To access bus configuration features that are available for other bus configuration elements, e.g., communication controllers, ISignal groups, global time domains, you must right-click the related elements in the Bus Configurations table.

The Bus Simulation Features, Bus Inspection Features, and Bus Manipulation Features table provide the following subviews:

Subview	Purpose
PDU Features	Provides a list of all the PDUs that are assigned to the related bus configuration part and the available PDU-related bus configuration features. This is the default view.
Signal Features	Provides a list of all the ISignals that are assigned to the related bus configuration part and the available signal-related bus configuration features.

The following illustration is an example of the Bus Simulation Features table, displaying the PDU Features subview.

Bus Simulation Features													
PDU Features													
Name	Element Type	Direction	Bus Configuration	Communication Matrix	ECU	PDU Raw Data	PDU User Code	User Code ID	PDU Trigger	Trigger Mode	Frame Access	Maximum Length	PDU Enabled
FuelRateCombiPdu	Bus Signal PDU	RX	Combi_BusConf...	FuelSys_System	Combi	Enabled	Disabled						
CorrectedSensorsPdu	Bus ISignal PDU	RX	Combi_BusConf...	FuelSys_System	Combi	Disabled	Enabled						
RawSensorsPdu	Bus ISignal PDU	RX	Controller_BusC...	FuelSys_System	Controller	Disabled	Disabled				Enabled	64 byte	
FuelRateEnginePdu	Bus ISignal PDU	TX	Controller_BusC...	FuelSys_System	Controller	Enabled	Disabled		Enabled	Positive edge	Disabled		Enabled

**Further information**

- For more information on working with bus configuration features, refer to [Working with Bus Configuration Features](#) on page 115.

- For more information on these tables, refer to
  - [Bus Simulation Features Table \(ConfigurationDesk User Interface Reference\)](#)
  - [Bus Inspection Features Table \(ConfigurationDesk User Interface Reference\)](#)
  - [Bus Manipulation Features Table \(ConfigurationDesk User Interface Reference\)](#)

### Bus Configuration Ports table

Bus configurations provide [function ports](#) and/or [event ports](#) for enabled bus configuration features. The ports let you access ISignal values, LIN schedule tables, PDU raw data, etc.

This table provides access to the available ports of all the bus configurations of the active ConfigurationDesk application. In the table, you can configure the ports: For example, you can enable Model access for several function ports at the same time or map [model ports](#) to function ports or event ports via drag & drop. For a better overview, you can filter this table via various port filters. When you select a port in the list, all the properties of the port are displayed in the [Properties Browser](#).

Name	Part	Communication ...	ECU	Feature	Model Access	Connected Model Ports	Test Autom...	Related TX E...	Related R...
RawSensorsIPdu_RawData	Simulated ECUs	CanEngineCluster_1	Controller	PDU Raw Data	Enabled	RawSensorsIPdu_Ra...	Disabled	EngineModel	Controller
RawThrottleISignal_Value	Simulated ECUs	CanEngineCluster_1	Controller	ISignal Value	Disabled		Disabled	EngineModel	Controller
FuelRateEngineIPdu_Trigger	Simulated ECUs	CanEngineCluster_1	Controller	PDU Trigger	Disabled		Disabled	Controller	EngineMo
FuelRateEngineIPdu_RawData	Simulated ECUs	CanEngineCluster_1	Controller	PDU Raw Data	Enabled	FuelRateEngineIPdu_...	Disabled	Controller	EngineMo
FuelRateEngineIPdu_Enable	Simulated ECUs	CanEngineCluster_1	Controller	PDU Enable	Disabled		Disabled	Controller	EngineMo
FuelRateEngineIPdu_TimingControlP...	Simulated ECUs	CanEngineCluster_1	Controller	PDU Cyclic Timing Control	Disabled		Disabled	Controller	EngineMo
FuelRateEngineIPdu_TimingControlO...	Simulated ECUs	CanEngineCluster_1	Controller	PDU Cyclic Timing Control	Enabled	FuelRateEngineIPdu_...	Disabled	Controller	EngineMo
FuelRateEngineISignal_Value	Simulated ECUs	CanEngineCluster_1	Controller	ISignal Value	Disabled		Disabled	Controller	Controller
LinBodyCommController_ScheduleIndex	LinBodyCluster_1	Controller		LIN Schedule Table	Disabled		Disabled		
FuelRateEngineISignal_Value	Inspection	CanEngineCluster_1	EngineModel; ...	ISignal Value	Enabled	FuelRateEngineISigna...	Disabled	Controller	EngineMo
FuelRateEngineIPdu_Counter	Inspection	CanEngineCluster_1	EngineModel; ...	PDU RX Status	Disabled		Disabled	Controller	EngineMo
FuelRateEngineIPdu_State	Inspection	CanEngineCluster_1	EngineModel; ...	PDU RX Status	Disabled		Disabled	Controller	EngineMo
FuelRateEngineIPdu_Time	Inspection	CanEngineCluster_1	EngineModel; ...	PDU RX Status	Disabled		Disabled	Controller	EngineMo
FuelRateEngineIPdu_DeltaTime	Inspection	CanEngineCluster_1	EngineModel; ...	PDU RX Status	Disabled		Disabled	Controller	EngineMo

### Further information

- For more information on configuring function ports, refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.
- For an example of mapping model ports to function ports or event ports via this table, refer to [Example of Mapping Model Ports to Bus Configuration Ports via Tables](#) on page 249.
- For more information on filtering elements, refer to [Using Display Filters \(ConfigurationDesk Real-Time Implementation Guide\)](#).
- For more information on this table, refer to [Bus Configuration Ports Table \(ConfigurationDesk User Interface Reference\)](#).

## Bus Configuration Function Block

### Introduction

In working views, each bus configuration can be accessed via its Bus Configuration function block.

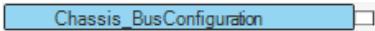
**Port-based appearance of Bus Configuration function blocks**

The appearance of Bus Configuration function blocks is port-based, i.e., each function block provides a hierarchically structured access to the ports of a bus configuration. Because the available ports and the related hierarchies depend on the configured bus communication, the appearance of each Bus Configuration function block depends on the bus communication that is configured for the related bus configuration.

---

**Default view of Bus Configuration function blocks**

By default, bus configurations do not provide any ports. Therefore, a default Bus Configuration function block displays only the name of the bus configuration.

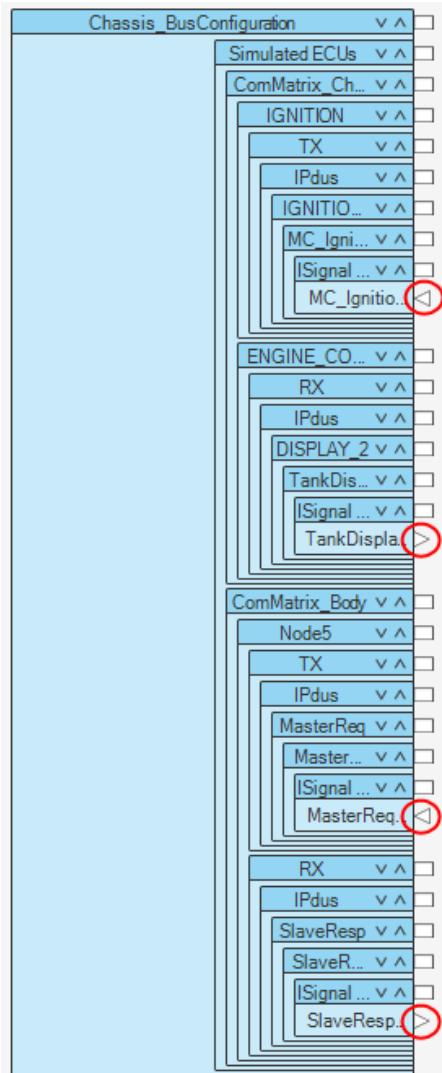


**Function ports of Bus Configuration function blocks**

Function ports are available when you configure bus communication by using bus configuration features that provide function ports. In many cases, the bus communication that is configured for a bus configuration results in hundreds of signals that can be accessed via function ports. Therefore, Bus Configuration function blocks can provide hundreds of function ports.

To improve the overview, function ports with disabled model access are hidden by default. The related hierarchical structures are also hidden. Because model access is disabled for all the function ports by default, Bus Configuration function blocks display no function ports by default.

When you enable model access for function ports, the function ports and the related hierarchical structures are displayed. In the following example, model access is enabled for four function ports.

**Tip**

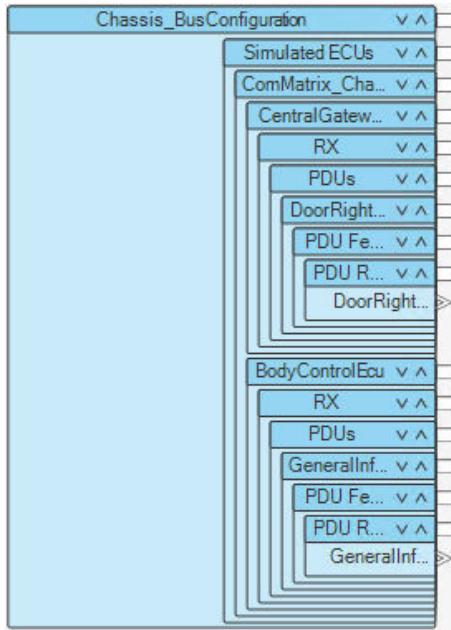
To display function ports with disabled model access, you must deactivate the filter for mappable ports. To do so, right-click a free area of the working view and clear **Filter for Mappable Ports** in the context menu. The function ports with disabled model access are indicated by and symbols.

For more information on working with bus configuration features and configuring function ports, refer to [Basics on Working with Bus Configuration Features](#) on page 116.

#### Event ports of Bus Configuration function blocks

Event ports are available only if you add bus configuration features that provide event ports to the bus configuration. If you do this, the related Bus

Configuration function block displays the event ports, including their hierarchical structure, as shown in the following example.



### Tip

- In contrast to function ports, all the available event ports are displayed by default.
- When you configure bus communication, events are created automatically. However, for these events no event ports are available. For more information, refer to [Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager](#) on page 94.

### Default view of Bus Configuration function blocks providing a high number of mappable ports

When you open a working view that contains a Bus Configuration function block with a high number of mappable ports (i.e., function ports with enabled model access and/or event ports), the function block is collapsed by default. To expand the function block, click the expand arrow.



If you click the expand arrow once and ports of the Bus Configuration function block are mapped to model ports, the mapping lines are still collapsed and drawn to the parent port of the function block. To expand the mapping lines as well, click the expand arrow twice.

# Assigning Communication Matrix Elements to Bus Configurations

## Overview

To simulate, inspect, and/or manipulate bus communication, you must assign communication matrix elements to specific bus configuration parts:

- To simulate bus communication, assign communication matrix elements to the **Simulated ECUs** bus configuration part.
- To inspect bus communication, assign communication matrix elements to the **Inspection** bus configuration part.
- To manipulate bus communication, assign communication matrix elements to the **Manipulation** bus configuration part.

The bus configuration parts do not influence each other. Therefore, you can configure the bus communication independently for simulation, inspection, and manipulation purposes.

You can access the bus configuration parts in the **Bus Configurations** table.

Name	Direction	Element Type
Combi_BusConfiguration		Bus Configuration
Simulated ECUs		Bus Configuration Part Simulated ECUs
Inspection		Bus Configuration Part Inspection
Manipulation		Bus Configuration Part Manipulation
Gateways		Bus Configuration Part Gateways
Controller_BusConfiguration		Bus Configuration

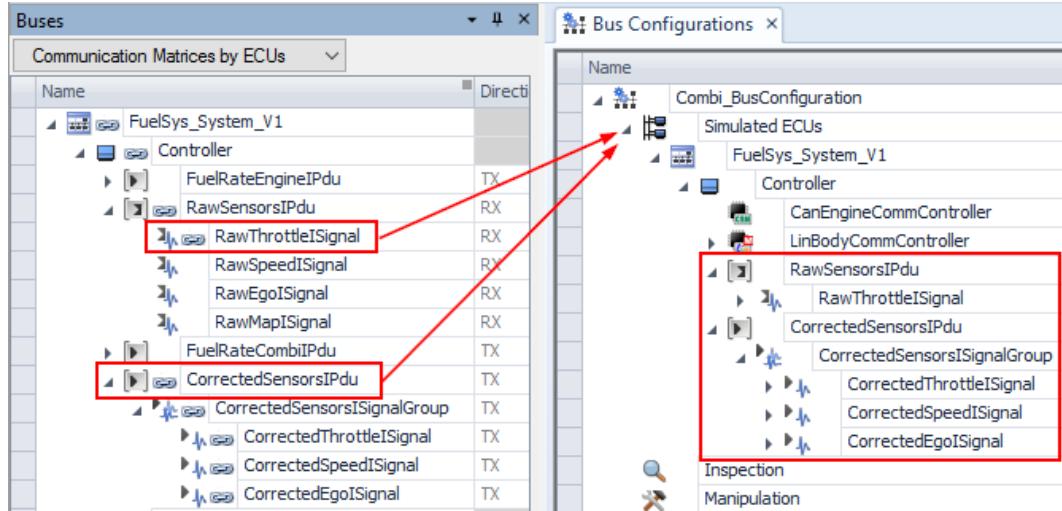
In the **Buses Browser**, the assigned communication matrix elements are marked by a chain symbol (🔗). These elements are part of the signal chain and are implemented in [bus simulation containers](#) when you generate the containers.

## General assignment rules

You can assign communication matrices completely or in part to bus configurations by dragging communication matrix elements from the **Buses Browser** to bus configurations. When you assign a communication matrix element to a bus configuration, the following general assignment rules apply:

- Relevant higher-level elements are assigned as well.
- Relevant subelements are assigned as well.
- Elements of the same hierarchy level are not assigned.

In the following example, 'RawThrottleSignal' and 'CorrectedSensorPdu' are assigned to a bus configuration. For 'RawThrottleSignal', the related higher-level [ISignal IPDU](#) is assigned as well. The [ISignals](#) on the same hierarchy level as 'RawThrottleSignal' are not assigned. For 'CorrectedSensorPdu', all its subelements (i.e. the ISignal group and the ISignals) are assigned as well.



Which higher-level elements and subelements are relevant and assigned as well depends on the selected communication matrix element and the bus configuration part (Simulated ECUs, Inspection, Manipulation) to which you assign the selected element. For more information, refer to:

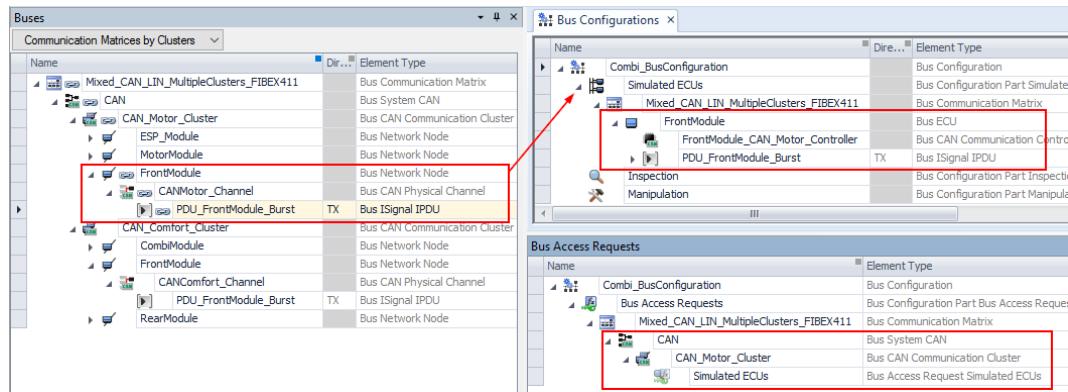
- [Assigning elements to the Simulated ECUs part](#) on page 76
- [Assigning elements to the Inspection and/or Manipulation part](#) on page 78
- [Notes on assigning specific PDU types](#) on page 80

PDUs and ISignals that are assigned to a bus configuration are instantiated. For more information, refer to [Instantiating PDUs and ISignals](#) on page 81.

#### Assigning elements to the Simulated ECUs part

Simulating bus communication is ECU-dependent. Therefore, communication matrix elements are assigned in the context of their sending or receiving ECU when you assign the elements to the Simulated ECUs part. The assignment of the related higher-level elements and subelements differs, depending on whether you select the communication matrix elements in the Communication Matrices by Clusters view or Communication Matrices by ECUs view of the Buses Browser, as shown in the following examples.

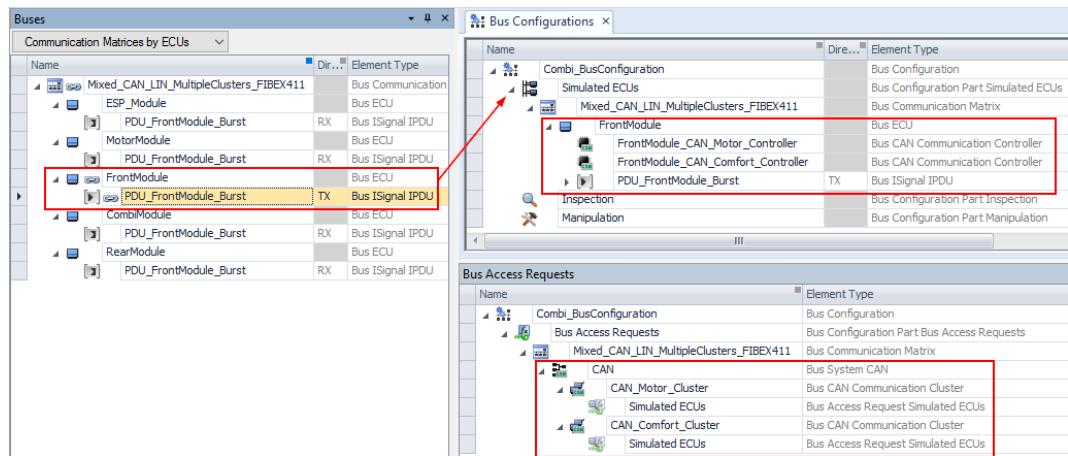
**Assigning communication matrix elements via the Communication Matrices by Clusters view** If you assign a communication matrix element from the Communication Matrices by Clusters view, it is assigned in the context of the related [network node](#) and [communication cluster](#):



In the example above, 'PDU\_FrontModule\_Burst' is selected for the 'FrontModule' network node of 'CAN\_Motor\_Cluster' and assigned to the Simulated ECUs part. As a result, only the elements that are relevant for the bus communication of the 'FrontModule' network node via the 'CAN\_Motor\_Cluster' are assigned as well. Elements that are relevant for the bus communication of the 'FrontModule' network node via the 'CAN\_Comfort\_Cluster' are not assigned.

### Assigning communication matrix elements via the Communication Matrices by ECUs view

If you assign a communication matrix element from the Communication Matrices by ECUs view, it is assigned in the context of the related ECU<sup>(?)</sup>:



In the example above, 'PDU\_FrontModule\_Burst' is selected for the 'FrontModule' ECU and assigned to the Simulated ECUs part. Since 'FrontModule' transmits the PDU via two communication clusters ('CAN\_Motor\_Cluster', 'CAN\_Comfort\_Cluster'), all elements that are relevant for the bus communication of both communication clusters are assigned as well. However, other ECUs that transmit or receive the PDU as well (e.g., 'MotorModule') are not affected and not assigned.

### Assigning elements to the Inspection and/or Manipulation part

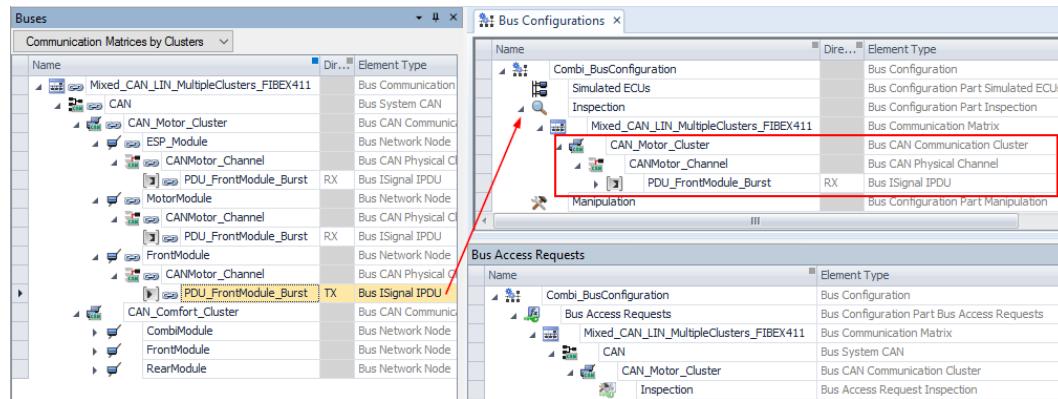
Inspecting and manipulating bus communication is cluster-dependent. Additionally, inspecting bus communication is only possible for received bus communication while manipulating is only possible for transmitted bus communication. When you assign communication matrix elements to the Inspection or Manipulation part, the elements are therefore assigned in the following ways:

- The elements are assigned in the context of their communication clusters, regardless of their sending and/or receiving ECUs.
- If you assign elements to the Inspection part, the direction of the assigned elements is always RX, regardless of whether you assign an RX or TX communication matrix element.
- If you assign elements to the Manipulation part, the direction of the assigned elements is always TX, regardless of whether you assign a TX or RX communication matrix element.

Assigning the related higher-level elements and subelements does not differ for the Inspection and Manipulation part. However, the assignment differs depending on whether you select the communication matrix elements in the Communication Matrices by Clusters view or Communication Matrices by ECUs view of the Buses Browser, as shown in the following examples.

**Assigning communication matrix elements via the Communication Matrices by Clusters view** If you assign a communication matrix element from the Communication Matrices by Clusters view to the Inspection or Manipulation part, it is assigned in the context of the related communication cluster:

- In the following example, 'PDU\_FrontModule\_Burst' TX PDU is selected for the 'FrontModule' network node of 'CAN\_Motor\_Cluster' and assigned to the Inspection part of a bus configuration:



In the bus configuration, the direction of the PDU is converted to RX and all the elements that are relevant for the bus communication of the related communication cluster are assigned as well. The network node for which the PDU is selected does not directly affect the assignment. Therefore, the PDU is marked with the chain symbol for all the network nodes of 'CAN\_Motor\_Cluster'.

- In the following example, 'PDU\_FrontModule\_Burst' RX PDU is selected for the 'MotorModule' network node of 'CAN\_Motor\_Cluster' and assigned to the Manipulation part of a bus configuration:

The screenshot shows two windows side-by-side. On the left is the 'Buses' window titled 'Communication Matrices by Clusters'. It lists various bus configurations and their components. A red arrow points from a selected 'PDU\_FrontModule\_Burst' entry in the 'CAN\_Motor\_Cluster' section to the 'Bus Configurations' window on the right. The 'Bus Configurations' window has three tabs: 'Combi\_BusConfiguration', 'Bus Access Requests', and 'Manipulation'. The 'Manipulation' tab is active, showing a table with columns 'Name', 'Direction', and 'Element Type'. A row for 'Mixed\_CAN\_LIN\_MultipleClusters\_FIBEX411' is selected, and its details are shown in a expanded tree view. The 'PDU\_FrontModule\_Burst' entry is highlighted with a red border in both the Buses and Bus Configurations tables.

In the bus configuration, the direction of the PDU is converted to TX. The assignment of the relevant related elements is identical to the assignment of the Inspection part.

**Assigning communication matrix elements via the Communication Matrices by ECUs view** If you assign a communication matrix element from the Communication Matrices by ECUs view to the Inspection or Manipulation part, it is assigned in the context of all the related communication clusters. In the following examples, 'PDU\_FrontModule\_Burst' TX PDU is selected for the 'FrontModule' ECU and assigned to the Inspection and Manipulation part of a bus configuration:

This screenshot is similar to the one above but shows the 'Communication Matrices by ECUs' view in the Buses window instead. A red arrow points from a selected 'PDU\_FrontModule\_Burst' entry in the 'FrontModule' section to the 'Bus Configurations' window. The 'Bus Configurations' window shows the same 'Manipulation' tab with the 'Mixed\_CAN\_LIN\_MultipleClusters\_FIBEX411' row selected. The expanded tree view in the 'Manipulation' tab shows the 'PDU\_FrontModule\_Burst' entry under the 'CAN\_Motor\_Cluster' cluster, with its direction set to RX. This indicates that the TX direction from the Buses view was converted to RX for the specific bus configuration context.

The screenshot shows three main windows:

- Buses**: A tree view of communication matrices by ECUs. It includes nodes like Mixed\_CAN LIN\_MultipleClusters\_FIBEX411, ESP\_Module, MotorModule, FrontModule, CombiModule, and RearModule. Some PDU entries have TX or RX directions and element types like Bus Communication, Bus ECU, or Bus ISignal IPDU.
- Bus Configurations**: A tree view of bus configurations. It includes nodes like Combi\_BusConfiguration, Simulated ECUs, Inspection, Manipulation, and Mixed\_CAN LIN\_MultipleClusters\_FIBEX411. This node is expanded to show sub-clusters: CAN\_Motor\_Cluster (with CANmotor\_Channel and PDU\_FrontModule\_Burst) and CAN\_Confort\_Cluster (with CANcomfort\_Channel and PDU\_FrontModule\_Burst). These sub-clusters are highlighted with a red box.
- Bus Access Requests**: A tree view of bus access requests, showing the same structure as the Bus Configurations window but for access requests.

For the Inspection part, the direction of the PDU is converted to RX. For the Manipulation part, the direction of the PDU remains TX. Because the ECU transmits the PDU via two communication clusters ('CAN\_Confort\_Cluster', 'CAN\_Motor\_Cluster'), all the elements that are relevant for the bus communication of both communication clusters are assigned to the Inspection and Manipulation part as well. However, the ECU for which the PDU is selected does not directly affect the assignment. Therefore, the PDU is marked with the chain symbol for all the ECUs of 'CAN\_Confort\_Cluster' and 'CAN\_Motor\_Cluster'.

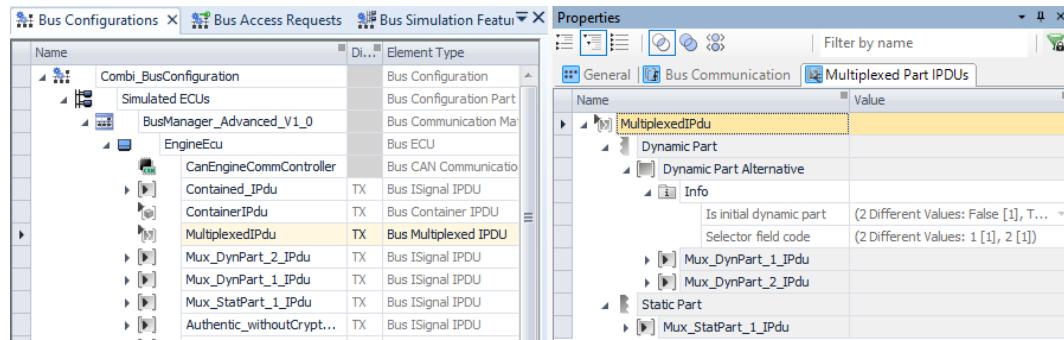
#### Notes on assigning specific PDU types

Container IPDUs<sup>②</sup>, multiplexed IPDUs<sup>③</sup>, and secured IPDUs<sup>④</sup> can have dependent IPDUs, which are displayed as subelements of the IPDUs in the Buses Browser. To these IPDU types and their dependent IPDUs, the same assignment rules apply as to other communication matrix elements. However, PDUs are always displayed on the same hierarchy level in bus configurations:

The screenshot shows three main windows:

- Buses**: A tree view of communication matrices by clusters. It includes nodes like CanBodyCluster, CanPowertrainCluster, and EngineEcu. Under EngineEcu, there are sub-nodes: CanPowertrainPhysicalChannel, ContainerIPdu, MultiplexedIPdu, Mux\_DynPart\_1\_IPdu, Mux\_DynPart\_2\_IPdu, Mux\_StatPart\_1\_IPdu, Secured\_withoutCryptographic\_IPdu, and Authentic\_withoutCryptographic\_IPdu.
- Bus Configurations**: A tree view of bus configurations. It includes nodes like Combi\_BusConfiguration, Simulated ECUs, BusManager\_Advanced\_V1\_0, and EngineEcu. Under EngineEcu, it shows sub-nodes: CanEngineCommController, Contained\_IPdu, ContainerIPdu, MultiplexedIPdu, Mux\_DynPart\_1\_IPdu, Mux\_DynPart\_2\_IPdu, Mux\_StatPart\_1\_IPdu, Authentic\_withoutCryptographic\_IPdu, and Secured\_withoutCryptographic\_IPdu. Red arrows point from the Buses tree to the Bus Configurations tree, indicating the relationship between the two levels of hierarchy.
- Bus Access Requests**: A tree view of bus access requests, showing the same structure as the Bus Configurations window but for access requests.

When you select a container IPDU, multiplexed IPDU, or secured IPDU in the bus configuration, the Properties Browser provides the Contained IPDUs, Multiplexed Part IPDUs, or Authentic IPDUs page, which displays the related dependent IPDUs, as shown in the following example:

**Tip**

In the example above, the Is initial dynamic part and Selector field code properties are grouped for the multiplexed IPDU. To display the properties for each dynamic part IPDU, select the intersection mode in the Properties Browser.

### Instantiating PDUs and ISignals

PDUs and ISignals that are assigned to bus configurations are instantiated. The instances of one PDU/ISignal are not synchronized. Therefore, you can configure different settings for each instance.

Depending on the bus configuration part (Simulated ECUs, Inspection, Manipulation) you assign PDUs and ISignals to, they are instantiated in the following ways:

- PDUs and ISignals that are assigned to the Simulated ECUs part of one or more bus configurations are instantiated for each related transmitting or receiving ECU and for each bus configuration.

In the following example, 'PDU\_FrontModule\_Burst' is assigned to two bus configurations: To 'Bus Configuration (1)', the PDU is assigned in the context of two ECUs. To 'Bus Configuration (2)', the PDU is assigned in the context of three ECUs. Therefore, five instances are generated for the PDU.

The screenshot shows two windows side-by-side. On the left is the 'Buses' window titled 'Communication Matrices by ECUs'. It lists a communication matrix named 'Mixed\_CAN\_LIN\_MultipleClusters\_FIBEX411' containing various bus components like ESP\_Module, MotorModule, FrontModule, CombiModule, and RearModule, each with specific port assignments (RX or TX) and element types (Bus ECU, Bus Signal IPDU). On the right is the 'Bus Configurations' window. It shows two bus configurations: 'Bus Configuration (1)' and 'Bus Configuration (2)'. Each configuration contains a 'Simulated ECUs' section with a 'Mixed\_CAN\_LIN\_MultipleClusters\_FIBEX411' matrix. Within these matrices, specific PDU entries ('PDU\_FrontModule\_Burst') are highlighted with red boxes, indicating they are being assigned to the 'Inspection' or 'Manipulation' parts of the bus configurations.

- PDUs and ISignals that are assigned to the Inspection or Manipulation part of one or more bus configurations are instantiated for each related communication cluster and channel, and for each bus configuration.

In the following example, 'PDU\_FrontModule\_Burst' is assigned to the Inspection part of two bus configurations: To 'Bus Configuration (1)', the PDU is assigned in the context of one communication cluster. To 'Bus Configuration (2)', the PDU is assigned in the context of two communication clusters. Therefore, three instances are generated for the PDU.

This screenshot illustrates a similar setup but using 'Communication Matrices by Clusters'. The 'Buses' window shows a matrix 'Mixed\_CAN\_LIN\_MultipleClusters\_FIBEX411' with clusters like CAN, CAN\_Motor\_Cluster, CANComfort\_Cluster, and FrontModule. The 'Bus Configurations' window shows 'Bus Configuration (1)' and 'Bus Configuration (2)'. In 'Bus Configuration (1)', the PDU is assigned to the 'CAN\_Motor\_Cluster' cluster. In 'Bus Configuration (2)', the PDU is assigned to both the 'CAN\_Motor\_Cluster' and 'CANComfort\_Cluster' clusters. Red boxes highlight the PDU entries in both configurations, demonstrating how multiple clusters can lead to multiple instantiations of the same PDU assignment.

## Conflicting assignments

A communication matrix can contain elements that cannot be used with the Bus Manager, for example, PDUs with exceeding ISignals or ISignals with an invalid coding. If you assign such elements to bus configurations, conflicts occur. In most cases, you must resolve these conflicts before you can generate bus simulation containers that include the affected bus configurations. Refer to [Basics on Bus Manager-Related Conflicts](#) on page 99.

**Related topics****Basics**

Basics on Bus Access Requests.....	91
Introduction to the Properties Browser (ConfigurationDesk Real-Time Implementation Guide)	
Use Scenarios for Configuring Bus Communication with the Bus Manager.....	20

## Removing Communication Matrix Elements from Bus Configurations

**Introduction**

You can remove communication matrix elements from bus configurations by deleting them in bus configuration tables..

**Removing communication matrix elements in bus configuration tables**

To remove a communication matrix element from a bus configuration, you can select it in a bus configuration table (e.g., in the Bus Configurations table or Bus Simulation Features table) and delete it via the context menu or the **Delete** key. Deleting a communication matrix element has the following effects on the related bus configuration:

- Subelements of the deleted element are removed as well.
- Higher-level elements of the deleted element are removed as well if all of the following conditions are fulfilled:
  - The higher-level elements have no other subelements.
  - The higher-level elements have no configurable properties themselves.
  - No bus configuration features are available for the higher-level elements.
- Elements that directly depend on the deleted element are removed as well:
  - If you delete a bus channel (available for the Inspection and Manipulation bus configuration parts), the related cluster is removed as well even though it has configurable properties.
  - If you delete a communication controller (available only for the Simulated ECUs bus configuration part), all the elements that are transmitted or received by the related ECU via the affected cluster are removed as well.

For example, you work with a bus configuration as shown in the following illustration and you delete the 'FrontModule\_CAN\_Comfort\_Controller'.

Name	Element Type	Related Clusters
Bus Configuration (1)	Bus Configuration	
Simulated ECUs	Bus Configuration Part Simulated ECUs	
Mixed_CAN_LIN_MultipleClusters_FIBEX411	Bus Communication Matrix	
FrontModule	Bus ECU	
FrontModule_CAN_Motor_Controller	Bus CAN Communication Controller	CAN_Motor_Cluster; CAN_Comfort_Cluster;...
FrontModule_CAN_Comfort_Controller	Bus CAN Communication Controller	CAN_Comfort_Cluster
PDU_ESP2Front_AC	RX Bus Signal IPDU	CAN_Motor_Cluster
PDU_Motor2Front_AC	RX Bus Signal IPDU	CAN_Motor_Cluster
PDU_Combi2Front_AC	RX Bus ISignal IPDU	CAN_Comfort_Cluster
PDU_Rear2Front_AC	RX Bus Signal IPDU	CAN_Comfort_Cluster
PDU_Front2ESP_AC	TX Bus Signal IPDU	CAN_Motor_Cluster
PDU_Front2Motor_AC	TX Bus Signal IPDU	CAN_Motor_Cluster
PDU_FrontModule_Burst	TX Bus ISignal IPDU	CAN_Motor_Cluster; CAN_Comfort_Cluster
PDU_Front2Combi_AC	TX Bus Signal IPDU	CAN_Comfort_Cluster
PDU_Front2Rear_AC	TX Bus ISignal IPDU	CAN_Comfort_Cluster
Inspection	Bus Configuration Part Inspection	
Manipulation	Bus Configuration Part Manipulation	

In this case, all the PDUs that are transmitted or received only via 'CAN\_Comfort\_Cluster' are removed as well. 'PDU\_FrontModule\_Burst' is transmitted via two clusters. Therefore, the PDU itself is not removed but only the frame that transmits the PDU via 'CAN\_Comfort\_Cluster'.

## Working with User Code

### Introduction

When you configure bus communication with the Bus Manager, you can work with user code. You can use user code to implement user-specific algorithms in [executable applications](#) .

### User code and the DsBusCustomCode interface

When you work with user code, you also need the functionality of the Bus Custom Code interface.

**User code** User code is C code or C++ code that contains user-specific algorithms. You can use user-specific algorithms to provide additional functionality to the Bus Manager, for example, for calculating checksum or counter signal values or generating authentication information in secure onboard communication (SecOC) scenarios. A user code implementation in ConfigurationDesk consists of one or more source files (\*.c, \*.cpp) and optional include files (\*.h, \*.hpp), such as header files.

**Bus Custom Code interface** The Bus Custom Code interface is a C-based interface provided by dSPACE. It is required to implement user code and its functionalities in executable applications. In general, the Bus Custom Code interface lets you do the following:

- Specify a user code ID for each user code implementation. This ID is required to unambiguously reference specific user code implementations in bus configurations.
- Initialize the functionalities provided by user code in the executable application.
- Exchange data between the user code and the Bus Manager, for example, to access properties of secured IPDUs or write calculated checksum values to ISignals of a PDU.

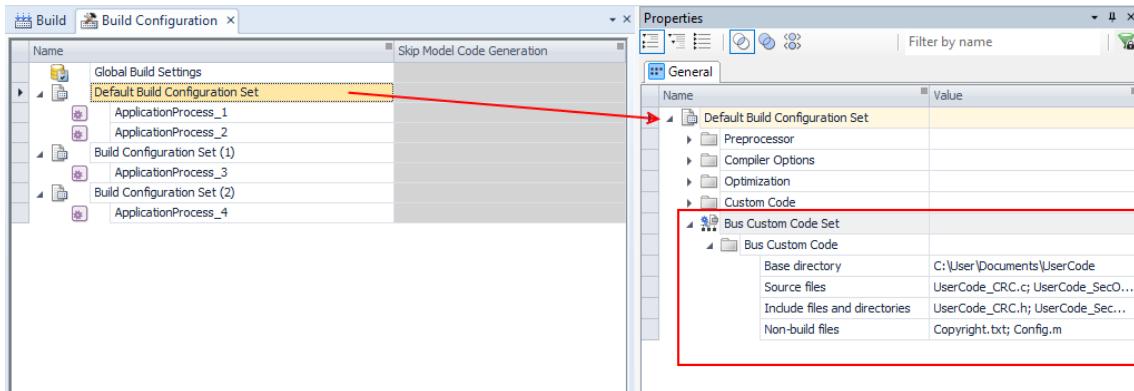
For more information on the Bus Custom Code interface, and on the required content and structure of user code implementations, refer to [Bus Custom Code Interface Handling](#) .

### Implementing user code in ConfigurationDesk applications

To implement user code in ConfigurationDesk applications, you must add the required user code implementations to build configuration sets and reference the user code in bus configurations.

**Adding user code implementations to build configuration sets** You can add user code implementations to a ConfigurationDesk application via properties of build configuration sets. When you select a build configuration set in the

Build Configuration table, you can access the relevant properties in the Properties Browser, as shown in the following example.



### Tip

If the Build Configuration table is closed, you can open it on the View ribbon via Switch Controlbars.

Via the properties of the Bus Custom Code category of each build configuration set, you can add one or more user code implementations. The related user code applies to all application processes that are assigned to this build configuration set. Application processes of other build configuration sets are not affected by the user code.

When you generate bus simulation containers, the user code is included in the bus simulation containers.

Via the Bus Custom Code category, you can add the following files:

- Source files (\*.c; \*.cpp)
- Include files (\*.h; \*.hpp), such as header files
- Non-build files (\*.\*), such as copyright information, read-me files, or configuration files.

### Tip

For example, if you use bus simulation containers to archive bus communication, you can add non-build files to include the information that is required to ensure that the archives comply with applicable law.

For more information on build configuration sets, refer to [Specifying Options for the Build Process \(ConfigurationDesk Real-Time Implementation Guide\)](#).

For more information on the properties of the Bus Custom Code category, refer to [Build Configuration Table \(ConfigurationDesk User Interface Reference\)](#).

**Referencing user code in bus configurations** You can add various user code implementations to a ConfigurationDesk application. This allows you to use different user code implementations for different use cases, e.g., to implement secure onboard communication or provide checksum algorithms to be used with the PDU User Code feature.

To unambiguously reference specific user code implementations in bus configurations, user code IDs are used. Exactly one source file (\*.c, \*.cpp) of a user code implementation must specify a unique user code ID via the `DS_BUS_CUSTOM_CODE_FEATURE_NAME` definition. You must reference the specified ID in the required user code ID property of a bus configuration. For more information, refer to:

- [Implementing Secure Onboard Communication in Executable Applications](#) on page 87
- [Applying User Code to PDUs](#) on page 170

However, to use the related user code, the bus configuration must be assigned to an application process of the build configuration set to which you added the referenced user code implementation.

---

**Restrictions for implementing user code in ConfigurationDesk applications**

Regarding user code implementations and build configuration sets, the following principles apply:

- User code implementations contain functions of the `DsBusCustomCode_SecOC` and/or `DsBusCustomCode_PduUserCode` module of the Bus Custom Code interface.
- All user code implementations that are added to one build configuration set apply to all application processes that are assigned to this build configuration set.

These principles result in the following restriction: Each application process that is assigned to a build configuration set must use all the modules, i.e., the `DsBusCustomCode_SecOC` and/or `DsBusCustomCode_PduUserCode` module, that are contained in any user code implementation of the build configuration set.

**Note**

You must manually ensure that each application process uses all required modules. If an application process does not use a required module and you generate bus simulation containers, the BSC file that is generated for the application process is invalid.

An application process uses the modules, for example, in the following cases:

- `DsBusCustomCode_SecOC` module: SecOC support is enabled for at least one bus configuration of the application process.
- `DsBusCustomCode_PduUserCode` module: The PDU User Code feature is added to at least one bus configuration of the application process.

For more information on the modules, refer to [Basic Principles of the Bus Custom Code Interface \(Bus Custom Code Interface Handling\)](#).

# Implementing Secure Onboard Communication in Executable Applications

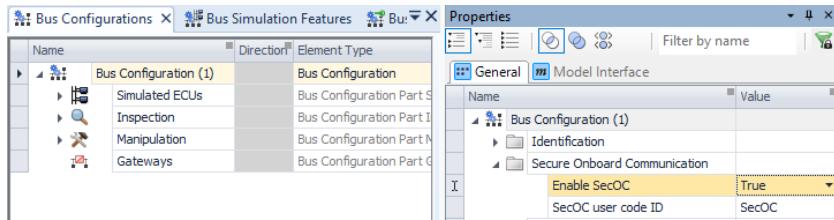
## Introduction

The Bus Manager supports secure onboard communication (SecOC) according to AUTOSAR. Refer to [Secure onboard communication for PDUs](#) on page 41.

To implement secure onboard communication in [executable applications](#), you must enable SecOC support and provide the OEM-specific implementation for generating and/or verifying authentication information via user code. For basic information on user code, refer to [Working with User Code](#) on page 84.

## Enabling SecOC support

Each bus configuration provides an Enable SecOC property that lets you enable the support of secure onboard communication for the bus configuration.



**Effects of enabled SecOC support** If Enable SecOC is set to True, the payload of authentic IPDUs is secured by the related secured IPDUs, i.e., by authentication information that is generated according to an OEM-specific implementation provided by user code. Additionally, you can configure to verify the authentication information of received IPDUs by the OEM-specific implementation. The required user code must be referenced via a user code ID. Refer to [Providing user code for secure onboard communication](#) on page 87.

**Effects of disabled SecOC support** If Enable SecOC is set to False, no user code is used, regardless of whether a user code ID is specified. No authentication information is generated and the payload of assigned authentic IPDUs is not secured. If secured IPDUs are assigned to the bus configuration, they are configured as follows:

- The payload of the related authentic IPDUs is directly included in the secured IPDUs.
- The bits of the secured IPDUs that are reserved for the authentication information are unused. These bits are filled with the related bit pattern for unused bits, i.e., the Bus Manager's default bit pattern 0 or the bit pattern that is specified for each affected secured IPDU.

## Providing user code for secure onboard communication

To use secure onboard communication at run time, you must provide the required OEM-specific implementation for generating and/or verifying authentication information via user code as follows:

1. Specify the user code, i.e., the implementation for generating and/or verifying authentication information, according to your requirements in C code (\*.c, \*.h) or C++ code (\*.cpp, \*.hpp).

2. Extend the user code by functions for initializing secure onboard communication and for exchanging data with the Bus Manager, such as accessing properties of secured IPDUs or writing generated authentication information to secured IPDUs. Use functions of the `DsBusCustomCode` and `DsBusCustomCode_SecOC` modules of the Bus Custom Code interface for this purpose.  
For more information, refer to [Bus Custom Code Interface Handling](#).
3. Add the code files, i.e., the user code implementation, to the ConfigurationDesk application. Refer to [Implementing user code in ConfigurationDesk applications](#) on page 84.
4. Reference the required user code. To do so, type the user code ID as specified in the user code implementation into the edit field of the SecOC user code ID property. In the user code implementation, the user code ID is specified via the `DS_BUS_CUSTOM_FEATURE_NAME` definition of the related source file (\*.c, \*.cpp).
5. If you want to verify the authentication information of received secured IPDUs, add the SecOC bus configuration feature to the related secured IPDUs. Refer to [Verifying the Authentication Information of Received Secured IPDUs](#) on page 175.

**Tip**

- IPDU-specific configurations are required only for verifying received authentication information, not for generating authentication information.
- Additional bus configuration features are available for actions such as manipulating authentication information. For an overview of the available bus configuration features, refer to [Basics on Bus Configuration Features](#) on page 116.

## Implementing Global Time Synchronization in Executable Applications

### Introduction

The Bus Manager supports global time synchronization (GTS) according to AUTOSAR. Refer to [Global time synchronization](#) on page 43.

If global time synchronization is specified in the communication matrix, the Bus Manager lets you implement global time synchronization in [executable applications](#).

### Representation of global time domains

When you add a communication matrix specifying global time synchronization to the Buses Browser, you can access global time domains via global time domain elements that are available for general-purpose PDUs. For each global time domain, there can be several global time domain elements, one element for each ECU that is a member of this global time domain.

Because an ECU can be a member of multiple global time domains, there can also be multiple global time domain elements for an ECU: For each global time domain of which an ECU is a member, a separate global time domain element is available. The direction of the global time domain element indicates whether the ECU acts as time master or time slave:

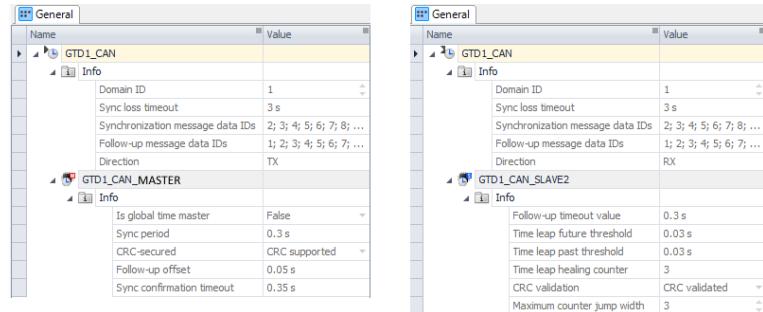
- TX global time domain element: Time master

GlobalTimeSyncDemo		Bus Communication Matrix	
		Bus ECU	
ECU_GTS_FR_CAN_GW			
GTS_CanFrame_TD2_PDU	TX	Bus General-Purpose PDU	
GTD2_CAN	TX	Bus Global Time Domain	

- RX global time domain element: Time slave

GlobalTimeSyncDemo		Bus Communication Matrix	
		Bus ECU	
ECU_GTS_FR_CAN_GW			
ECU_GTS_CAN_SLAVE3_TD2			
GTS_CanFrame_TD2_PDU	RX	Bus General-Purpose PDU	
GTD2_CAN	RX	Bus Global Time Domain	

When you select a global time domain element, the Properties Browser provides access to properties of the global time domain and the time master or time slave, as shown in the following example.



## Simulating global time synchronization

You can configure global time domains for simulation purposes. For each global time domain element you assign to the Simulated ECUs part of a bus configuration, an instance of the dSPACE ECU time base manager is generated. The dSPACE ECU time base manager implements functionalities of the AUTOSAR synchronized time base manager, i.e., it provides the synchronized time base.

At run time, the time bases of global time domains can be synchronized only after the related global time master has set the time at least once. Depending on whether a real ECU provides the global time master, the time must be set as follows:

Global Time Master Provided by Real ECU	Global Time Master Not Provided by Real ECU
<p>The global time master provided by the real ECU must set the time and transmit the time information on the bus at least once.</p> <p>After the SCALEXIO or MicroAutoBox III system has received the time, the related time bases can be synchronized, i.e.:</p> <ul style="list-style-type: none"> <li>▪ Simulated time slaves can synchronize their local time base instances.</li> </ul>	<p>You must simulate the global time master by using the RTI Synchronized Time Base Manager Blockset or the Bus Manager.</p> <p>However, even if you use the Bus Manager to simulate the global time master, you must work with a Simulink behavior model and use blocks of the RTI Synchronized Time Base Manager Blockset: If a TX global time domain element that is a global time master is assigned to a bus configuration, the Bus Manager can simulate the global time master and</p>

Global Time Master Provided by Real ECU	Global Time Master Not Provided by Real ECU
<ul style="list-style-type: none"> <li>Simulated time masters can distribute the synchronized time among their global time domain.</li> </ul>	<p>synchronize time bases only after the blockset's STBM_SET_GLOBAL_TIME block has set the time at least once.</p> <p>For more information on the RTI Synchronized Time Base Manager Blockset, refer to <a href="#">RTI Synchronized Time Base Manager Blockset Reference</a>.</p>

Additionally, bus configuration features are available for simulating global time synchronization. Via the bus configuration features, you can access the time base data of time masters and time slaves, for example. Refer to [Bus Configuration Features Available for Global Time Domains](#) on page 214.

---

<b>Run-time behavior regarding synchronization periods</b>	<p>The synchronization period of a time master determines the time that is available for transmitting a synchronization message and its related follow-up message on the bus. If the synchronization period is shorter than required for transmitting both messages, the time synchronization is executed as fast as possible, i.e., both messages are transmitted and not interrupted.</p> <p>This applies regardless of whether a synchronization period is specified in the communication matrix or via the <a href="#">GTS Transmission Control</a> bus configuration feature.</p>
--	--

---

<b>Handling CRC-secured time synchronization</b>	<p>Time synchronization can be CRC-secured. For this purpose, time masters and time slaves provide the following properties:</p> <table border="1"> <thead> <tr> <th>Property</th><th>Available for</th><th>Purpose</th></tr> </thead> <tbody> <tr> <td>CRC-secured</td><td>Time master</td><td>Specifies whether time synchronization is CRC-secured.</td></tr> <tr> <td>CRC validation</td><td>Time slave</td><td>Specifies whether received CRC-secured data is validated.</td></tr> </tbody> </table>	Property	Available for	Purpose	CRC-secured	Time master	Specifies whether time synchronization is CRC-secured.	CRC validation	Time slave	Specifies whether received CRC-secured data is validated.
Property	Available for	Purpose								
CRC-secured	Time master	Specifies whether time synchronization is CRC-secured.								
CRC validation	Time slave	Specifies whether received CRC-secured data is validated.								

The settings of the properties are derived from the communication matrix. If the communication matrix does not specify settings for the properties or specified settings are invalid, the Bus Manager uses the following default settings:

Property	Setting	Purpose
CRC-secured	CRC not supported	CRC support is disabled for time synchronization.
CRC validation	CRC ignored	CRC-secured data is ignored, i.e., regardless of whether CRC-secured data is received it is not validated.

---

<b>Validating received time synchronization messages</b>	<p>Time synchronization messages that are received by time slaves are validated, i.e., the synchronization (SYNC) message and its related follow-up (FUP) message are validated. The time base of the time slave is synchronized only if both messages pass the validity checks.</p>
--	--

**Tip**

The GTS Validation feature lets you access the validation results and exclude certain validity checks from the validation of time synchronization messages.

### **Restrictions for implementing global time synchronization in executable applications**

For implementing global time synchronization in executable applications, the following restrictions apply:

- Configuring global time domains for inspection or manipulation purposes is not supported.
- If you configure frame gateways for exchanging synchronized time bases, the time base instances of the affected time slaves are not completely synchronous to the time base instance of the related time master. There is a delay resulting from the time that is required for exchanging frames via the gateway.
- If you configure a TX global time domain for simulation purposes, you cannot use any bus configuration feature that is available for the related TX PDU and TX frame.
- If you want to implement global time synchronization in [offline simulation applications](#), you must simulate global time masters either by using the Bus Manager or the RTI Synchronized Time Base Manager Blockset. In both cases, the time must be set via the STBM\_SET\_GLOBAL\_TIME block in a Simulink model. Therefore, [bus simulation containers](#) without mapped behavior models are not supported for implementing global time synchronization in offline simulation applications.

Additionally, some limitations apply for implementing global time synchronization in executable applications. Refer to [Limitations for Communication Matrices and Communication Standards](#) on page 319.

## Basics on Bus Access Requests

### **Introduction**

Bus access requests are generated for communication clusters and CAN frame gateways of bus configurations.

### **Bus access request and bus access**

Bus access requests contain the requirements regarding the bus access.

**Bus access request** Each communication cluster that is assigned to the Simulated ECUs, Inspection, or Manipulation part of a bus configuration requires access to a bus. This request is represented by a bus access request. The bus access request contains the requirements for the bus channels that are to be used for the cluster's bus communication at run time.

Additionally, each frame gateway that is added to the Gateways part of a bus configuration generates two bus access requests. These bus access requests are required to specify the bus channels for exchanging bus communication.

**Bus access** Communication clusters and CAN frame gateways of a bus configuration can access bus channels only via a bus access.

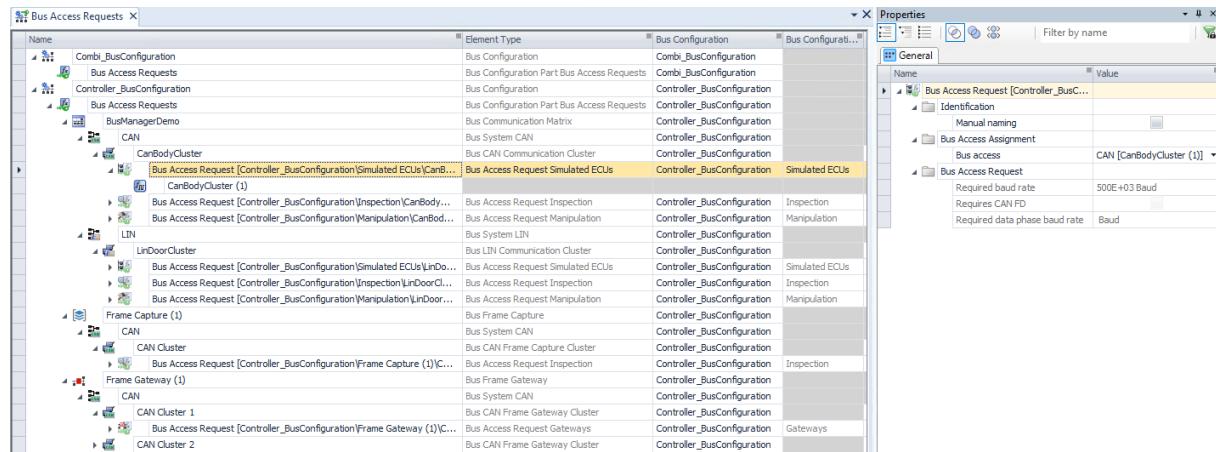
A bus access defines a run-time communication cluster. To be a member of a run-time communication cluster, a bus access request must be assigned to a bus access.

With the Bus Manager (stand-alone), you cannot assign bus access requests to bus accesses. However, when you generate bus simulation containers, the bus access requests are included in the generated BSC files. Refer to [Availability of bus access requests in bus simulation containers](#) on page 255.

## Accessing bus access requests

You can access all bus access requests of the active ConfigurationDesk application via the Bus Access Requests table.

When you select a bus access request, the Properties Browser displays the requirements it places on the bus access.



## Names of bus access requests

By default, each bus access request is generated with a unique name according to the following scheme:

- Names of bus access requests generated for elements assigned to the Simulated ECUs, Inspection, or Manipulation part of a bus configuration:

```
Bus Access Request [<bus configuration name>\<bus configuration part name>\<communication cluster name>\name of communication matrix in bus configuration]
```

Controller_BusConfiguration	Bus Configuration
Bus Access Requests	Bus Configuration Part Bus Access Requests
Controller_BusConfiguration	Bus Communication Matrix
BusManagerDemo	Bus System CAN
CAN	Bus CAN Communication Cluster
CanBodyCluster	Bus Access Request Simulated ECUs
Bus Access Request [Controller_BusConfiguration]Simulated ECUs[CanBodyCluster]BusManagerDemo	

- Names of bus access requests generated for frame gateways of the Gateways part of a bus configuration:

Bus Access Request [<bus configuration name>\<frame gateway name>\<communication cluster name>]	
Controller_BusConfiguration	Bus Configuration
Bus Access Requests	Bus Configuration Part Bus Access Requests
Frame Gateway (1)	Bus Frame Gateway
CAN	Bus System CAN
CAN Cluster 1	Bus CAN Frame Gateway Cluster
Bus Access Request [Controller_BusConfiguration\Frame Gateway (1)\CAN Cluster 1]	Bus Access Request Gateways

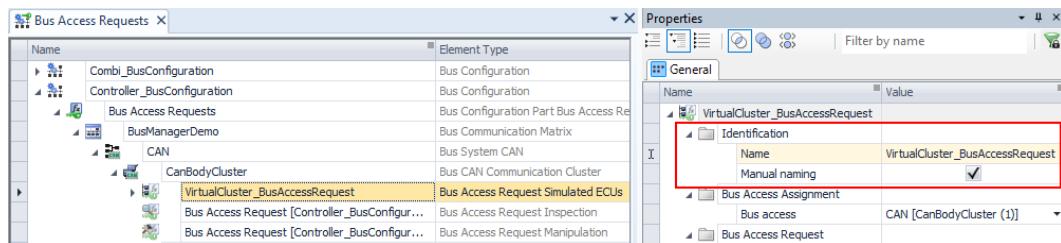
If you change a name that is used for the naming scheme, e.g., the name of a bus configuration, the names of the affected bus access requests are automatically synchronized. Therefore, the generated names unambiguously identify the element for which a bus access request is generated.

Instead of using the generated names, you can specify user-defined names for bus access requests.

### Specifying user-defined names for bus access requests

If required, you can specify user-defined names for bus access requests.

To specify a user-defined name for a bus access request, you must select the **Manual naming** checkbox of the bus access request. Then, you can specify a user-defined name via the **Name** property.



The screenshot shows the ConfigurationDesk interface with the 'Bus Access Requests' tab selected. On the left, a tree view displays the hierarchy of bus configurations, frame gateways, and communication clusters. A specific bus access request named 'VirtualCluster\_BusAccessRequest' is selected. On the right, the 'Properties' panel is open, showing the 'General' tab. In the 'Identification' section, the 'Name' field is set to 'VirtualCluster\_BusAccessRequest' and the 'Manual naming' checkbox is checked. Other sections like 'Bus Access Assignment' and 'Bus Access Request' are also visible.

If you specify user-defined names for bus access requests, note the following:

- The names of bus access requests must be unique in a ConfigurationDesk application.
- When you select the **Manual naming** checkbox of a bus access request, the automatic synchronization of the name is disabled, regardless of whether you change the generated name.

- When you clear the Manual naming checkbox of a bus access request, the name is synchronized immediately. Any changes you made to the generated name are lost.

## Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager

### Introduction

When you configure bus communication with the Bus Manager, the [tasks](#), [events](#), and [Runnable functions](#) that are needed to trigger the bus communication at run time are created and configured automatically. The preconfigured settings are sufficient in most use scenarios so that you do not have to make any manual changes. If required, you can change some of the preconfigured settings.

### Overview

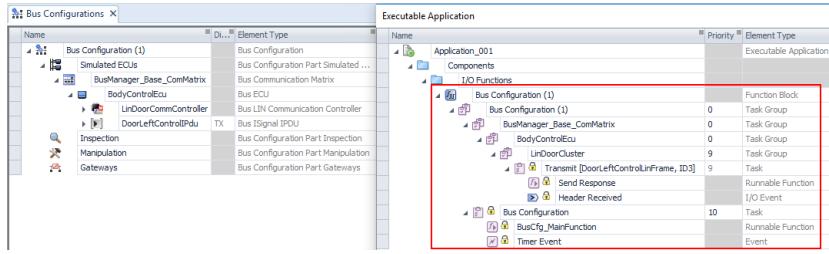
The following table provides an overview of the tasks, events, and runnable functions that can be created automatically.

Name	Type	Created for/Assigned to
Bus Configuration	Periodic task	Created for each bus configuration that is added to the ConfigurationDesk application.
BusCfg_MainFunction	Runnable function	Automatically assigned to the Bus Configuration task of each bus configuration.
Timer Event	Event	Automatically assigned to the Bus Configuration task of each bus configuration.
Transmit [<frame name>, <frame triggering ID>]	Asynchronous task	Created for each LIN TX frame that is assigned to the Simulated ECUs part of a bus configuration.
Send Response	Runnable function	Automatically assigned to each Transmit [<frame name>, <frame triggering ID>] task.
Header Received	I/O event	Automatically assigned to each Transmit [<frame name>, <frame triggering ID>] task.

### Tip

Bus Configuration function blocks do not provide event ports for Header Received I/O events. If you create model port blocks for bus configurations, no Runnable Function blocks are created for these I/O events.

The following illustration is an example of the different tasks, events, and runnable functions available for a bus configuration.



The tasks that are available for a bus configuration are grouped by task groups as follows:

- All tasks of a bus configuration are grouped by a task group that is named after the bus configuration, e.g., 'Bus Configuration (1)'.
- All Transmit [<frame name>, <frame triggering ID>] tasks of a bus configuration are additionally hierarchically grouped by three task groups. The names of these task groups are derived from higher-level elements of the related LIN TX frame as follows:

Level of Task Group	Task Group Name Derived from
First	Communication matrix nodes in the bus configuration, e.g., 'BusManager_Base_ComMatrix'
Second	Transmitting ECU, e.g., 'BodyControlEcu'
Third	LIN cluster, e.g., 'LinDoorCluster'

## Overview of the preconfigured settings

The tasks, events, and runnable functions that are generated when you configure bus communication with the Bus Manager provide the following preconfigured settings:

- The assignment of events and runnable functions to tasks depends on the configuration of each bus configurations. You cannot change this assignment, i.e., you can neither delete nor add events or runnable functions from or to the tasks.
- The priority of each task is specified. If required, you can change the preconfigured priority.
- For the Timer Event of each Bus Configuration task, a period of 1 ms is specified. If required, you can change the preconfigured period.

You can also configure further settings for tasks via their configurable properties. For more information, refer to [Configuring Tasks in ConfigurationDesk \(ConfigurationDesk Real-Time Implementation Guide\)](#).

However, some limitations apply for configuring task, events, and runnable functions. Refer to [Limitations for Bus Configuration Handling](#) on page 328.

**Affects of the Bus Configuration task for transmitting PDUs**

The Bus Configuration task is the main task for transmitting PDUs that are assigned to the Simulated ECUs part of a bus configuration. Each time the Bus Configuration task is executed, a PDU can be transmitted once. The timing of the actual transmission depends on the following process steps:

1. The transmission of a PDU is triggered via behavior model or experiment software, e.g., cyclically via a cyclic timing or asynchronously via the PDU Trigger bus configuration feature.
  2. The Bus Configuration task is executed. Now, the trigger is evaluated and the required data is written to the PDU, such as ISignal values, counter signal values, end-to-end protection information, etc.
- There might be a delay between the trigger and its evaluation by the Bus Configuration task, e.g., due to the Bus Configuration task's priority or the period that is specified for the Timer Event of the Bus Configuration task.
3. The PDU can be transmitted according to the specifications in the related communication matrix.

Depending on these specifications, the transmission might be delayed or the PDU is not transmitted at all, as shown in the following examples:

- The PDU is a contained PDU and the transmission of its related container PDU is not triggered yet.
- The PDU has a LIN frame triggering and the transmission of the related LIN frame is not scheduled by the LIN master.

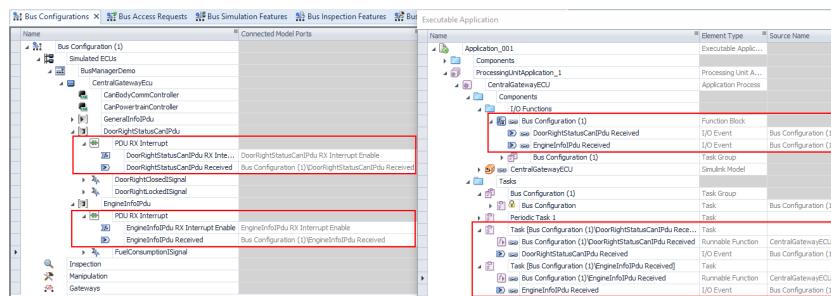
**Tip**

You can additionally prohibit the transmission of the PDU by using bus configuration features, such as the Bus Configuration Enable or Communication Controller Enable feature.

For transmitting PDUs that are assigned to the Simulated ECUs part of a bus configuration, some limitations apply. Refer to [Limitations for Bus Configuration Handling](#) on page 328.

**Events provided by the PDU RX Interrupt feature**

When you add the PDU RX Interrupt feature to a CAN RX PDU, an I/O event is generated each time the PDU is received on the bus. For this I/O event, the bus configuration provides an event port. To use the I/O event at run time, you must map the event port to a runnable function port of a behavior model. When you do this, an asynchronous task is created and the I/O event and runnable function are assigned to this task. This task is not included in the task group of the bus configuration.



For more information on the PDU RX Interrupt feature, refer to [Triggering the Execution of Functions in a Behavior Model via RX Interrupts](#) on page 183.

## **Further information**

For more information on tasks, events, runnable functions, and their effects on executable applications, refer to [Modeling Executable Applications and Tasks](#) ([ConfigurationDesk Real-Time Implementation Guide](#) ).

## Accessing Bus Manager Features via ConfigurationDesk's Automation Interface

## Introduction

You can automate Bus Manager features via ConfigurationDesk's automation interface.

## Accessing Bus Manager features

The automation interface lets you perform tasks such as accessing the following Bus Manager features:

- Adding bus configurations and communication matrices to a ConfigurationDesk application.
  - Assigning and removing communication matrix elements to and from bus configurations.
  - Configuring bus configurations.
  - Specifying user-defined settings for communication matrix elements and exporting overviews of the modified communication matrix elements.
  - Replacing the currently assigned communication matrix of a bus configuration by another communication matrix.
  - Generating bus simulation containers that contain bus configurations of the ConfigurationDesk application.

## Accessing elements via XPath expressions

In addition to using standard queries to access elements, you can access elements of the Bus Manager by using XPath expressions. For example, you can use XPath expressions to access an ISignal of a bus configuration directly without

having to go through the complete hierarchy of the bus configuration. You can access elements by using regular expressions.

To access Bus Manager elements via XPath, the XPath expressions must comply with XPath 1.0. Other XPath versions are not supported.

---

**Further information**

For details on automating Bus Manager features and accessing Bus Manager elements via XPath expressions, refer to [Automating Bus Manager Features](#) ([ConfigurationDesk Automating Tool Handling](#) ). For a detailed description of ConfigurationDesk's automation interface, refer to [ConfigurationDesk Automating Tool Handling](#) .

# Handling Bus Manager-Related Conflicts

---

## Where to go from here

## Information in this section

Basics on Bus Manager-Related Conflicts.....	99
Resolving Bus Manager-Related Conflicts.....	103

## Basics on Bus Manager-Related Conflicts

---

### Overview

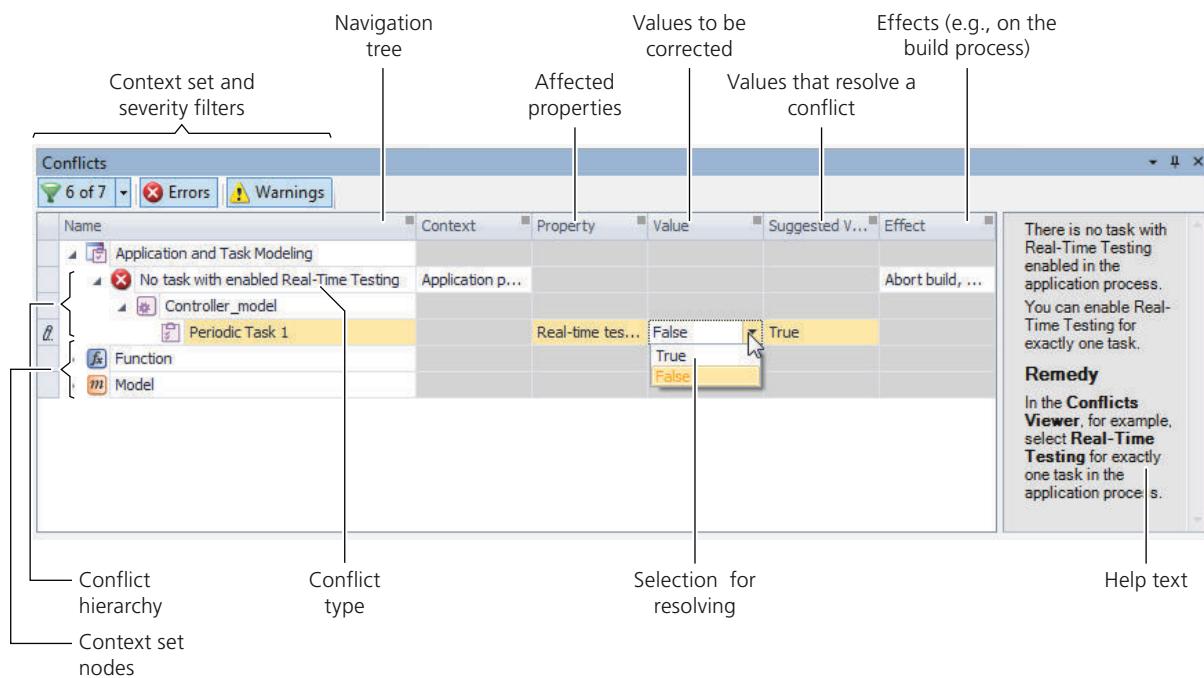
There are two types of Bus Manager-related conflicts, which depend on the conflict source: Communication matrix conflicts and bus configuration conflicts. The conflict types differ in their effects on the [ConfigurationDesk application](#) and/or the [executable application](#).

Handling these Bus Manager-related conflicts differs slightly from handling other ConfigurationDesk conflicts.

---

### Accessing conflicts

The Conflicts Viewer lets you access all conflicts of a ConfigurationDesk application and resolve many of them. The following illustration provides an overview of the Conflicts Viewer.

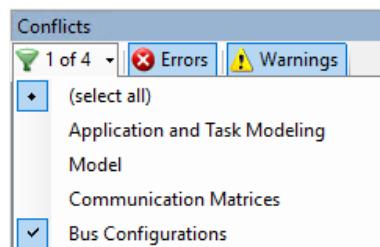


### Enabling the evaluation of conflicts

The Conflicts Viewer provides several context sets to which conflicts are assigned. To enable the evaluation of specific conflicts, you must select the related context set. The Bus Manager-related conflicts are assigned to the following context sets:

- Communication matrix conflicts are assigned to the Communication Matrices context set.
- Bus configuration conflicts are assigned to the Bus Configurations context set.

In the following example, only the evaluation of bus configuration conflicts is enabled.



**Tip**

- Conflicts of other context sets can also be related to the configuration of bus communication. For example, conflicting settings for configured tasks can cause conflicts in the Application and Task Modeling context set.
- The selection of context sets is stored locally on the host PC and applies to all ConfigurationDesk applications, even when you close the Bus Manager (stand-alone).

**Basics on communication matrix conflicts**

Communication matrix conflicts occur in the following cases:

- You import a [communication matrix](#) containing the following:
  - Elements that are not supported by the Bus Manager, such as unsupported frame types.
  - Elements with conflicting settings, such as two or more ECUs with identical names.
- You modify a communication matrix in ConfigurationDesk and these modifications result in conflicting settings. For example, you extend the length of a PDU and due to this change, the PDU exceeds the boundaries of the related frame.

Most of the communication matrix conflicts have no effect on generating bus simulation containers. In most cases, you therefore do not have to resolve these conflicts before you generate bus simulation containers. Only the following two conflicts might result in unintended behavior in the Bus Manager and/or at run time:

- Communication matrix: Failed consistency checks during import of communication matrix
- Communication matrix: Failed schema check during import of communication matrix

**Note**

To prevent unintended behavior, e.g., faulty task generation by the Bus Manager, it is recommended to resolve these two conflicts by correcting the original communication matrix. Refer to [Correct conflicting elements in the original communication matrix](#) on page 105.

**Basics on bus configuration conflicts**

Bus configuration conflicts occur in the following cases:

- You assign conflicting communication matrix elements to a [bus configuration](#).
- You configure conflicting settings for a bus configuration.
- You do not configure all the necessary settings of a bus configuration.

Bus configuration conflicts affect the generation of bus simulation containers. In most cases, you must therefore resolve them before you can generate bus simulation containers.

**Tip**

In the Conflicts Viewer, many elements of bus configuration conflicts provide the **Select Related Elements in Bus Configurations Table** command in the context menu. You can use this command for the following:

- Select all element instances in the affected bus configuration that are related to the conflicting element, e.g., all the PDUs that are included in a conflicting frame and assigned to the bus configuration.
- Access the properties of the selected element instances in the Properties Browser.

**Best practice for handling Bus Manager-related conflicts**

Depending on the complexity of the bus communication that is specified in communication matrices, evaluating Bus Manager-related conflicts can significantly influence the performance of the ConfigurationDesk application. It is therefore recommended to enable the conflict evaluation only when and as long as required.

**Note**

For optimum performance when opening ConfigurationDesk projects or activating ConfigurationDesk applications, it is especially recommended to disable the evaluation of Bus Manager-related conflicts before you close the active ConfigurationDesk application.

For optimum performance, the following is recommended:

- Check a communication matrix once for conflicts after you add it to the ConfigurationDesk application. To do this, select the **Communication Matrices** context set and resolve conflicts, if required. Then, clear the context set.
- Repeatedly check the configured bus communication for conflicts. To do this, configure a reasonable set of bus communication, e.g., the PDUs and ISignals of one ECU you want to simulate. Then, select the **Bus Configurations** context set and resolve conflicts, if required. Finally, clear the context set and configure the next reasonable set of bus communication.

# Resolving Bus Manager-Related Conflicts

## Overview

Depending on the conflict cause, you have the following options to resolve Bus Manager-related conflicts:

- [Complete the implementation of the bus communication in the signal chain on page 103.](#)
- [Remove conflicting communication matrix elements from bus configurations on page 103.](#)
- [Remove or configure conflicting bus configuration features on page 104.](#)
- [Correct conflicting communication matrix elements in ConfigurationDesk on page 104.](#)
- [Correct conflicting elements in the original communication matrix on page 105.](#)

## Complete the implementation of the bus communication in the signal chain

Some bus configuration conflicts occur each time you start implementing bus communication in the signal chain, such as the 'Bus configuration: No valid application process assigned' conflict. In most cases, you resolve these conflicts automatically when you complete the implementation of the bus communication in the signal chain (e.g., as described in [Implementing Bus Communication in the Signal Chain Using the Bus Manager](#) on page 233).

## Remove conflicting communication matrix elements from bus configurations

Bus configuration conflicts that result from conflicting communication matrix elements provide **Is assigned** properties: One for each conflicting element and one for the bus configuration, as shown in the following example.

Name	Context	Property	Value	Suggested Values
Bus Configurations				
Too many PDU-to-frame mappings connected (bus configuration)	CAN frame			
Bus Configuration (1)		Is assigned	True	False
Fuel_System				
FuelRateEngineFrame				
FuelRateEngineFrame				
RawSensorsFrame				
RawSensorsFrame				

You can resolve such conflicts for each separate conflicting element or for the entire bus configuration by setting the related **Is assigned** property to **False**.

When you do this, the affected conflicting elements and their related communication matrix elements are removed from the bus configuration, e.g., a conflicting frame and the related PDUs.

### Note

If you set the **Is assigned** property of the bus configuration to **False**, more elements might be removed than necessary. For example, two ECUs with identical names are assigned to one bus configuration. To resolve the conflict, you have to remove or rename only one of these ECUs. However, if you set the **Is assigned** property of the bus configuration to **False**, both ECUs are removed.

**Tip**

The Select Related Elements in Bus Configurations Table command can help you identify the elements that are also removed.

### **Remove or configure conflicting bus configuration features**

You can resolve bus configuration conflicts that result from conflicting bus configuration features by removing or configuring the conflicting bus configuration features.

**Remove conflicting bus configuration features** Bus configuration conflicts that result from conflicting bus configuration features provide Enabled properties: One for each conflicting feature and one for the bus configuration. You can resolve these conflicts for each separate conflicting feature or for the entire bus configuration by setting the related Enabled property to False.

If you set the Enabled property of the bus configuration to False and the conflict results from an unsupported combination of bus configuration features, all the conflicting features except the main feature are removed. You can identify the main feature by its position in the hierarchy of the Conflicts Viewer, as shown in the following example:

Name	Context	Property	Value	Suggested Values
Feature				
Unsupported combination of Frame Access feature and other PDU or signal features				
Controller_BusConfiguration		Enabled	True	False
FuelSys_System				
FuelRateEngineIPdu				
Frame Access		Enabled	True	False
FuelRateEngineIPdu				
PDU Cyclic Timing Control		Enabled	True	False
PDU Enable		Enabled	True	False
PDU Raw Data		Enabled	True	False
PDU Trigger		Enabled	True	False
FuelRateEngineISignal		Enabled	True	False
Counter Signal		Enabled	True	False
ISignal Value		Enabled	True	False

In the example above, the Frame Access feature is the main feature. If you set the Enabled property of Controller\_BusConfiguration to False, all displayed bus configuration features except the Frame Access feature are removed.

**Configure conflicting bus configuration features** You can resolve some bus configuration conflicts that result from conflicting bus configuration features by specifying valid settings for the feature and/or by specifying user-defined settings for the related communication matrix element as described below.

### **Correct conflicting communication matrix elements in ConfigurationDesk**

You can resolve some Bus Manager-related conflicts by specifying user-defined settings for the conflicting communication matrix elements. By doing so, you change the communication matrix within the active ConfigurationDesk application. The changes apply to all the bus configurations of the active application. The original communication matrix remains unchanged. For more information, refer to [Modifying Communication Matrices](#) on page 267.

---

**Correct conflicting elements  
in the original communication  
matrix**

You can resolve most of the Bus Manager-related conflicts by correcting the original communication matrix, for example, via the tool you used to generate it. You must delete the old, conflict-causing communication matrix from the ConfigurationDesk application and import the corrected communication matrix instead. If you assigned elements of the communication matrix to a bus configuration, you must also replace these elements with the corrected elements. For more information on replacing elements of a bus configuration, refer to [Replacing Assigned Communication Matrices](#) on page 291.

---

**Related topics****Basics**

[Resolving Conflicts \(ConfigurationDesk Real-Time Implementation Guide\)](#)  
[Resolving the Bus Configuration Conflict: No Valid Application Process Assigned.....](#) 341



# Specifying CAN Frame Captures and Gateways

---

## Where to go from here

## Information in this section

Capturing CAN Frames.....	107
Specifying CAN Gateways.....	109
Specifying Filters for Frame Captures and Frame Gateways.....	111

## Capturing CAN Frames

---

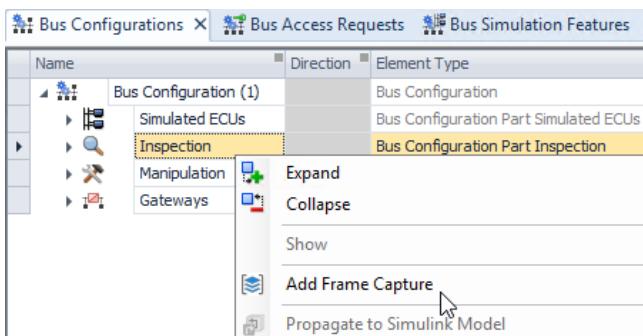
### Introduction

You can capture CAN frames that are received on a bus. Capturing CAN frames lets you access specific parameters and the raw data of received frames, and provide the captured data to a mapped behavior model, for example. To capture CAN frames, you do not need a communication matrix. Instead, you can specify the number of frames that can be captured and specify filters to capture only specific frames.

---

### Adding frame captures to a bus configuration

You can add one or more frame captures to a bus configuration. You can add a frame capture via the Add Frame Capture context menu command of the Inspection bus configuration part. The Inspection part is available in the Bus Configurations table.



Each frame capture is added with a unique default name. You can rename the frame capture, for example, in the Bus Configurations table. However, the name must be unique for all frame captures and frame gateways of the bus configuration.

#### Configuring frame captures

Each frame capture node in the bus configuration provides the following properties that let you configure the frame capture:

Property	Value Range	Description
Maximum number of captured frames	1 ... 256	Lets you specify the maximum number of frames that can be captured in one sampling step.
Maximum frame length	0 ... 8, 12, 16, 20, 24, 32, 48, and 64 bytes	Lets you specify the maximum payload length of frames that can be captured. Only the raw data of the specified number of bytes can be captured at run time.

#### Note

For optimum run-time performance, it is recommended to specify only the number of frames and bytes you really want to capture in one sampling step.

The specified settings influence the data that can be accessed via the Frame Capture Data feature.

#### Elements of frame captures

For each frame capture, the following elements are available to let you specify the frames to be captured:

Element	Description
Bus access request	A <a href="#">bus access request</a> is available that lets you specify the <a href="#">communication cluster</a> for capturing frames. When you generate bus simulation containers, the bus access request is available in the generated BSC file. Refer to <a href="#">Basics on Bus Access Requests</a> on page 91.
Frame Capture Data feature	The Frame Capture Data feature lets you access the data of captured frames. Refer to <a href="#">Accessing the Data of Captured Frames</a> on page 222.
Frame capture filter	The CAN Cluster Filter element lets you specify one or more filter rules for capturing CAN frames. Refer to <a href="#">Specifying Filters for Frame Captures and Frame Gateways</a> on page 111.
Filter Control feature	The Filter Control feature is available for the frame capture filter. It lets you enable and disable the filter, and specify the filter mode. Refer to <a href="#">Controlling Filters of Frame Captures and Frame Gateways</a> on page 227.

**Restrictions for specifying frame captures**

You cannot specify frame captures for LIN clusters.

## Specifying CAN Gateways

**Introduction**

You can specify gateways to exchange CAN frames between two [communication clusters](#).

**Gateways for classic CAN and CAN FD frames**

Gateways are supported for classic CAN and CAN FD frames as follows:

- Classic CAN frames can be routed to classic CAN and CAN FD clusters.
- CAN FD frames can only be routed to CAN FD clusters.

If you specify a gateway and a CAN FD frame is to be routed to a classic CAN cluster, the CAN FD frame is discarded and lost, i.e., it is not routed to the classic CAN cluster.

**Note**

CAN FD frames that are to be routed to a classic CAN cluster are discarded and lost without notice.

**Specifying gateways via frame gateways**

**Adding frame gateways** To specify a gateway between two communication clusters, you must add a frame gateway to the Gateways part of a bus configuration. You can add one or more frame gateways via the Add Frame Gateway context menu command of the Gateways part. The Gateways part is available in the Bus Configurations table.

Name	Direction	Element Type
Bus Configuration (1)		Bus Configuration
Simulated ECUs		Bus Configuration Part Simulated ECUs
Inspection		Bus Configuration Part Inspection
Manipulation		Bus Configuration Part Manipulation
Gateways		Bus Configuration Part Gateways

Each frame gateway is added with a unique default name. You can rename the frame gateways, for example, in the Bus Configurations table. However, the name must be unique for all frame gateways and frame captures of the bus configuration.

**Elements of frame gateways** For each frame gateway, the following elements are available to let you specify the gateway:

Element	Description
Bus access request	Two <a href="#">bus access requests</a> are available. They are required for specifying the communication clusters between which the CAN communication is exchanged. When you generate bus simulation containers, the bus access requests are available in the generated BSC file. Refer to <a href="#">Basics on Bus Access Requests</a> on page 91.
Frame Gateway Direction feature	The Frame Gateway Direction feature lets you specify the gateway direction or disable a gateway. Refer to <a href="#">Specifying the Direction of CAN Frame Gateways</a> on page 226.
Gateway filter	The CAN Cluster 1 Filter and CAN Cluster 2 Filter elements let you specify filter rules for each communication cluster. You can specify multiple filter rules for each filter element to filter frames for gatewaying. Refer to <a href="#">Specifying Filters for Frame Captures and Frame Gateways</a> on page 111.
Filter Control feature	The Filter Control feature is available for each gateway filter element. The Filter Control feature lets you enable and disable the related gateway filter, and specify the filter mode. Refer to <a href="#">Controlling Filters of Frame Captures and Frame Gateways</a> on page 227.

---

#### Restrictions for assigning gateway bus access requests to bus accesses

When you assign gateway bus access requests to [bus accesses](#), you must ensure that the bus accesses are not connected in a loop. If bus accesses are connected in a loop, the bus communication is gatewayed in an endless, multiplying loop that results in a high bus load.

When you generate bus simulation containers and import the BSC files in the VEOS Player, the bus access requests are available as communication controllers. To avoid loops, you must assign the communication controllers to separate bus accesses, i.e., communication clusters.

---

#### Restrictions for specifying frame gateways

The following restrictions apply for specifying frame gateways:

- You cannot specify frame gateways for LIN clusters.
- In global time synchronization (GTS) scenarios, the time base instances of affected time slaves are not completely synchronous to the time base instance of the related time master. For more information, refer to [Implementing Global Time Synchronization in Executable Applications](#) on page 88.

---

#### Related topics

##### Basics

[Use Scenarios for Configuring Bus Communication with the Bus Manager.....](#) 20

# Specifying Filters for Frame Captures and Frame Gateways

## Introduction

Frame captures and frame gateways provide filters that let you filter CAN frames for capturing and gatewaying, respectively. To specify a filter, you must add one or more filter rules to the filter. Each filter rule lets you filter CAN frames by their frame triggerings.

## Filters of frame captures and frame gateways

For frame captures and frame gateways, the following filters are available:

- For each frame capture, one filter is available.

Name	Element Type
Bus Configuration (1)	Bus Configuration
Simulated ECUs	Bus Configuration Part Simulated ECUs
Inspection	Bus Configuration Part Inspection
Frame Capture (1)	RX Bus Frame Capture
Frame Capture Data	RX Bus Frame Capture Data Inspection
CAN Cluster Filter	RX Bus Frame Capture Filter
Manipulation	Bus Configuration Part Manipulation
Gateways	Bus Configuration Part Gateways

- For each frame gateway, two filters are available. Each filter is unambiguously assigned to one communication cluster of the frame gateway, i.e., the communication clusters are filtered as follows:
  - CAN Cluster 1 Filter filters the CAN frames of CAN Cluster 1.
  - CAN Cluster 2 Filter filters the CAN frames of CAN Cluster 2.

Name	Element Type	Name	Element Type
Bus Configuration (1)	Bus Configuration	Bus Configuration (1)	Bus Configuration
Simulated ECUs	Bus Configuration Part Simulated ECUs	Bus Access Requests	Bus Configuration Part Bus Access ...
Inspection	Bus Configuration Part Inspection	Frame Gateway (1)	Bus Frame Gateway
Manipulation	Bus Configuration Part Manipulation	CAN	Bus System CAN
Gateways	Bus Configuration Part Gateways	CAN Cluster 1	Bus CAN Frame Gateway Cluster
Frame Gateway (1)	Bus Frame Gateway	CAN Cluster 2	Bus CAN Frame Gateway Cluster
Frame Gateway Direction	Bus Frame Gateway Direction	Bus Access Request...	Bus Access Request Gateways
CAN Cluster 1 Filter	Bus Frame Gateway Filter	CAN Cluster 2 Filter	Bus Frame Gateway Filter
CAN Cluster 2 Filter	Bus Frame Gateway Filter	Bus Access Request...	Bus Access Request Gateways

You cannot change this assignment.

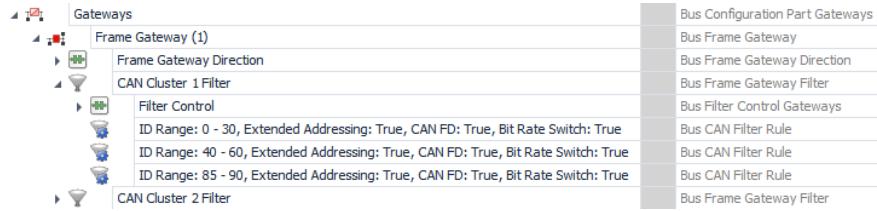
## Specifying filter rules

**Adding filter rules** To each filter, you can add one or more filter rules via the Add CAN Filter Rule context menu command, as shown in the following example.



For each filter rule, a separate filter rule element is added to the filter. The names of the filter rule elements are derived from the specified filter settings.

If you add two or more filter rules to a filter, all frames that match at least one of the filter rules pass the filter.



**Specifying filter settings for filter rules** Filter rules let you filter CAN frames by their frame triggering. For this purpose, each filter rule element provides the following properties:

Property	Value Range	Description
Extended addressing	Checkbox: <ul style="list-style-type: none"><li>▪ Selected</li><li>▪ Cleared</li></ul>	Lets you filter frames by their identifier format: <ul style="list-style-type: none"><li>▪ Selected: Frames with extended identifier format (29-bit) pass the filter.</li><li>▪ Cleared: Frames with standard identifier format (11-bit) pass the filter.</li></ul>
Minimum identifier value	0 ... 536870911	Lets you specify the minimum and maximum identifier value to filter frames on the basis of an identifier value range. Frames whose identifier value is within the specified value range pass the filter.
Maximum identifier value		
CAN FD frame support	Checkbox: <ul style="list-style-type: none"><li>▪ Selected</li><li>▪ Cleared</li></ul>	Lets you filter frames by their CAN FD support status: <ul style="list-style-type: none"><li>▪ Selected: CAN FD frames pass the filter.</li><li>▪ Cleared: Classic CAN frames pass the filter.</li></ul>
Bit rate switch	Checkbox: <ul style="list-style-type: none"><li>▪ Selected</li><li>▪ Cleared</li></ul>	Only for CAN FD frames: Lets you filter CAN FD frames by their bit rate switch: <ul style="list-style-type: none"><li>▪ Selected: CAN FD frames with enabled bit rate switch pass the filter.</li><li>▪ Cleared: CAN FD frames with disabled bit rate switch pass the filter.</li></ul> Classic CAN frames are not affected by this filter property.

Only frames whose frame triggering matches all of the specified settings pass the filter.

#### Note

To each filter rule, all of the settings apply for filtering frames. You cannot disable individual settings for frame filtering.

#### Run-time behavior of the filters

By default, frame capture and frame gateway filters are active at run time. Frames that pass at least one filter rule of a filter are not captured or gatewayed while all other frames of the related cluster are. You can change the default

behavior by using the Filter Control feature. Refer to [Controlling Filters of Frame Captures and Frame Gateways](#) on page 227.

---

**Restrictions for using frame gateway filters**

Frame gateway filters do not support J1939 transport protocols. J1939 transport protocols use specific ID ranges to transmit verifying information, e.g., *Request to Send (RTS)*, *Clear to Send (CTS)*, and related data packets. Frames can be filtered only on the basis of their frame triggering, not on the basis of their payload. Therefore, J1939 communication can be significantly disturbed, e.g., if an RTS passes a frame gateway filter while a related CTS is blocked.

For more information on J1939 transport protocols, refer to [Aspects of Supported CAN Bus Features](#) on page 47.



# Working with Bus Configuration Features

---

Where to go from here	Information in this section
	Basics on Working with Bus Configuration Features..... 116
	Manipulating Bus Communication via Bus Configuration Features..... 131
	Bus Configuration Features Available for ISignals..... 139
	Bus Configuration Features Available for ISignal Groups..... 152
	Bus Configuration Features Available for PDUs..... 155
	Bus Configuration Features Available for Frames..... 190
	Bus Configuration Features Available for Communication Controllers..... 203
	Bus Configuration Features Available for Global Time Domains..... 214
	Bus Configuration Features Available for Frame Captures and Frame Gateways..... 222
	Bus Configuration Features Available for Bus Configurations..... 230

# Basics on Working with Bus Configuration Features

## Where to go from here

## Information in this section

Basics on Bus Configuration Features.....	116
Configuring Function Ports for Bus Configuration Features.....	120
Accessing Function Ports with Enabled Test Automation Support in Variable Description Files.....	124
Triggering the Transmission of PDUs and Frames via User-Defined Triggers.....	128

## Basics on Bus Configuration Features

### Introduction

You can add bus configuration features to certain elements of a [bus configuration](#). The bus configuration features provide element-specific functionalities which you can configure for run time.

### Overview of the bus configuration features

Depending on the bus configuration element (e.g., ISignal, PDU) and the bus configuration part (i.e., Simulated ECUs, Inspection, Manipulation, or Gateways) an element assigned to, the following bus configuration features are available:

Bus Configuration Feature	Purpose	Available for Elements Assigned to	Further Information
<b>For ISignals and ISignal Groups</b>			
ISignal Value	Lets you access the signal value of an ISignal.	<ul style="list-style-type: none"> <li>▪ Simulated ECUs</li> <li>▪ Inspection</li> </ul>	<a href="#">Working with ISignal Values on page 139</a>
Counter Signal	Lets you specify an ISignal as a counter signal.	Simulated ECUs	<a href="#">Working with Counter Signals on page 141</a>
ISignal Overwrite Value	Lets you overwrite the value of an ISignal with a user-defined value.	Manipulation	<a href="#">Overwriting ISignal Values on page 146</a>
ISignal Offset Value	Lets you add an offset value to the value of an ISignal.	Manipulation	<a href="#">Adding Offset Values to ISignal Values on page 149</a>
ISignal Group End-to-End Protection Status	Lets you observe the status of a received end-to-end-protected ISignal group.	<ul style="list-style-type: none"> <li>▪ Simulated ECUs</li> <li>▪ Inspection</li> </ul>	<a href="#">Observing the Status of Received End-to-End-Protected ISignal Groups on page 152</a>

Bus Configuration Feature	Purpose	Available for Elements Assigned to	Further Information
<b>For PDUs</b>			
PDU Enable	Lets you enable and disable the transmission of a PDU.	Simulated ECUs	<a href="#">Enabling and Disabling the Transmission of PDUs on page 155</a>
PDU Raw Data	Lets you access the payload of a PDU in raw data format.	<ul style="list-style-type: none"> <li>▪ Simulated ECUs</li> <li>▪ Inspection</li> </ul>	<a href="#">Accessing the Payload of PDUs in Raw Data Format on page 157</a>
PDU Cyclic Timing Control	Lets you control the cyclic timing of a CAN PDU.	Simulated ECUs	<a href="#">Controlling the Cyclic Timing of CAN PDUs on page 159</a>
PDU Trigger	Lets you trigger the transmission of a PDU according to a user-defined trigger.	Simulated ECUs	<a href="#">Specifying User-Defined Triggers for Transmitting PDUs on page 162</a>
PDU Length	Lets you access the payload length of a CAN PDU, e.g., to simulate a dynamic length CAN PDU.	Simulated ECUs	<a href="#">Accessing the Payload Length of CAN PDUs on page 165</a>
PDU RX Status	Lets you observe the status of a received PDU.	<ul style="list-style-type: none"> <li>▪ Simulated ECUs</li> <li>▪ Inspection</li> </ul>	<a href="#">Observing the Status of Received PDUs on page 167</a>
PDU User Code	Lets you apply user code to a PDU.	<ul style="list-style-type: none"> <li>▪ Simulated ECUs</li> <li>▪ Inspection</li> <li>▪ Manipulation</li> </ul>	<a href="#">Applying User Code to PDUs on page 170</a>
SecOC	Lets you verify the authentication information of a received secured IPDU.	<ul style="list-style-type: none"> <li>▪ Simulated ECUs</li> <li>▪ Inspection</li> </ul>	<a href="#">Verifying the Authentication Information of Received Secured IPDUs on page 175</a>
SecOC Authenticator Invalidation	Lets you invalidate the authenticator before transmitting a secured IPDU.	Manipulation	<a href="#">Invalidating the Authenticator of Secured IPDUs on page 177</a>
SecOC Freshness Overwrite Value	Lets you overwrite the freshness value of a secured IPDU with a user-defined freshness value.	Manipulation	<a href="#">Overwriting the Freshness Value of Secured IPDUs on page 180</a>
PDU RX Interrupt	Lets you use RX interrupts to trigger the execution of functions in a behavior model.	Simulated ECUs	<a href="#">Triggering the Execution of Functions in a Behavior Model via RX Interrupts on page 183</a>
<b>For Frames</b>			
Frame Access	Lets you access settings of a CAN frame.	Simulated ECUs	<a href="#">Accessing CAN Frame Settings on page 190</a>
Frame Length	Lets you manipulate the payload length of a CAN frame.	Manipulation	<a href="#">Manipulating the Payload Length of CAN Frames on page 197</a>
Suspend Frame Transmission	Lets you suspend the transmission of a frame.	Manipulation	<a href="#">Suspending the Transmission of Frames on page 200</a>
<b>For Communication Controllers</b>			
Communication Controller Enable	Lets you enable and disable a communication controller of an ECU.	Simulated ECUs	<a href="#">Enabling and Disabling Communication Controllers on page 203</a>
LIN Wake-Up	Lets you send LIN wake-up signals on the LIN bus.	Simulated ECUs	<a href="#">Sending Wake-Up Signals on a LIN Bus on page 205</a>
LIN Schedule Table	Lets you access the schedule tables of a LIN master.	Simulated ECUs	<a href="#">Working with LIN Schedule Tables on page 207</a>

Bus Configuration Feature	Purpose	Available for Elements Assigned to	Further Information
J1939 Network Management Enable	Lets you enable and disable J1939 network management for a network node and specify the enable state.	Simulated ECUs	<a href="#">Enabling and Disabling J1939 Network Management</a> on page 211
<b>For Global Time Domains</b>			
GTS Transmission Control	Lets you control the timing of time synchronization.	Simulated ECUs	<a href="#">Controlling the Timing of Time Synchronization</a> on page 214
GTS Time Base Data	Lets you access the time base data of time masters and time slaves.	Simulated ECUs	<a href="#">Accessing the Time Base Data of Time Masters and Time Slaves</a> on page 216
GTS Validation	Lets you access the validity checks for time synchronization messages.	Simulated ECUs	<a href="#">Accessing Validity Checks for Time Synchronization Messages</a> on page 218
<b>For Frame Captures, Gateways, and Bus Configurations</b>			
Frame Capture Data	Lets you access the data that is captured by a CAN frame capture.	Inspection	<a href="#">Accessing the Data of Captured Frames</a> on page 222
Frame Gateway Direction	Lets you specify the gateway direction of a CAN frame gateway.	Gateways	<a href="#">Specifying the Direction of CAN Frame Gateways</a> on page 226
Filter Control	Lets you specify the filter mode of a frame capture or frame gateway filter, and enable or disable the filter.	<ul style="list-style-type: none"> <li>▪ Inspection</li> <li>▪ Gateways</li> </ul>	<a href="#">Controlling Filters of Frame Captures and Frame Gateways</a> on page 227
Bus Configuration Enable	Lets you enable and disable a bus configuration.	- (available only for bus configuration nodes)	<a href="#">Enabling and Disabling Bus Configurations</a> on page 230

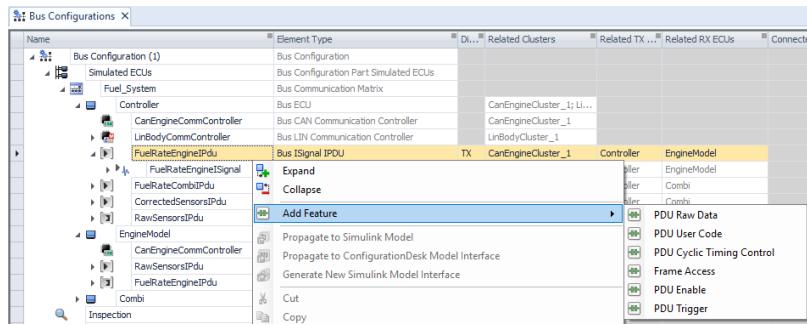
**Tip**

In addition to the bus configuration features, you can influence the bus communication by specifying user-defined settings for certain communication matrix elements. Refer to [Modifying Communication Matrices](#) on page 267.

**Adding bus configuration features**

There are two ways to add bus configuration features:

- You can select a bus configuration element in the Bus Configurations, Bus Simulation Features, Bus Inspection Features, or Bus Manipulation Features table and add an available bus configuration feature via the context menu.

**Tip**

You can also add bus configuration features via context menu for multi-selected elements, even if their element types differ. When you select a feature from the context menu, it is added to all selected elements for which it is valid.

- To add a bus configuration feature to a PDU or ISignal, you can select the PDU or ISignal in the PDU Features or Signal Features subview of the Bus Simulation Features, Bus Inspection Features, or Bus Manipulation Features table and enable an available bus configuration feature in the related column.

Bus Simulation Features															
Signal Features															
Name	Element Type	Direction	Bus Configuration	Communication...	ECU	PDU	(Signal Value)	Counter Signal	Minimum Value	Maximum Value	Increment Value	Initial Value	Step Length	Related Clusters	Related TX
KL_1STSignal	ISignal	TX	Bus Configuration...	BusManagerDemo	CentralGat...	GeneralInfr...	Disabled	GearBoxInfr...	0	255	1	0	1	CarBodyCluster	CentralG...
CurrentGearSignal	ISignal	TX	Bus Configuration...	BusManagerDemo	CentralGat...	GeneralInfr...	Enabled	GearBoxInfr...	0	255	1	0	1	CarBodyCluster; Can...	GearBoxE
SpeedSignal	ISignal	TX	Bus Configuration...	BusManagerDemo	CentralGat...	GeneralInfr...	Disabled	GearBoxStat...	0	100	1	0	1	CarBodyCluster; Can...	GearBoxE
KL_1STSignal	ISignal	RX	Bus Configuration...	BusManagerDemo	CentralGat...	PowerStat...	Enabled	DoorLeftSt...	0	1	1	0	1	CarPowerTrainCluster	CentralGe...
DoorLeftClosedSignal	ISignal	RX	Bus Configuration...	BusManagerDemo	CentralGat...	PowerStat...	Disabled	DoorLeftSt...	0	1	1	0	1	CarBodyCluster	BodyCont...

**Tip**

- The ISignal Value feature and LIN Schedule Table feature are added automatically when you assign a related communication matrix element (i.e., an ISignal or a LIN master communication controller) to the Simulated ECUs part of a bus configuration.
- The Frame Capture Data and Filter Control features are added automatically when you add a frame capture to the Inspection part of a bus configuration.
- The Frame Gateway Direction and Filter Control features are added automatically when you add a frame gateway to the Gateways part of a bus configuration.

## Effects of added bus configuration features

A bus configuration feature applies only to the bus configuration element for which it was added. The related communication matrix, instances of the element in the related and other bus configurations, and other bus configuration elements are not affected by the bus configuration feature.

Each bus configuration feature provides feature-specific properties and/or function ports. Depending on the available properties and function ports, you can configure the bus configuration feature and/or specify initial settings for the

feature. Via the function ports, you can access the related feature settings at run time.

Depending on the bus configuration feature for which function ports are added, you cannot access the related feature at run time by default. To access such features at run time, you must change the function port default settings. For more information, refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

**Tip**

If no bus configuration feature is added to a bus configuration, the bus configuration does not have any function ports.

---

**Disabling bus configuration features**

You can disable bus configuration features if you do not need them. Disabling a bus configuration feature removes the related function ports from the bus configuration. You can disable a bus configuration feature in the following ways:

- Right-click on a feature node ( ) in the Bus Configurations table and choose Delete from Application from the context menu.

**Tip**

You can also press the **Delete** key to delete a selected feature node.

- To disable a bus configuration feature for a PDU or ISignal, you can also select the PDU or ISignal in the Bus Simulation Features, Bus Inspection Features, or Bus Manipulation Features table and choose Disabled from the list of the related table column.

## Configuring Function Ports for Bus Configuration Features

---

**Introduction**

Some settings of bus configuration features can be accessed via function ports. The behavior of these settings is determined by the related function ports. For the function ports, you can configure the following settings to specify the behavior of the related bus configuration feature settings.

---

**Specifying initial values**

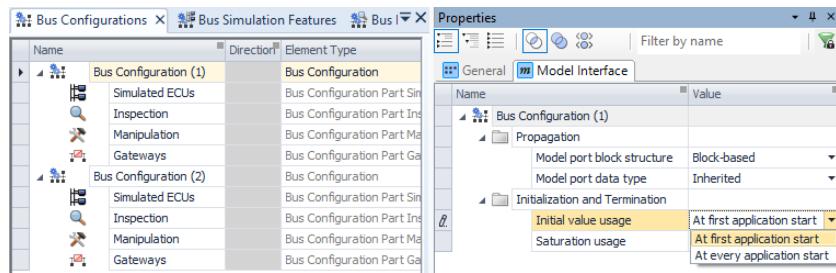
Every function port provides a default initial value that is specific for the related bus configuration feature. Via the function port's **Initial value** property, you can specify a user-defined initial value.

**Note**

Depending on the related bus configuration feature, a user-defined initial value might be saturated, for example, if you replace or modify a communication matrix.

The function port value is set to the initial value during system initialization. This value is used directly after an [executable application](#) is started. During run time, an initial function port value can be overwritten, for example, by a value that is received on the bus, provided by a [behavior model](#), or changed via experiment software.

Each bus configuration provides an **Initial value usage** property. This property lets you specify how the initial values of the bus configuration's function ports are used. You can specify whether the initial values are set only once at first application start or every time the application starts, including when the application state switches from stopped to running.

**Tip**

- If you enable test automation support, you can additionally specify an initial substitute value. You can select whether the value of the **Initial value** or **Initial substitute value** property is used as the initial function port value. Refer to [Enabling test automation support](#) on page 122.
- The setting of the **Initial value usage** property applies to both initial values, i.e., to the values of the **Initial value** and **Initial substitute value** properties.

**Enabling model access**

To exchange data between bus configuration features and a behavior model, you must set the **Model access** property for the related function ports to **Enabled** and map the function ports to [model port blocks](#). This lets you, for example:

- Use signal values that are dynamically calculated by the control algorithms of the behavior model.
- Provide status information to the behavior model.

By default, model access is disabled for all function ports that are available for bus configuration features.

**Note**

Enabling model access increases the complexity of Bus Configuration function blocks and the [signal chain](#). For optimum performance, it is therefore recommended to enable model access only for those function ports whose values you want to exchange with the behavior model.

**Enabling test automation support**

**Basics on test automation support** To access bus configuration features via experiment software such as ControlDesk or automation scripts, you must set the **Activate test automation support** property for the related function ports to Enabled. This lets you, for example:

- Switch the signal source, i.e., use the original signal or a substitute signal.
- Manipulate signal values via experiment software.
- Carry out test automation tasks.

When test automation support is enabled for a function port, you can specify the initial signal source (i.e., the original signal or substitute signal) via the **Initial switch setting** property.

For the substitute signal, each function port provides a default initial substitute value that is specific for the related bus configuration feature. Via the **Initial substitute value** property, you can specify a user-defined initial substitute value.

**Tip**

For the values of the **Initial substitute value** and **Initial value** properties, the same rules apply. For example, if the default value of the **Initial value** property is derived from the communication matrix, this also applies to the default value of the **Initial substitute value** property.

**Note**

Depending on the related bus configuration feature, a user-defined initial substitute value might be saturated, for example, if you replace or modify a communication matrix.

When test automation support is enabled, variables are generated into the variable description file when you generate bus simulation containers. The variables let you access the function port via experiment software and automation scripts. Refer to [Accessing Function Ports with Enabled Test Automation Support in Variable Description Files](#) on page 124.

**Default enable state** The default enable state of test automation support depends on the bus configuration feature for which a function port is available and the function port type, as shown in the following table.

Bus Configuration Feature	Function Port Type	Default Settings of Function Ports	
		Test Automation Support	Initial Switch Setting
Bus simulation features	In	Disabled	-
	Out	<ul style="list-style-type: none"> <li>▪ Disabled</li> <li>▪ Only for the PDU Raw Data function port (PDU Raw Data feature): Enabled</li> </ul>	Only for the PDU Raw Data function port (PDU Raw Data feature): I/O signal
Bus inspection features	In	Enabled	Substitute value
	Out	Enabled	I/O signal
Bus manipulation features	In	Enabled	Substitute value
	Out	Enabled	I/O signal
Bus gateway features	In	Disabled	-

**Tip**

For bus gateway features, no function outports are available.

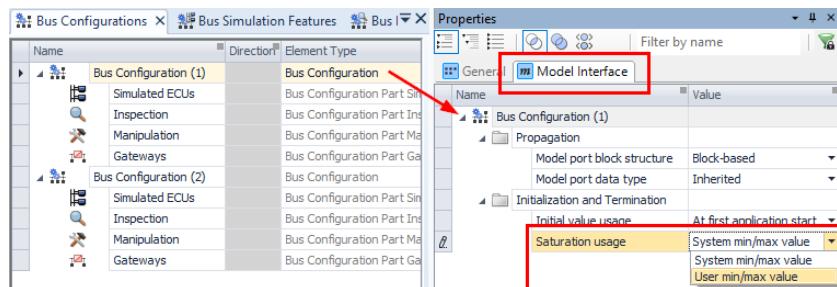
### Specifying saturation values for function imports

If required, the values of function imports are saturated at run time. By default, system-specific saturation limits are used. Depending on the related bus configuration feature, the system-specific limits are derived from the function import's data type, for example. For some function imports, you can specify user-defined saturation values to limit the minimum and maximum values of the function port.

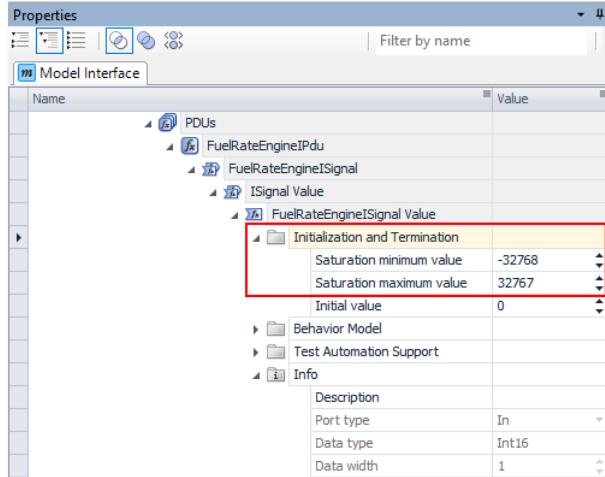
**Note**

Depending on the related bus configuration feature, user-defined saturation values might be saturated, for example, if you replace or modify a communication matrix.

To use user-defined saturation values, you must set the **Saturation usage** property of the related bus configuration to **User min/max value**, as shown in the following example.



If you do this, the function imports that let you specify user-defined saturation values provide a Saturation minimum value and Saturation maximum value property, as shown in the following example.



The properties let you specify user-defined minimum and maximum saturation values in the range of the function import's value range. If necessary, the Bus Manager automatically adjusts the specified saturation values to avoid overflows at run time, for example, when an ISignal is encoded in a PDU and the specified saturation values exceed the ISignal's boundaries.

For details on working with saturation values, refer to [Specifying User Saturation \(ConfigurationDesk I/O Function Implementation Guide\)](#).

## Related topics

## References

[Bus Configuration \(ConfigurationDesk Function Block Properties\)](#)

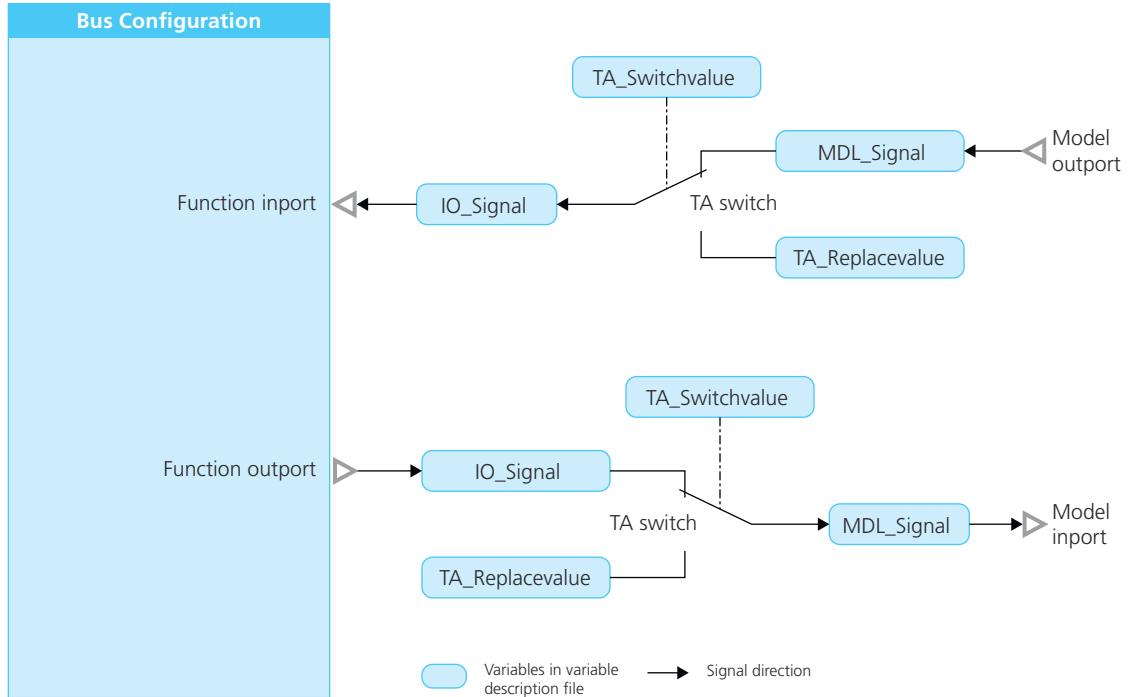
## Accessing Function Ports with Enabled Test Automation Support in Variable Description Files

### Introduction

For function ports with enabled test automation support, variables are generated into the variable description file ([TRC file](#)) when you generate bus simulation containers. The variables let you access the function ports via experiment software and automation scripts.

**Overview of the variables**

In general, up to four variables can be generated for each function port with enabled test automation support, as shown in the following example.

**Tip**

Examples of the function port types are:

- Function imports: TX ISignal Value function ports
- Function outports: RX ISignal Value function ports

The variables are used as follows:

Variable	Purpose
<b>IO_Signal</b>	<ul style="list-style-type: none"> <li>Function import: Provides the input signal to the function port.</li> <li>Function outport: Provides the output signal of the function port.</li> </ul>
<b>MDL_Signal</b>	<ul style="list-style-type: none"> <li>Function import: Provides the output signal of a mapped model port, i.e., of the related function in the behavior model.</li> <li>Function outport: Provides the input signal to a mapped model port, i.e., to the related function in the behavior model.</li> </ul>
<b>TA_Switchvalue</b>	<p>Lets you switch the signal source between the original signal or a substitute signal. The original signal depends on the function port type:</p> <ul style="list-style-type: none"> <li>Original signal of a function import: Signal of the <b>MDL_Signal</b> variable.</li> <li>Original signal of a function outport: Signal of the <b>IO_Signal</b> variable.</li> </ul>
<b>TA_Replacevalue</b>	Provides the value of the substitute signal.

However, the configuration of each function port determines which variables are generated. By default, an optimized set of variables is generated for each bus configuration function port.

#### Optimized set of variables

When you add a bus configuration to the ConfigurationDesk application, an optimized set of variables is generated for each of its function ports by default. This ensures optimum run-time performance.

##### Note

For bus configurations that were added to the ConfigurationDesk application with dSPACE Release 2020-B or earlier, the default setting differs. Refer to [Bus configurations added with dSPACE Release 2020-B or earlier](#) on page 127.

For each function port of the bus configuration, only those variables are generated into the TRC file that are typically used in common bus use scenarios. To determine these variables, the following aspects are considered:

- Function port type, i.e., function input or function output
- Function port mapped to model port

The optimized set of variables also determines the value that is used as the initial function port value.

The following table provides an overview of the optimized set of variables.

Function Port Type	Conditions	Generated Variables	Initial Function Port Value
In	Mapped to model port	<ul style="list-style-type: none"> <li>▪ IO_Signal</li> <li>▪ MDL_Signal</li> <li>▪ TA_Replacevalue</li> <li>▪ TA_Switchvalue</li> </ul>	Depends on the Initial switch setting property.
	Not mapped to model port and Model access set to Disabled	<ul style="list-style-type: none"> <li>▪ IO_Signal</li> <li>▪ TA_Replacevalue</li> </ul>	Value specified for the Initial substitute value property.
	Not mapped to model port and Model access set to Enabled	<ul style="list-style-type: none"> <li>▪ IO_Signal</li> <li>▪ MDL_Signal</li> <li>▪ TA_Replacevalue</li> <li>▪ TA_Switchvalue</li> </ul>	Depends on the Initial switch setting property.
Out	Mapped to model port	<ul style="list-style-type: none"> <li>▪ IO_Signal</li> <li>▪ MDL_Signal</li> </ul>	Value specified for the Initial value property.
	Not mapped to model port	IO_Signal	Value specified for the Initial value property.

For details on the optimized set of variables, refer to [Optimized Set of Variables for Bus Configuration Function Ports](#) on page 338.

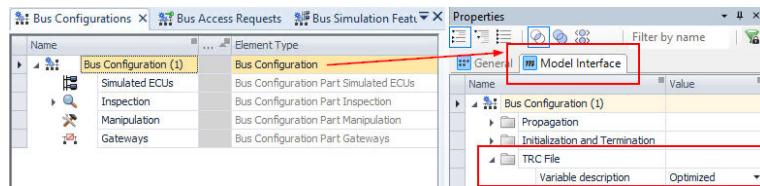
#### Specifying the set for generating variables

Each bus configuration provides a Variable description property that lets you specify whether the optimized set of variables or a complete set is generated.

**Note**

For optimum run-time performance, it is strongly recommended to generate the optimized set of variables.

When you select a bus configuration (e.g., in the Bus Configurations table), the Variable description property is available on the Model Interface page in the Properties Browser.



The Variable description property lets you specify which set of variables to generate for all function ports of the bus configuration:

- Optimized: The optimized set of variables is generated.
- Complete: The complete set of variables is generated.

You can use bus configurations with different settings of the Variable description property in the same ConfigurationDesk application.

**Complete set of variables**

If you set the Variable description property to Complete, the following variables are generated:

Function Port Type	Condition	Generated Variables
In	-	<ul style="list-style-type: none"> <li>▪ IO_Signal</li> <li>▪ MDL_Signal</li> <li>▪ TA_Replacevalue</li> <li>▪ TA_Switchvalue</li> </ul>
Out	Mapped to model port	<ul style="list-style-type: none"> <li>▪ IO_Signal</li> <li>▪ MDL_Signal</li> <li>▪ TA_Replacevalue</li> <li>▪ TA_Switchvalue</li> </ul>
	Not mapped to model port	IO_Signal

For more information, refer to [Configuring Test Automation Support \(ConfigurationDesk I/O Function Implementation Guide\)](#)

**Bus configurations added with dSPACE Release 2020-B or earlier**

If the ConfigurationDesk application contains bus configurations that were added with dSPACE Release 2020-B or earlier, the Variable description property of these bus configurations is set to Complete. For optimum run-time performance, it is recommended to set the Variable description property of the bus configurations to Optimized. If you do this and you use automation scripts to access variables of TRC files, you might have to adapt the automation scripts.

<b>Restrictions for the optimized set of variables</b>	<p>For the function ports of the following bus configuration features, no optimized set of variables can be generated:</p> <ul style="list-style-type: none"> <li>▪ Frame Capture Data feature</li> <li>▪ Filter Control feature</li> <li>▪ LIN Schedule Table feature</li> </ul> <p>Instead, the complete set of variables is always generated for the function ports of these features, even if the Variable description property is set to Optimized.</p>
--	--

## Triggering the Transmission of PDUs and Frames via User-Defined Triggers

### Introduction

Typically, the transmission of PDUs and frames is triggered according to specifications in the communication matrix, such as cyclic timings. Via the PDU Trigger and Frame Access feature, you can specify user-defined triggers to trigger the transmission of PDUs and frames. This lets you transmit PDUs and frames continuously or asynchronously to a cyclic timing, for example.

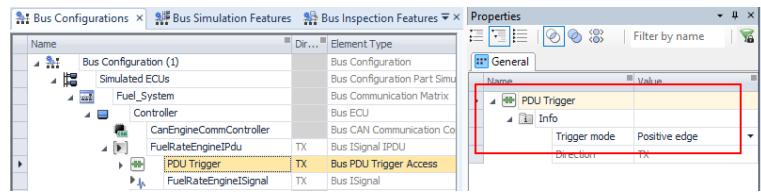
### Specifying user-defined triggers for the transmission of PDUs and frames

Specifying user-defined triggers for transmitting a PDU or frame comprises the following basic steps:

1. Add the PDU Trigger or Frame Access feature to the PDU. Refer to [Adding bus configuration features](#) on page 118.  
For more information on the features, refer to [Specifying User-Defined Triggers for Transmitting PDUs](#) on page 162 and [Accessing CAN Frame Settings](#) on page 190, respectively.
2. Select a triggering mode. Refer to [Triggering modes for triggering the transmission of PDUs and frames](#) on page 128.
3. Configure the Trigger function port. Refer to [Configuring Trigger function ports](#) on page 129.

### Triggering modes for triggering the transmission of PDUs and frames

When you add the PDU Trigger or Frame Access feature to a PDU, a Trigger mode property is available, as shown in the following example.



Via the Trigger mode property, you can select one of the following triggering modes to specify the trigger behavior:

- Positive edge: The transmission of the PDU or frame is triggered once each time a positive edge is detected, i.e., when the value of the related Trigger function port changes from 0 to 1.

**Tip**

If the function port value was set via an experiment software during run time, the function port value is automatically reset to 0 after the PDU/frame was triggered.

- **Level-triggered:** The transmission of the PDU or frame is triggered continuously as long as the value of the related Trigger function port is set to 1. The transmission of the PDU/frame stops when the function port value is set to 0 via model input or experiment software.

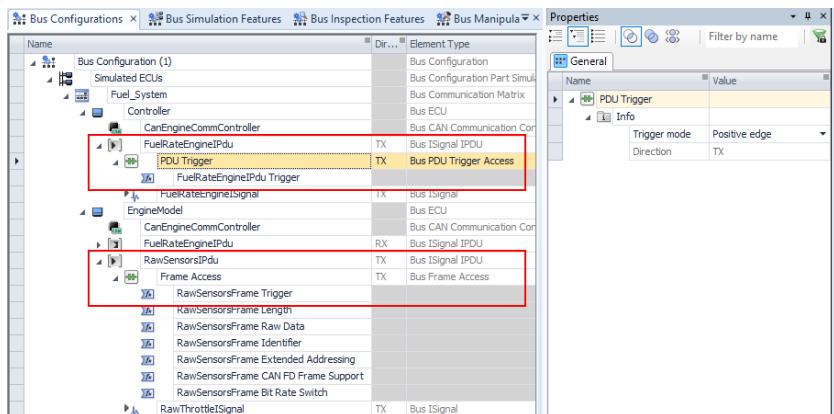
The selected triggering mode does not influence the timing of the transmission. This is determined as follows:

- For triggering the transmission of PDUs via the PDU Trigger feature, the timing depends on the related Bus Configuration task and the specifications in the communication matrix.
- For triggering the transmission of frames via the Frame Access feature, the timing depends on the related Bus Configuration task. Specifications of the communication matrix do not influence the timing. When the Bus Configuration task evaluates the trigger, the frame data is calculated according to the specified settings of the Frame Access feature and the frame is transmitted on the bus.

For more information on the Bus Configuration task, refer to [Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager](#) on page 94.

## Configuring Trigger function ports

When you add the PDU Trigger or Frame Access feature to a PDU, a Trigger function port is available that lets you trigger the transmission of the PDU or its related frame at run time.



The default settings of the Trigger function port prohibit the transmission of the related PDU or frame during run time. Depending on the selected triggering mode, you must therefore change the default settings of the Trigger function port to trigger the transmission of the PDU/frame during run time:

- If you selected Positive edge as the triggering mode, the transmission of the PDU/frame is triggered when the function port value changes from 0 to 1

during run time. To change the function port value during run time, you must enable model access or test automation support for the function port.

- If you selected Level-triggered as the triggering mode, the transmission of the PDU/frame is triggered when the function port value is 1 during run time. You can specify 1 as a constant value by setting the function port's Initial value to True. Alternatively, you can enable model access or test automation support to change the function port value during run time.

For more information on enabling model access and test automation support, refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

---

**Related topics**

Basics

[Limitations for Communication Matrices and Communication Standards.....](#) 319

# Manipulating Bus Communication via Bus Configuration Features

## Where to go from here

## Information in this section

Basics on Bus Manipulation Features.....	131
Recalculating End-to-End Protection and Authentication Information.....	135
Execution Sequence of Bus Manipulation Features.....	137

## Basics on Bus Manipulation Features

### Introduction

Manipulating bus communication differs slightly from simulating or inspecting bus communication: In addition to feature-specific elements and settings, each bus manipulation feature provides common manipulation elements and settings. These elements and settings influence the run-time behavior of bus communication manipulation.

### Common manipulation elements and settings

**Overview** The following elements and settings are common to bus manipulation features:

Element/Setting	Description	Refer to
Permanent or temporary manipulation	For each bus manipulation feature, you can specify to manipulate the bus communication permanently or temporarily.	<a href="#">Permanent or temporary manipulation on page 132</a>
Feature switch	For bus manipulation features that are available for ISignals, you can specify which feature is active at run time.	<a href="#">Feature switch of ISignals on page 132</a>
Tunable properties	Bus manipulation features let you access their tunable properties in different ways.	<a href="#">Accessing tunable properties on page 133</a>
Recalculate end-to-end protection	For bus manipulation features that affect end-to-end-protected ISignal groups, you can specify whether the end-to-end protection information is recalculated.	<a href="#">Recalculating End-to-End Protection and Authentication Information on page 135</a>
Recalculate authentication information	For bus manipulation features that affect secure onboard communication, you can specify whether the authentication information of secured IPDUs is recalculated.	<a href="#">Recalculating End-to-End Protection and Authentication Information on page 135</a>

**Specifying the manipulation behavior** Via the common manipulation elements and settings of bus manipulation features, you can specify the manipulation behavior in one of the following ways:

- For bus manipulation features that do not apply to ISignals, you can specify the manipulation behavior via the feature-specific feature node, its related function ports, and the Bus Manipulation Features table. Refer to [Specifying the manipulation behavior for bus manipulation features that do not apply to ISignals on page 134](#).
- For bus manipulation features that apply to ISignals, you can specify the manipulation behavior via the ISignal's feature switch, its related function ports, and the Bus Manipulation Features table. Refer to [Specifying the manipulation behavior for bus manipulation features that apply to ISignals on page 134](#).

Additionally, if an element is affected by multiple bus manipulation features at run time, the execution sequence influences the manipulation behavior. Refer to [Execution Sequence of Bus Manipulation Features on page 137](#).

#### Permanent or temporary manipulation

For each bus manipulation feature, you can specify whether it manipulates the related bus configuration element permanently or temporarily at run time:

- Permanent manipulation

The related bus configuration element is manipulated by the bus manipulation feature until the manipulation is explicitly stopped, e.g., via experiment software.

- Temporary manipulation

During temporary manipulation, the bus configuration element is manipulated for a defined number of times. You can specify the number of manipulations via a countdown value. The specified countdown value is decremented with each transmission of the bus configuration element. When the value reaches 0, the manipulation stops and the bus configuration element is transmitted with its regular value, for example. You can use temporary manipulation to test the fault memory of an ECU, for example.

#### Feature switch of ISignals

When you manipulate ISignals via bus manipulation features, only one feature can be active at a time. Therefore, a feature switch is available for each ISignal when you add a bus manipulation feature to the ISignal, as shown in the following example.

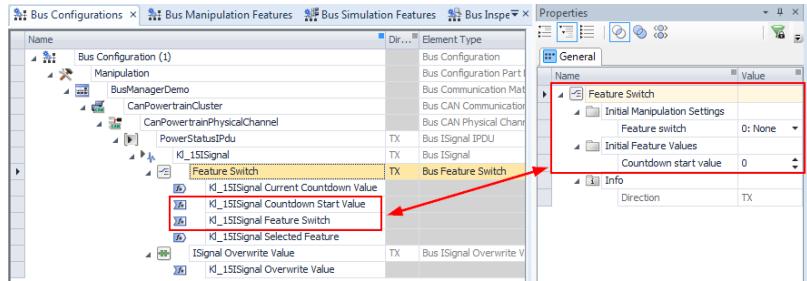
		TX	Bus ISignal
		TX	Bus Feature Switch
	FuelRateEngineISignal		
	Feature Switch		
	FuelRateEngineISignal Current Countdown Value		
	FuelRateEngineISignal Countdown Start Value		
	FuelRateEngineISignal Feature Switch		
	FuelRateEngineISignal Selected Feature		

Via the feature switch, you can specify which of the available bus manipulation features is active and whether it manipulates the ISignal permanently or temporarily. Via the available function ports, you can change the specified settings at run time. When you remove all bus manipulation features from the ISignal, the feature switch is deleted automatically.

## Accessing tunable properties

Tunable properties can be accessed and changed at run time, e.g., via experiment software. Typically, the tunable properties that are available for bus configuration features can be accessed in the ConfigurationDesk application via function ports only. For bus manipulation features, most of the tunable properties that are represented by function imports can additionally be accessed in the following ways:

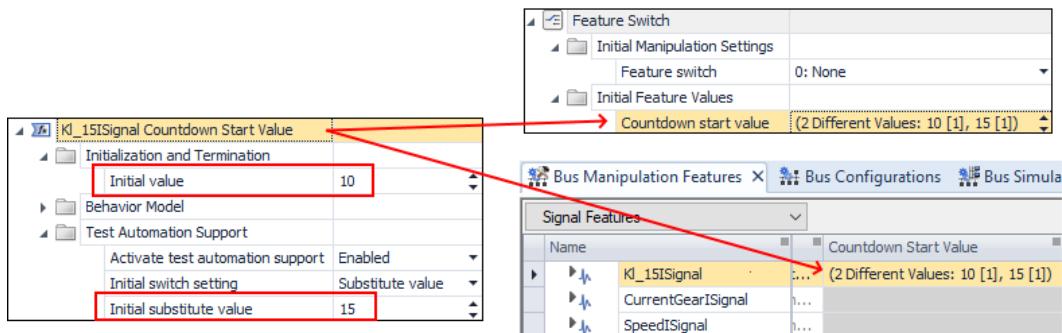
- Via properties of the related feature node () and/or Feature Switch node () in the Bus Configurations table.



- Via the Bus Manipulation Features table

Name	PDU	Countdown Start Value	Feature Switch	Signal Overwrite Value	Overwrite Value	Recalculate End-to-End...	Recalculate SerOC...
KJ_15ISignal	PowerStat...	0	0: None	Enabled			
CurrentGear1Signal	GearBoxIn...			Disabled			
Speed1Signal	GearBoxIn...			Enabled			
FuelConsumption1Signal	EngineInfo...	0	1: ISignal Ove...	Enabled			
EngineSpeed1Signal	EngineInfo...			Disabled			
EngineTemp1Signal	EngineInfo...			Disabled			

The values that are specified via the properties of the related nodes or in the Bus Manipulation Features table apply to the Initial value and Initial substitute value properties of the related function ports and vice versa. If you specify different values for the Initial value and Initial substitute value properties of a function port, the related property and table cell display both values.



For more information on the Initial value and Initial substitute value properties of function ports, refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

**Specifying the manipulation behavior for bus manipulation features that do not apply to ISignals**

You can specify the manipulation behavior by means of various elements. For bus manipulation features that do not apply to ISignals, these elements are available as properties of the feature-specific feature node (H), as function ports, and/or in the Bus Manipulation Features table as follows:

Element Name	Node Property	Function Port	Table Entry	Value Range	Purpose
Enable	✓	✓	–	<ul style="list-style-type: none"> <li>▪ 0: Disabled</li> <li>▪ 1: Permanently enabled</li> <li>▪ 2: Temporarily enabled</li> </ul>	<p>Specifies the initial state of the feature.</p> <p>If the feature is temporarily enabled, it manipulates the related bus configuration element for the number of times that are specified for the Countdown start value. Then, this manipulation feature is disabled. However, the related bus configuration element can still be manipulated by other bus manipulation features.</p> <p>Bus manipulation features are disabled by default.</p>
Enable state	–	✓	–	0 … 255	Provides the current state of the feature, e.g., to a mapped behavior model.
Countdown start value	✓	✓	✓	0 … UInt32 <sub>max</sub>	<p>Specifies the countdown start value. If the bus manipulation feature is temporarily enabled, this value determines how often the bus configuration element is manipulated.</p> <p>If you specify the countdown start value via the Bus Manipulation Features table, the value applies to all manipulation features of the selected bus configuration element. If you specify the countdown start value via a feature node or function port, it applies only to the individual manipulation feature of the related bus configuration element.</p>
Current countdown value	–	✓	–	0 … UInt32 <sub>max</sub>	Provides the current countdown value, e.g., to a mapped behavior model. If the bus manipulation feature is temporarily enabled, the current countdown value is decremented with each transmission of the bus configuration element. When the value reaches 0, the bus manipulation feature is disabled.

For the Enable and Countdown Start Value function ports, you can additionally specify user-defined saturation values to limit the minimum and maximum values of the function ports. Refer to [Specifying saturation values for function imports](#) on page 123.

**Specifying the manipulation behavior for bus manipulation features that apply to ISignals**

You can specify the manipulation behavior by means of various elements. For bus manipulation features that apply to ISignals, these elements are available as properties of the ISignal's Feature Switch node (E), as function ports, and/or in the Bus Manipulation Features table as follows:

Element Name	Node Property	Function Port	Table Entry	Value Range	Purpose
Feature switch	✓	✓	✓	<ul style="list-style-type: none"> <li>▪ 0: None</li> <li>▪ n: &lt;bus manipulation feature 1&gt; (permanently)</li> <li>▪ n+1: &lt;bus manipulation feature 1&gt; (temporarily)</li> <li>▪ m: &lt;bus manipulation feature 2&gt; (permanently)</li> <li>▪ m+1: &lt;bus manipulation feature 2&gt; (temporarily)</li> </ul>	<p>Specifies the initially active feature and its state. If the active feature is temporarily enabled, it manipulates the related ISignal for the number of times that is specified for the Countdown start value. Then, the manipulation of the ISignal stops, i.e., the feature switch is set to 0: None. The default value of the feature switch is 0: None.</p> <p><b>Tip</b></p> <p>The feature switch value unambiguously relates to a specific feature and its state. For example, a value of 3 enables the ISignal Offset Value feature permanently, regardless of whether other manipulation features are added to the ISignal.</p>
Selected feature	–	✓	–	0 ... UInt32 <sub>max</sub>	Provides the number of the currently active feature, e.g., to a mapped behavior model.
Countdown start value	✓	✓	✓	0 ... UInt32 <sub>max</sub>	Specifies the countdown start value. If the active bus manipulation feature is temporarily enabled, this value determines how often the ISignal is manipulated.
Current countdown value	–	✓	–	0 ... UInt32 <sub>max</sub>	Provides the current countdown value, e.g., to a mapped behavior model. If the active bus manipulation feature is temporarily enabled, the current countdown value is decremented with each transmission of the ISignal. When the value reaches 0, the manipulation of the ISignal stops.

For the Feature Switch and Countdown Start Value function ports, you can additionally specify user-defined saturation values to limit the minimum and maximum values of the function ports. Refer to [Specifying saturation values for function imports](#) on page 123.

## Recalculating End-to-End Protection and Authentication Information

### Introduction

In the following cases, using bus manipulation features might result in unintended effects:

- You manipulate ISignals that are included in an end-to-end-protected ISignal group.
- You manipulate PDUs and/or ISignals that are included in a secured IPDU [?](#).

In these cases, the end-to-end protection information of the ISignal group and/or the authentication information of the secured IPDU might become incorrect because the related information is calculated without considering the manipulated data.

To prevent this, you can enable recalculation of the related end-to-end protection and/or authentication information before transmission.

#### Enabling and disabling the recalculation

For bus manipulation features that can affect end-to-end protection and/or authentication information, you can enable and disable the recalculation as follows:

Bus Manipulation Feature Affecting ...	Properties Browser	Bus Manipulation Features Table
End-to-end protection information	Via the Recalculate end-to-end protection property that is available for the feature node  .	In the Recalculate End-to-End Protection (<feature>) column.
Authentication information	Via the Recalculate SecOC information property that is available for the feature node  .	In the Recalculate SecOC Information (<feature>) column.

However, you cannot disable the recalculation of the authentication information for the SecOC Freshness Overwrite Value feature. Refer to [Overwriting the Freshness Value of Secured IPDUs](#) on page 180.

The recalculation of the end-to-end protection and authentication information is enabled by default. You cannot change the specified setting during run time.

#### Run-time behavior for recalculating data

Data is recalculated if an ISignal group or secured IPDU is affected by at least one bus manipulation feature at run time for which recalculation is enabled. This applies even if the ISignal group/secured IPDU is also affected by bus manipulation features for which recalculation is disabled.

For example, a secured IPDU contains an ISignal IPDU with two ISignals, each of which configured as follows:

Element	Bus Manipulation Feature	Recalculate SecOC Information
Secured IPDU	SecOC Authenticator Invalidiation	-
ISignal IPDU	PDU User Code	✓
ISignal 1	ISignal Overwrite Value	✓
ISignal 2	ISignal Overwrite Value	✓
	ISignal Offset Value	-

At run time, the authentication information of the secured IPDU is recalculated as long as the PDU User Code feature of the ISignal IPDU or the ISignal Overwrite Value feature of ISignal 1 or ISignal 2 is enabled permanently or temporarily.

**Tip**

Different enable states for recalculating the related data might be useful if you want to configure an element once for different manipulation scenarios. For example, with the settings above, you can enable and disable the related features at run time to switch between data manipulation that affects the authentication information and data manipulation that does not.

However, recalculating the end-to-end protection or authentication information is performed only once per sampling step, regardless of whether it is enabled for multiple bus manipulation features that affect the ISignal group or secured IPDU. For more information, refer to [Execution Sequence of Bus Manipulation Features](#) on page 137.

#### **Specific aspects for recalculating authentication information**

If a secured IPDU is configured as a cryptographic IPDU, the cryptographic IPDU and its related authentic IPDU are required for recalculating the authentication information. This results in the following:

- The authentication information can be recalculated only if both IPDUs are assigned to the Manipulation part of the same bus configuration.
- The authentication information cannot be recalculated if both IPDUs are contained in a static container IPDU and the cryptographic IPDU triggers the transmission of the container IPDU before the authentic IPDU is included in the container IPDU.
- If the transmission of the cryptographic IPDU is triggered first at run time, the transmission is delayed until the data of the authentic IPDU is available. After the authentication information is recalculated, both IPDUs are transmitted in the original order.

#### **Related topics**

##### Basics

[Aspects of Supported AUTOSAR Features.....](#) 39

## Execution Sequence of Bus Manipulation Features

#### **Introduction**

Bus configuration elements can be affected by multiple bus manipulation features. In this case, the data that is provided to bus manipulation features and/or transmitted on the bus is determined by the sequence in which the features are executed.

This particularly applies if end-to-end protection or authentication information is recalculated: For an affected bus configuration element, the recalculation is performed only once per sampling step, regardless of whether recalculation is enabled for multiple bus manipulation features.

**Execution sequence of bus manipulation features**

At run time, bus manipulation features are executed in the following sequence:

1. Suspend Frame Transmission feature
2. Manipulation features that are available for ISignals
3. Recalculate end-to-end protection information
4. SecOC Authenticator Invalidiation feature
5. PDU User Code feature
6. SecOC Freshness Overwrite Value feature and recalculate authentication information
7. Frame Length feature

# Bus Configuration Features Available for ISignals

## Where to go from here

## Information in this section

Working with ISignal Values.....	139
Working with Counter Signals.....	141
Overwriting ISignal Values.....	146
Adding Offset Values to ISignal Values.....	149

## Working with ISignal Values

### Introduction

You can work with the values of [ISignals](#) that are assigned to the Simulated ECUs or Inspection part of a bus configuration.

### Conditions for accessing ISignal values

To access the value of an ISignal, you must add the ISignal Value feature to the ISignal. You can add the feature to the following ISignals:

- TX ISignals
- RX ISignals

For ISignals that are assigned to the Simulated ECUs part of a bus configuration, the ISignal Value feature is added automatically. For ISignals that are assigned to the Inspection part of a bus configuration, you must add the ISignal Value feature manually.

### Effects of adding the ISignal Value feature

Depending on the direction of the ISignal (TX or RX), a TX ISignal Value function port or RX ISignal Value function port is available for the ISignal.

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

	RawSensorsIPdu	TX	Bus ISignal IPDU
	RawThrottleISignal	TX	Bus ISignal
	ISignal Value	TX	Bus ISignal Value Access
	RawThrottleISignal Value		
	FuelRateEngineIPdu	RX	Bus ISignal IPDU
	FuelRateEngineISignal	RX	Bus ISignal
	ISignal Value	RX	Bus ISignal Value Access
	FuelRateEngineISignal Value		

**Accessing ISignal values**

The ISignal Value function port provides the ISignal value as follows:

- For TX ISignals, the function port value specifies the ISignal value that is transmitted on the bus.
- For RX ISignals, the function port provides the received ISignal value, e.g., to a mapped behavior model.

The data type and value range of the function port are determined by the physical base data type of the ISignal.

By default, the initial function port value is the ISignal value that is specified in the [communication matrix](#). If the communication matrix does not specify an ISignal value, 0 is used as the initial function port value.

Via the function port's Initial value property, you can specify a user-defined initial ISignal value. If required, the value of the Initial value property is automatically saturated when the physical base data type of the ISignal changes. This can happen in the following cases:

- You modify the communication matrix. Refer to [Basics on Modifying Communication Matrices](#) on page 267.
- You replace the assigned communication matrix. Refer to [Replacing Assigned Communication Matrices](#) on page 291.

To access the ISignal value at run time, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

---

**Specifying saturation values for TX ISignals**

By default, TX ISignal values are limited by the ranges of the ISignal's physical base data type and automatically saturated at run time if required. Via the TX ISignal Value function port, you can specify user-defined saturation values to limit the minimum and maximum TX ISignal values. If you want to do this, you must first set the bus configuration's Saturation usage property to User min/max value. Refer to [Specifying saturation values for function imports](#) on page 123.

If required, the saturation values are automatically saturated when the physical base data type changes. This can happen in the following cases:

- You modify the communication matrix. Refer to [Basics on Modifying Communication Matrices](#) on page 267.
  - You replace the assigned communication matrix. Refer to [Replacing Assigned Communication Matrices](#) on page 291.
- 

**Restrictions for using the ISignal Value feature**

If you use the ISignal Value feature, the following restrictions apply:

- You cannot use the ISignal Value feature in parallel with the following bus configuration features:
  - Counter Signal feature (applies only to TX ISignals and TX PDUs)
  - PDU Raw Data feature (applies only to TX ISignals and TX PDUs)
  - Frame Access feature

If you add the ISignal Value feature and at least one of the other features to an ISignal or to a PDU in which the ISignal is included, conflicts occur.

### Tip

If you add the ISignal Value, Counter Signal, and PDU Raw Data feature to an RX ISignal and to the RX PDU in which the ISignal is included, the values provided by each feature are updated at run time and can be accessed in parallel.

- Enabling model access and/or test automation support for a high number of ISignals might reduce the performance. For optimum performance, enable these function port properties only if you really need them, i.e.:
  - Enable model access only for those ISignals whose signal values you want to exchange with the behavior model.
  - Enable test automation support only for those ISignals whose signal values you want to access via an experiment software such as ControlDesk.

---

### Related topics

### References

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

[Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Working with Counter Signals

---

### Introduction

You can configure [ISignals](#) that are assigned to the Simulated ECUs part of a bus configuration as counter signals. You can use counter signals to detect lost PDUs or test counters that are implemented in the ECU under test, for example.

---

### Conditions for working with counter signals

To use an ISignal as a counter signal, you must add the Counter Signal feature to the ISignal and specify the available counter settings. You can add the Counter Signal feature to TX and RX ISignals with the following base data types:

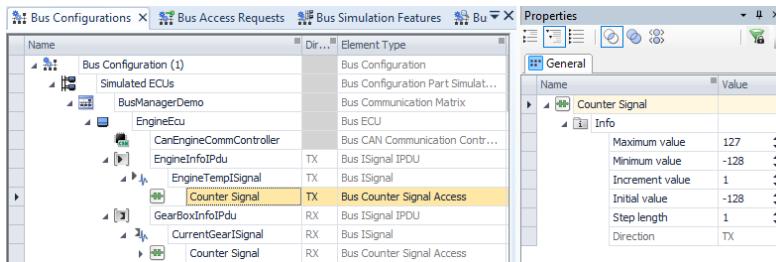
- Physical base data type: Int8 ... Int64
- Physical base data type: UInt8 ... UInt64

However, you cannot add the Counter Signal feature to array signals.

### Effects of adding the Counter Signal feature

Adding the Counter Signal feature to an ISignal has the following effects:

- Counter settings are available for the ISignal that let you configure the counter. You can access the counter settings in two ways:
  - In the Properties Browser when you select the Counter Signal node in the Bus Configurations table.



- In the Bus Simulation Features table.

Signal Features								
Name	Element Type	Dir...	Counter Signal	Minimum Value	Maximum Value	Increment Value	Initial Value	Step Length
EngineTempISignal	Bus Signal	TX	Enabled	-128	127	1	-128	1
CurrentGearISignal	Bus ISignal	RX	Enabled	0	255	1	0	1

You cannot change the counter settings during run time. Therefore, no function ports are available for these settings.

- If you add the Counter Signal feature to an RX ISignal, a Counter State function port is available for the ISignal. The function port provides the counter state, i.e., if a received counter value is the expected value according to the counter settings that are specified for the RX ISignal.

EngineInfoIPdu	TX	Bus ISignal IPDU
EngineTempISignal	TX	Bus ISignal
Counter Signal	TX	Bus Counter Signal Access
GearBoxInfoIPdu	RX	Bus ISignal IPDU
CurrentGearISignal	RX	Bus ISignal
Counter Signal	RX	Bus Counter Signal Access
CurrentGearISignal Counter State		

### Overview of the configurable counter settings

When you add the Counter Signal feature to an ISignal, you can configure the following counter settings:

Counter Setting	Purpose	Value Range
Minimum value	Lets you specify the minimum counter value.	<ul style="list-style-type: none"> <li>▪ Depends on the physical and coded base data types, and the length of the ISignal</li> <li>▪ Maximum range: Int32<sub>min</sub> ... Int32<sub>max</sub></li> <li>▪ Default value: Minimum valid value according to the conditions above</li> </ul>
Maximum value	Lets you specify the maximum counter value.	<ul style="list-style-type: none"> <li>▪ Depends on the physical and coded base data types, and the length of the ISignal</li> <li>▪ Maximum range: Int32<sub>min</sub> ... Int32<sub>max</sub></li> <li>▪ Default value: Maximum valid value according to the conditions above</li> </ul>

Counter Setting	Purpose	Value Range
Increment value	Lets you specify the step size for incrementing the counter value.	<ul style="list-style-type: none"> <li>▪ -(&lt;maximum value&gt; - &lt;minimum value&gt;) ... +(&lt;maximum value&gt; - &lt;minimum value&gt;)</li> <li>▪ Default value: 1</li> </ul> <p><b>Note</b> If the increment value is 0, the counter is disabled.</p>
Initial value	Lets you specify the initial value of the counter.	<ul style="list-style-type: none"> <li>▪ &lt;minimum value&gt; ... &lt;maximum value&gt;</li> <li>▪ Default value: &lt;minimum value&gt;</li> </ul>
Step length	Lets you specify the frequency for incrementing the counter value, i.e., if the counter value is incremented with each transmission/reception of the related PDU, with every other transmission/reception etc.	<ul style="list-style-type: none"> <li>▪ 1 ... Int32<sub>max</sub></li> <li>▪ Default value: 1</li> </ul>

The following table provides examples of counter values according to the specified counter settings.

Minimum Value	Maximum Value	Increment Value	Initial Value	Step Length	Counter Values
0	15	2	0	1	0, 2, 4, 6, 8, 10 ...
0	15	2	0	2	0, 0, 2, 2, 4, 4, 6, 6 ...
-8	15	2	-6	1	-6, -4, -2, 0, 2, 4 ...
-8	15	-4	15	1	15, 11, 7, 3, -1, -5 ...

## Counter behavior

**Basic counter behavior** Counter values are calculated according to the counter settings that are specified for the counter signal in the bus configuration. Depending on the direction of the counter signal (TX  or RX ), this results in the following counter behavior:

- In case of a TX counter signal, the counter value is calculated each time the transmission of the related TX PDU is triggered. The calculated counter value is then included in the PDU and transmitted on the bus.  
The timing for calculating TX counter signal values depends on the related Bus Configuration task. Refer to [Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager](#) on page 94.
- In case of an RX counter signal, the counter value is calculated each time the related RX PDU is received on the bus. The calculated counter value is compared with the counter value that is actually received with the PDU. The Counter State function port indicates if the received counter value was an expected value. Refer to [Counter states](#) on page 144.

Some bus configuration features can influence the calculation of the counter values. Refer to [Effects of bus configuration features on counter values](#) on page 145.

**Counter behavior when exceeding the counter limits** If the counter value exceeds the counter maximum or minimum value, the counter continues counting as shown in the following examples:

- Counter maximum value is exceeded:

Minimum Value	Maximum Value	Increment Value	Initial Value	Step Length	Counter Values
0	10	3	0	1	0, 3, 6, 9, 1, 4, 7, 10, 2, 5 ...
-5	10	3	-5	1	-5, -2, 1, 4, 7, 10, -3, 0, 3, 6 ...

- Counter minimum value is exceeded:

Minimum Value	Maximum Value	Increment Value	Initial Value	Step Length	Counter Values
0	10	-3	10	1	10, 7, 4, 1, 9, 6, 3, 0, 8, 5 ...
-4	10	-3	10	1	10, 7, 4, 1, -2, 10, 7, 4, 1, -3, 0 ...

## Counter states

RX counter signals provide a Counter State function port. The function port value indicates the counter states as follows:

Function Port Value	Counter State
0	Ok: The counter signal is received with the expected counter value.
4	Initial state: The counter signal is not received yet.
7	Error: The counter signal is only partially received, e.g., because of a reduced PDU length. Therefore, the counter signal is not decoded at all.
64	Wrong sequence: The counter signal is received with an unexpected counter value.

By default, the initial function port value is 4: Initial state. Via the function port's Initial value property, you can change the default initial value. To access the counter state at run time, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

**Calculation of the expected counter value** The expected counter value is calculated according to the counter settings that are specified for the RX counter signal in the bus configuration. However, the basis for calculating the expected counter value is the last received counter value, regardless of whether this counter value was an expected or unexpected value.

This synchronizes the RX counter with the related TX counter and ensures that one received unexpected counter value does not result in permanent counter errors.

#### Signal conversion of counter signal values

Like all signals that are transmitted and received by the Bus Manager, counter signal values are converted in the following ways:

- For transmission, the physical signal values are converted into coded signal values.
- For reception, the coded signal values are converted into physical signal values.

The signal values are converted according to computation methods that are specified in the communication matrix. For more information, refer to [Signal Conversion by the Bus Manager](#) on page 54.

Depending on the computation methods, rounding effects might occur. The rounding effects can result in different physical signal values on the sender and receiver side. However, these effects have no influence on the counter state that is provided by the Counter State function port because the function port evaluates the coded counter signal values.

#### Effects of bus configuration features on counter values

The following bus configuration features can affect the calculation of counter values.

**PDU Enable feature** The calculation of counter values stops if a counter signal is included in a TX PDU and the transmission of the PDU is disabled via the PDU Enable feature. When the transmission of the PDU is enabled again, the calculation of counter values continues, starting on the basis of the last transmitted counter value. This ensures that no counter errors occur due to the PDU Enable feature.

**Communication Controller Enable feature and LIN Schedule Table feature** The calculation of counter values stops if a counter signal is included in a PDU that is not transmitted or received via a communication cluster because of at least one of the following reasons:

- The related communication controller is disabled via the Communication Controller Enable feature.
- The LIN communication of the cluster is disabled via the LIN Schedule Table feature.

#### Note

If the PDU is transmitted or received via multiple clusters, the calculation of counter signals stops only if the PDU cannot be transmitted or received via any of the related clusters.

When the PDU is transmitted or received via at least one of the related clusters again, the counter value is reset to the initial counter value.

<b>Restrictions for using the Counter Signal feature</b>	<p>If you use the Counter Signal feature, the following restrictions apply:</p> <ul style="list-style-type: none"><li>▪ You cannot use the Counter Signal feature in parallel with the following bus configuration features:<ul style="list-style-type: none"><li>▪ ISignal Value feature (applies only to TX ISignals and TX PDUs)</li><li>▪ PDU Raw Data feature (applies only to TX ISignals and TX PDUs)</li><li>▪ Frame Access feature</li></ul></li><li>▪ If you add the Counter Signal feature and at least one of the other features to an ISignal or to a PDU in which the ISignal is included, conflicts occur.</li></ul>
<p><b>Tip</b></p> <p>If you add the Counter Signal, ISignal Value, and PDU Raw Data feature to an RX ISignal and to the RX PDU in which the ISignal is included, the values provided by each feature are updated at run time and can be accessed in parallel.</p>	

▪ If a counter signal is included in a TX PDU that is transmitted via a CAN communication cluster and a LIN communication cluster, the counter values are calculated only when the transmission of the PDU is triggered via the CAN cluster. The counter values are not calculated when the transmission of the PDU is triggered via the LIN cluster.

Related topics	References
	<p><a href="#">Function Outport Properties (Bus Configuration) (ConfigurationDesk Function Block Properties </a></p>

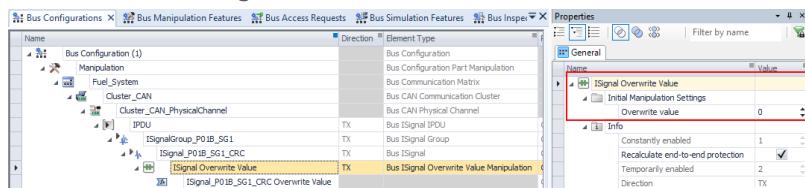
## Overwriting ISignal Values

<b>Introduction</b>	You can overwrite the signal values of ISignals that are assigned to the Manipulation part of a bus configuration with user-defined signal values.
<b>Conditions for overwriting ISignal values</b>	To overwrite the value of an ISignal with a user-defined value, you must add the ISignal Overwrite Value feature to the ISignal. You can add the feature to any ISignal that is assigned to the Manipulation part.

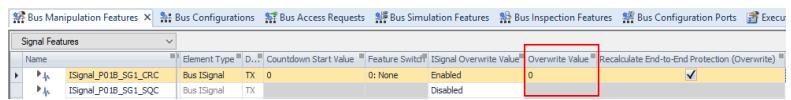
## Effects of adding the ISignal Overwrite Value feature

Adding the ISignal Overwrite Value feature to an ISignal has the following effects:

- If not already available, a feature switch is added to the ISignal. For the ISignal Overwrite Value feature, the feature switch provides the following values:
  - 1: ISignal Overwrite Value (permanently)
  - 2: ISignal Overwrite Value (temporarily)
- The feature switch lets you select the active bus manipulation feature and specify whether it manipulates the ISignal permanently or temporarily. For more information, refer to [Basics on Bus Manipulation Features](#) on page 131.
- An Overwrite value property is available that lets you specify the value that overwrites the ISignal value. You can access the property in following ways:
  - In the Properties Browser when you select the ISignal Overwrite Value node in the Bus Configurations table.



- In the Bus Manipulation Features table.



- If the ISignal is included in an end-to-end-protected ISignal group and/or involved in secure onboard communication, a Recalculate end-to-end protection and/or Recalculate SecOC information checkbox is available. You can access the checkboxes in the following ways:
  - In the Properties Browser when you select the ISignal Overwrite Value node in the Bus Configurations table.
  - In the Recalculate End-to-End Protection (Overwrite) and Recalculate SecOC Information (Overwrite) columns of the Bus Manipulation Features table, respectively.

If the related checkbox is selected, the end-to-end protection information of the affected ISignal group or the authentication information of the affected secured IPDU are recalculated before transmission. For more information, refer to [Recalculating End-to-End Protection and Authentication Information](#) on page 135.

- An Overwrite Value function port is available. The following illustration is an example of the function port as displayed in the Bus Configurations table:

	ISignal_P01B_SG1_CRC	TX	Bus ISignal
	Feature Switch	TX	Bus Feature Switch
	ISignal Overwrite Value	TX	Bus ISignal Overwrite Value Manipulation
	ISignal_P01B_SG1_CRC Overwrite Value		

---

<b>Specifying overwrite values</b>	<p>You can specify an overwrite value via the Overwrite value property. The specified value applies to the Initial value and Initial substitute value properties of the Overwrite Value function port automatically.</p> <p>The value range of the properties is determined by the physical base data type of the ISignal. If required, the property values are automatically saturated when the physical base data type changes. This can happen in the following cases:</p> <ul style="list-style-type: none"><li>▪ You modify the communication matrix. Refer to <a href="#">Basics on Modifying Communication Matrices</a> on page 267.</li><li>▪ You replace the assigned communication matrix. Refer to <a href="#">Replacing Assigned Communication Matrices</a> on page 291.</li></ul> <p>However, the maximum valid overwrite value also depends on the ISignal length. If the specified overwrite value exceeds the maximum valid value, it is automatically saturated at run time.</p>
<b>Run-time behavior of specified overwrite values</b>	<p>The specified value overwrites the original ISignal value at run time only if the ISignal Overwrite Value feature is enabled. The exact behavior depends on whether the feature is permanently or temporarily enabled. Refer to <a href="#">Basics on Bus Manipulation Features</a> on page 131.</p> <p>By default, you can change the specified overwrite value via experiment software at run time. If you want to change the value via a behavior model, you must enable model access for the Overwrite Value function port. Refer to <a href="#">Configuring Function Ports for Bus Configuration Features</a> on page 120.</p>
<b>Specifying saturation values for overwrite values</b>	<p>By default, overwrite values are limited by the ranges of the ISignal's physical base data type and automatically saturated at run time if required. Via the Overwrite Value function port, you can specify user-defined saturation values to limit the minimum and maximum overwrite values. If you want to do this, you must first set the bus configuration's Saturation usage property to User min/max value. Refer to <a href="#">Specifying saturation values for function imports</a> on page 123.</p> <p>If required, the saturation values are automatically saturated when the physical base data type changes. This can happen in the following cases:</p> <ul style="list-style-type: none"><li>▪ You modify in the communication matrix. Refer to <a href="#">Basics on Modifying Communication Matrices</a> on page 267.</li><li>▪ You replace the assigned communication matrix. Refer to <a href="#">Replacing Assigned Communication Matrices</a> on page 291.</li></ul>

---

**Related topics****References**

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Adding Offset Values to ISignal Values

### Introduction

You can add offset values to the values of ISignals that are assigned to the Manipulation part of a bus configuration.

### Conditions for adding offset values to ISignal values

To add an offset value to the value of an ISignal, you must add the ISignal Offset Value feature to the ISignal. You can add the feature to ISignals with the following base data types:

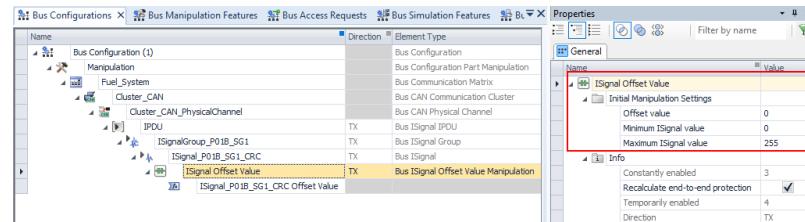
- Physical base data type: Int8 ... Int32
- Physical base data type: UInt8 ... UInt32

However, you cannot add the ISignal Offset Value feature to array signals.

### Effects of adding the ISignal Offset Value feature

Adding the ISignal Offset Value feature to an ISignal has the following effects:

- If not already available, a feature switch is added to the ISignal. For the ISignal Offset Value feature, the feature switch provides the following values:
  - 3: ISignal Offset Value (permanently)
  - 4: ISignal Offset Value (temporarily)
- The feature switch lets you select the active bus manipulation feature and specify whether it manipulates the ISignal permanently or temporarily. For more information, refer to [Basics on Bus Manipulation Features](#) on page 131.
- An Offset value, Minimum ISignal value, and Maximum ISignal value property are available that let you specify the offset value and the range of the ISignal. You can access the properties in following ways:
  - In the Properties Browser when you select the ISignal Offset Value node in the Bus Configurations table.



- In the Bus Manipulation Features table.

Name	Element Type	Countdown Start	Feature Switch	ISignal Offset Value	Offset Value	Minimum ISignal Value	Maximum ISignal Value	Recalculate End-to-End Protection (Offset)
ISignal_P01B_SG1_CRC	Bus ISignal	0	0: None	Enabled	0	0	255	<input checked="" type="checkbox"/>
ISignal_P01B_SG1_SOC	Bus ISignal			Disabled				

- If the ISignal is included in an end-to-end-protected ISignal group and/or involved in secure onboard communication, a Recalculate end-to-end protection and/or Recalculate SecOC information checkbox is available. You can access the checkboxes in the following ways:
  - In the Properties Browser when you select the ISignal Offset Value node in the Bus Configurations table.

- In the Recalculate End-to-End Protection (Offset) and Recalculate SecOC Information (Offset) columns of the Bus Manipulation Features table, respectively.

If the related checkbox is selected, the end-to-end protection information of the affected ISignal group or the authentication information of the affected IPDU are recalculated before transmission. For more information, refer to [Recalculating End-to-End Protection and Authentication Information](#) on page 135.

- The following function ports are available:
  - Offset Value function port
  - Minimum ISignal Value function port
  - Maximum ISignal Value function port

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

	ISignal_P01B_SG1_CRC	TX	Bus ISignal
	Feature Switch	TX	Bus Feature Switch
	ISignal Offset Value	TX	Bus ISignal Offset Value Manipulation
	ISignal_P01B_SG1_CRC Offset Value		
	ISignal_P01B_SG1_CRC Minimum ISignal Value		
	ISignal_P01B_SG1_CRC Maximum ISignal Value		

### Working with offset values

To work with offset values, you must specify the following values:

- The offset value
- The minimum value of the ISignal
- The maximum value of the ISignal

At run time, the specified offset value is added to the value of the ISignal. If the resulting ISignal value exceeds the specified minimum or maximum value, the ISignal value overruns, as shown in the following examples:

ISignal Value	Minimum ISignal Value	Maximum ISignal Value	Offset Value	Resulting ISignal Value
4	0	6	4	1
4	-6	6	4	-5
4	0	6	-5	6
4	3	6	-5	3

### Specifying offset, minimum, and maximum values

You can specify the offset value, minimum ISignal value, and maximum ISignal value via the Offset value, Minimum ISignal value, and Maximum ISignal value property, respectively. The specified value applies to the Initial value and Initial substitute value properties of the related function port automatically.

**Value range of the properties** The value range of the properties (including the value range of the function port properties) is determined by the physical base data type of the ISignal.

If the physical base data type is an unsigned integer type, the value range of the Offset value property is derived from the related integer type. For example, the

physical base data type is UInt8. In this case, the value range of the Offset value property is Int8<sub>min</sub> ... Int8<sub>max</sub>. The same applies to the Initial value and Initial substitute value properties of the Offset Value function port.

If required, the property values are automatically saturated when the physical base data type changes. This can happen in the following cases:

- You modify the communication matrix. Refer to [Basics on Modifying Communication Matrices](#) on page 267.
- You replace the assigned communication matrix. Refer to [Replacing Assigned Communication Matrices](#) on page 291.

However, the maximum valid value of the properties also depends on the ISignal length. If a specified property value exceeds the maximum valid value, the value is automatically saturated at run time.

#### **Run-time behavior of the specified values**

The specified values apply to the ISignal at run time only if the ISignal Offset Value feature is enabled. The exact behavior depends on whether the feature is permanently or temporarily enabled. Refer to [Basics on Bus Manipulation Features](#) on page 131.

By default, you can change the specified values via experiment software at run time. If you want to change the values via a behavior model, you must enable model access for the related function ports. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

#### **Specifying saturation values for the offset, minimum, and maximum values**

By default, the offset value and the minimum and maximum ISignal values are limited by the ranges of the ISignal's physical base data type. At run time, the offset, minimum, and maximum values are automatically saturated if required. Via the Offset Value, Minimum ISignal Value, and Maximum ISignal Value function ports, you can specify user-defined saturation values to limit the minimum and maximum values of the function ports. If you want to do this, you must first set the bus configuration's Saturation usage property to User min/max value. Refer to [Specifying saturation values for function imports](#) on page 123.

If required, the saturation values are automatically saturated when the physical base data type changes. This can happen in the following cases:

- You modify the communication matrix. Refer to [Basics on Modifying Communication Matrices](#) on page 267.
- You replace the assigned communication matrix. Refer to [Replacing Assigned Communication Matrices](#) on page 291.

#### **Related topics**

#### **References**

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

# Bus Configuration Features Available for ISignal Groups

## Observing the Status of Received End-to-End-Protected ISignal Groups

### Introduction

During run time, you can observe the status of end-to-end-protected RX ISignal groups that are assigned to the Simulated ECUs or Inspection part of a bus configuration.

### Conditions for observing the status of received end-to-end-protected ISignal groups

To observe the status of a received end-to-end-protected ISignal group at run time, you must add the ISignal Group End-to-End Protection Status feature to the ISignal group. You can add the feature to any RX ISignal group that is end-to-end-protected. For information on end-to-end protection, refer to [Aspects of Supported AUTOSAR Features](#) on page 39.

### Effects of adding the ISignal Group End-to-End Protection Status feature

When you add the ISignal Group End-to-End Protection Status feature to an ISignal group, the following function ports are available for the ISignal group:

- Calculated CRC function port
- State function port

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

	ISignalGroup_P01B_SG1	RX	Bus ISignal Group
	ISignal Group End-to-End Protection Status	RX	Bus ISignal Group End To End Protection Status Access
	ISignalGroup_P01B_SG1 Calculated CRC		
	ISignalGroup_P01B_SG1 State		

### Observing the status of end-to-end-protected ISignal groups via function ports

The function ports let you observe the status of a received end-to-end-protected ISignal group. When you enable model access or test automation support for a function port, you can use the data that is provided by the function port in a mapped behavior model or in experiment software, respectively. For more information, refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

The Bus Manager processes received ISignal groups even if the status indicates an error. If you want to reject an ISignal group in this case, you must model the rejection in a behavior model, e.g., by using data that is provided by the State function port. Refer to [Data provided by the State function port](#) on page 153.

### Data provided by the Calculated CRC function port

The Calculated CRC function port provides the checksum value that is calculated for the received ISignal group. The value is calculated according to the AUTOSAR end-to-end protection profile that is specified for the ISignal group.

By default, the initial function port value is 0 until the ISignal group is received for the first time. Via the function port's Initial Value property, you can specify a user-defined initial value. Refer to [Specifying initial values](#) on page 120.

The valid range of the function port depends on the related end-to-end protection profile, as shown in the following table.

End-to-End Protection Profile of ISignal Group	Value Range of Calculated CRC Function Port
Profile 01	UInt8 <sub>min</sub> ... UInt8 <sub>max</sub>
Profile 02	UInt8 <sub>min</sub> ... UInt8 <sub>max</sub>
Profile 05	UInt16 <sub>min</sub> ... UInt16 <sub>max</sub>
Profile 11	UInt8 <sub>min</sub> ... UInt8 <sub>max</sub>
Profile 22	UInt8 <sub>min</sub> ... UInt8 <sub>max</sub>

#### Data provided by the State function port

The State function port provides the state of the end-to-end protection, i.e., whether the received end-to-end protection information indicates an error. For each end-to-end protection profile, the possible states are specified in the AUTOSAR end-to-end communication protection library (E2E library).

The following table provides an overview of the possible function port values and the resulting states.

Function Port Value	Resulting State	Available for	Description
0: OK	OK	<ul style="list-style-type: none"> <li>▪ Profile 01</li> <li>▪ Profile 02</li> <li>▪ Profile 05</li> <li>▪ Profile 11</li> <li>▪ Profile 22</li> </ul>	The received end-to-end protection information is correct.
1: No new data	Error	<ul style="list-style-type: none"> <li>▪ Profile 01</li> <li>▪ Profile 02</li> <li>▪ Profile 05</li> <li>▪ Profile 11</li> <li>▪ Profile 22</li> </ul>	The end-to-end protection information has not changed between two consecutive receptions of the ISignal group.
2: Wrong checksum	Error	<ul style="list-style-type: none"> <li>▪ Profile 01</li> <li>▪ Profile 02</li> </ul>	The received checksum value is incorrect. Only for profile 01 with E2E_P01DataIdMode parameter set to E2E_P01_DATAID_NIBBLE: Alternatively, the low nibble of the high byte of the Data ID is incorrect.
3: Synchronization in progress	Not valid	<ul style="list-style-type: none"> <li>▪ Profile 01</li> <li>▪ Profile 02</li> </ul>	The received end-to-end protection information is correct, but a previously received counter value was incorrect. The continuity check of the current counter value has not finished yet.
4: Initial state	Initial	<ul style="list-style-type: none"> <li>▪ Profile 01</li> <li>▪ Profile 02</li> </ul>	The received checksum value is correct, but the received counter value cannot be verified because it is the first received counter value.
7: Error	Error	<ul style="list-style-type: none"> <li>▪ Profile 01</li> <li>▪ Profile 02</li> <li>▪ Profile 05</li> <li>▪ Profile 11</li> <li>▪ Profile 22</li> </ul>	Bus Manager-specific and for all profiles: At least one ISignal of the ISignal group is only partially received because the payload length of the PDU is too short to include the ISignal completely in the payload.

Function Port Value	Resulting State	Available for	Description
			<ul style="list-style-type: none"> <li>▪ Only for profile 05, 11, and 22: According to the E2E library, the received end-to-end protection information is incorrect, e.g., because of an incorrect checksum value. However, the received counter value is correct.</li> </ul>
8: Repeated data	Error	<ul style="list-style-type: none"> <li>▪ Profile 01</li> <li>▪ Profile 02</li> <li>▪ Profile 05</li> <li>▪ Profile 11</li> <li>▪ Profile 22</li> </ul>	An identical counter value was received for two consecutive receptions of the ISignal group.
32: OK, but some data lost	OK	<ul style="list-style-type: none"> <li>▪ Profile 01</li> <li>▪ Profile 02</li> <li>▪ Profile 05</li> <li>▪ Profile 11</li> <li>▪ Profile 22</li> </ul>	The received end-to-end protection information is correct but some data might be lost. However, the received counter value is in the range of the specified counter tolerance.
64: Wrong sequence	Error	<ul style="list-style-type: none"> <li>▪ Profile 01</li> <li>▪ Profile 02</li> <li>▪ Profile 05</li> <li>▪ Profile 11</li> <li>▪ Profile 22</li> </ul>	The received end-to-end protection information is incorrect because the received counter value exceeds the specified maximum counter tolerance.

The default initial value of the function port depends on the end-to-end protection profile. For the profiles 01 and 02, the initial value is 4: Initial state. For all other profiles, the initial value is 0: OK. The default initial value applies until the ISignal group is received for the first time. Via the function port's Initial Value property, you can change the default initial value. Refer to [Specifying initial values](#) on page 120.

---

#### Restrictions for using the ISignal Group End-to-End Protection Status feature

You cannot use the ISignal Group End-to-End Protection Status and the Frame Access feature in parallel. If you add the ISignal Group End-to-End Protection Status feature to an ISignal group that is included in a PDU for which the Frame Access feature is added, conflicts occur. This also applies if the PDU is included in another PDU and the Frame Access feature is added to this PDU.

---

#### Related topics

#### References

[Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

# Bus Configuration Features Available for PDUs

Where to go from here	Information in this section																						
	<table> <tr> <td>Enabling and Disabling the Transmission of PDUs.....</td> <td>155</td> </tr> <tr> <td>Accessing the Payload of PDUs in Raw Data Format.....</td> <td>157</td> </tr> <tr> <td>Controlling the Cyclic Timing of CAN PDUs.....</td> <td>159</td> </tr> <tr> <td>Specifying User-Defined Triggers for Transmitting PDUs.....</td> <td>162</td> </tr> <tr> <td>Accessing the Payload Length of CAN PDUs.....</td> <td>165</td> </tr> <tr> <td>Observing the Status of Received PDUs.....</td> <td>167</td> </tr> <tr> <td>Applying User Code to PDUs.....</td> <td>170</td> </tr> <tr> <td>Verifying the Authentication Information of Received Secured IPDUs.....</td> <td>175</td> </tr> <tr> <td>Invalidating the Authenticator of Secured IPDUs.....</td> <td>177</td> </tr> <tr> <td>Overwriting the Freshness Value of Secured IPDUs.....</td> <td>180</td> </tr> <tr> <td>Triggering the Execution of Functions in a Behavior Model via RX Interrupts.....</td> <td>183</td> </tr> </table>	Enabling and Disabling the Transmission of PDUs.....	155	Accessing the Payload of PDUs in Raw Data Format.....	157	Controlling the Cyclic Timing of CAN PDUs.....	159	Specifying User-Defined Triggers for Transmitting PDUs.....	162	Accessing the Payload Length of CAN PDUs.....	165	Observing the Status of Received PDUs.....	167	Applying User Code to PDUs.....	170	Verifying the Authentication Information of Received Secured IPDUs.....	175	Invalidating the Authenticator of Secured IPDUs.....	177	Overwriting the Freshness Value of Secured IPDUs.....	180	Triggering the Execution of Functions in a Behavior Model via RX Interrupts.....	183
Enabling and Disabling the Transmission of PDUs.....	155																						
Accessing the Payload of PDUs in Raw Data Format.....	157																						
Controlling the Cyclic Timing of CAN PDUs.....	159																						
Specifying User-Defined Triggers for Transmitting PDUs.....	162																						
Accessing the Payload Length of CAN PDUs.....	165																						
Observing the Status of Received PDUs.....	167																						
Applying User Code to PDUs.....	170																						
Verifying the Authentication Information of Received Secured IPDUs.....	175																						
Invalidating the Authenticator of Secured IPDUs.....	177																						
Overwriting the Freshness Value of Secured IPDUs.....	180																						
Triggering the Execution of Functions in a Behavior Model via RX Interrupts.....	183																						

## Enabling and Disabling the Transmission of PDUs

<b>Introduction</b>	You can enable and disable the transmission of <a href="#">basic PDUs</a> that are assigned to the Simulated ECUs part of a bus configuration.
<b>Conditions for enabling and disabling the transmission of PDUs</b>	To enable or disable the transmission of a PDU, you must add the PDU Enable feature to a selected TX PDU.  In general, you can add the feature to all TX <a href="#">basic PDUs</a> that are not configured as the static part of a <a href="#">multiplexed IPDU</a> . However, you can add the feature to a TX general-purpose PDU only if the PDU does not contain a global time domain in the bus configuration.
<b>Effects of adding the PDU Enable feature</b>	When you add the PDU Enable feature to a TX basic PDU, a PDU Enable function port is available for the PDU that lets you specify the enable state. The following illustration is an example of the function port as displayed in the Bus Configurations table:

	RawSensorsIPdu	TX	Bus ISignal IPDU
	PDU Enable	TX	Bus PDU Enable Access
	RawSensorsIPdu Enable		

**Specifying the enable state**

The PDU Enable function port lets you specify the enable state of the PDU as follows:

- False (0): The transmission of the PDU is disabled.
- True (1): The transmission of the PDU is enabled. This is the default state.

Via the Initial value property of the function port, you can change the default enable state. To change the state at run time, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

**Effects of disabling the transmission of PDUs**

When you disable the transmission of a PDU, you disable it for a specific transmitting ECU. This lets you test, for example, if another ECU in the communication cluster transmits the PDU instead.

Disabling the transmission of a PDU has the following effects on the active bus communication:

- If you disable the transmission of a PDU while the PDU is being transmitted, the transmission is not interrupted. The transmission of the PDU is disabled after the PDU is completely transmitted.
- If the transmission of a PDU is triggered while the transmission of the PDU is disabled, the trigger is discarded and lost, i.e., the trigger is not processed when the transmission of the PDU is enabled again.

**Restrictions for using the PDU Enable feature**

If you use the PDU Enable feature, the following restrictions apply:

- You can enable and disable the transmission of a PDU only for the entire transmitting ECU and not selectively for specific frames, channels, or clusters. For example, if you disable a PDU that is included in two frames, none of the frames transmits the PDU.
- You cannot use the PDU Enable and the Frame Access feature in parallel. If you add both features to a PDU, conflicts occur. This also applies if the PDU is included in another PDU and the Frame Access feature is added to this PDU.

**Related topics****References**

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Accessing the Payload of PDUs in Raw Data Format

### Introduction

You can access the payload of PDUs that are assigned to the Simulated ECUs or Inspection part of a bus configuration in raw data format.

### Conditions for accessing the payload of PDUs in raw data format

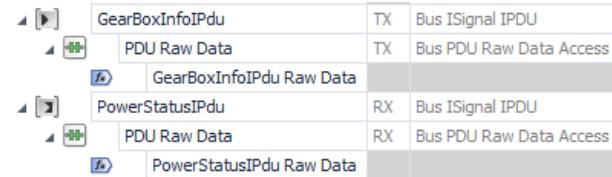
To access the payload of a PDU in raw data format, you must add the PDU Raw Data feature to the selected PDU.

In general, you can add the feature to any RX and TX PDU whose payload length is > 0 bytes. However, you can add the feature to a TX general-purpose PDU only if the PDU does not contain a global time domain in the bus configuration.

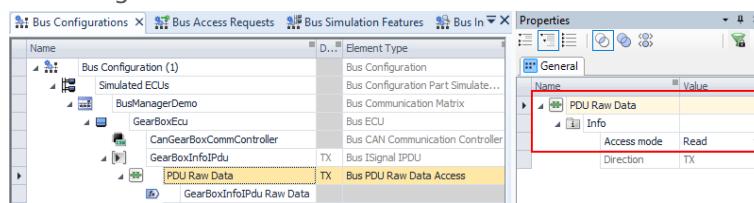
### Effects of adding the PDU Raw Data feature

Adding the PDU Raw Data feature to a PDU has the following effects:

- A PDU Raw Data function port is available for the PDU.



- If you add the PDU Raw Data feature to a TX basic PDU, an Access mode property is available that lets you specify to read or write the payload in raw data format. You can access the property in two ways:
- In the Properties Browser when you select the PDU Raw Data node in the Bus Configurations table.



- In the Bus Simulation Features table.

Name	Element Type	Direction	Bus Config...	Communicat...	ECU	PDU Raw Data	Access Mode
GearBoxInfoIPdu	Bus ISignal IPDU	TX	Bus Config...	Bus Manage...	GearBo...	Enabled	Read
PDU Raw Data	Bus PDU Raw Data Access		Bus Config...	Bus Manage...	GearBo...	Enabled	Read

### Accessing the payload via the PDU Raw Data function port

The PDU Raw Data function port lets you access the payload of a PDU in raw data format.

**Read or write raw data** By default, the function port provides read access to the payload raw data. This applies regardless of whether the PDU is an RX or TX PDU. You can change the access mode of the function port to write raw data

to the payload only if you add the PDU Raw Data feature to a TX basic PDU. Refer to [Writing raw data to the payload of a TX basic PDU](#) on page 158.

**Reading raw data** Reading the payload raw data lets you observe the payload data that is actually available on the bus, i.e.:

- RX PDU: The raw data that is received on the bus.
- TX PDU: The raw data that is transmitted on the bus.

If you only read the payload raw data, you can access signal-based payload data in parallel, e.g., you can add the ISignal Value and Counter Signal features to ISignals of the PDU to transmit or receive the values of ISignals and counter signals.

**Accessing the payload raw data at run time** For the PDU Raw Data function port, test automation support is enabled by default. This lets you access the payload raw data via experiment software at run time. If you want to access the raw data via a behavior model, you must enable model access for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

#### Writing raw data to the payload of a TX basic PDU

If you add the PDU Raw Data feature to a TX basic PDU, you can write raw data to the payload of the PDU. For this purpose, you can change the access mode of the PDU Raw Data function port via the Access mode property as follows:

- Read: The payload of the TX basic PDU is read in raw data format. This is the default mode.
- Write: Raw data is written to the payload of the TX basic PDU.

Setting the Access mode property to Write has the following effects:

- The port type of the PDU Raw Data function port changes: The [function outport](#) is converted to a [function import](#).

##### Note

If the function port is mapped to a model port, the mapping is lost.

- The payload of the TX basic PDU can be transmitted in raw data format only, i.e., you cannot transmit signal-based payload data, such as ISignal values, in parallel.

#### Value range and initial values of the PDU Raw Data function port

The value of the PDU Raw Data function port is a vector. The vector size is the length of the related PDU, e.g., for a PDU with a length of 2 bytes the vector size is 2. The vector size is automatically adjusted when the PDU length changes in the communication matrix. This can happen in the following cases:

- You modify the communication matrix. Refer to [Basics on Modifying Communication Matrices](#) on page 267.
- You replace the assigned communication matrix. Refer to [Replacing Assigned Communication Matrices](#) on page 291.

The value range of each vector element is 0 ... 255, the default value is 0. Via the function port's Initial value and Initial substitute value properties, you can

specify a user-defined initial raw data. Refer to [Specifying initial values](#) on page 120.

#### **Restrictions for using the PDU Raw Data feature**

- If you use the PDU Raw Data feature, the following restrictions apply:
    - You can access only the complete payload of a PDU in raw data format. You cannot specify specific bytes of the payload to be accessed in raw data format.
    - You cannot use the PDU Raw Data feature in parallel with the following bus configuration features:
      - Frame Access feature
      - For TX basic PDUs whose Access mode property is set to Write:
        - Counter Signal feature
        - ISignal Value feature
- If you add the PDU Raw Data feature and at least one of the other features to a PDU, its included ISignals, and/or to a PDU the PDU is included in, conflicts occur.

#### **Related topics**

#### **References**

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

[Function Output Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Controlling the Cyclic Timing of CAN PDUs

#### **Introduction**

You can control the cyclic timing of CAN TX PDUs that are assigned to the Simulated ECUs part of a bus configuration.

#### **Conditions for controlling the cyclic timing of CAN PDUs**

To control the cyclic timing of a CAN TX PDU, you must add the PDU Cyclic Timing Control feature to the PDU.

In general, you can add the feature to all CAN TX [basic PDUs](#) that are not configured as the static part of a [multiplexed IPDU](#). However, you can add the feature to a TX general-purpose PDU only if the PDU does not contain a global time domain in the bus configuration.

The PDU Cyclic Timing Control feature is independent of cyclic timings specified in the [communication matrix](#), i.e., you can add the feature to a CAN TX PDU regardless of whether the communication matrix specifies a cyclic timing for the PDU.

**Effects of adding the PDU Cyclic Timing Control feature**

When you add the PDU Cyclic Timing Control feature to a PDU, two function ports are available for the PDU:

- PDU Timing Control Period function port

This function port lets you specify a period (i.e., a cycle time) in seconds for the PDU.

- PDU Timing Control Offset function port

This function port lets you specify an offset (i.e., a delay time) in seconds for the PDU.

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

	RawSensorsIPdu	TX	Bus ISignal IPDU
	PDU Cyclic Timing Control	TX	Bus PDU Cyclic Timing Control Access
	RawSensorsIPdu Timing Control Period		
	RawSensorsIPdu Timing Control Offset		

**Controlling the cyclic timing of CAN PDUs via function ports**

When you add the PDU Cyclic Timing Control feature to a PDU, the default period and offset are derived from the communication matrix. These values are used as the initial values of the related function ports. If the communication matrix does not specify a period and/or offset, 0 is used as the initial value of the related function port. Via the Initial value property of the PDU Timing Control Period function port and PDU Timing Control Offset function port, you can specify a user-defined initial period and offset, respectively.

**Note**

If the period is 0, the PDU is not transmitted.

To change the period and offset at run time, you must enable model access or test automation support for the related function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

The actual timing for transmitting a PDU cyclically depends on the related Bus Configuration task. Additionally, specifications of the communication matrix might influence the timing (e.g., if the PDU is a contained IPDU and for the related container IPDU a container timeout value is specified). For more information on the Bus Configuration task, refer to [Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager](#) on page 94.

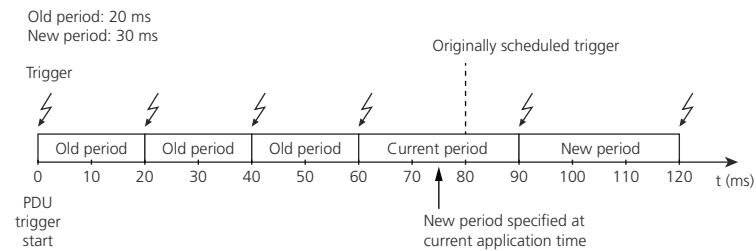
**Notes on working with an offset for a PDU**

Specifying an offset via the PDU Timing Control Offset function port lets you delay each first transmission of the related PDU after the following events:

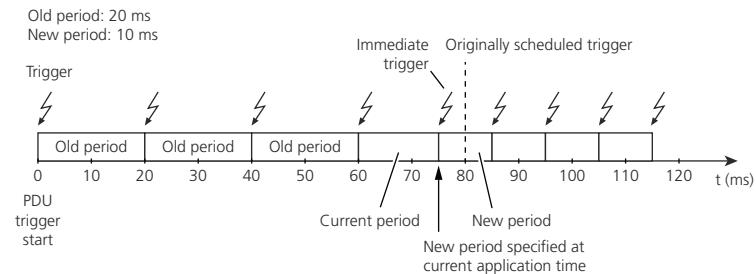
- The executable application is started or restarted.
- The related communication controller is enabled (e.g., via the Communication Controller Enable feature).
- You change the specified offset at run time.

### Modifying the specified period and/or offset at run time

**Modifying the period at run time** When you reduce or extend the period at run time, the active period is reduced or extended according to the changed period. The following illustration is an example of extending the period at run time:



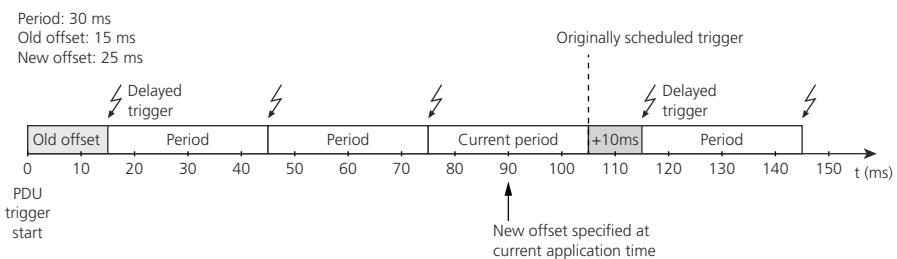
If you reduce the period at run time so much that the PDU would have had to be triggered in the past, the PDU is triggered immediately, as shown in the following example:



**Modifying the offset at run time** Extending or reducing the offset at run time directly affects the active period as well. The following illustrations are examples for the effects of a changed offset on the active period and the related PDU trigger:

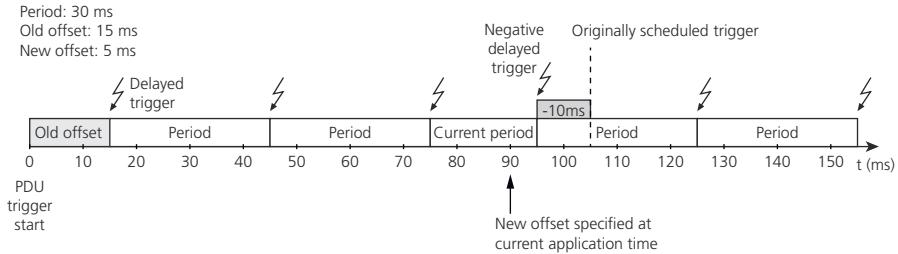
- Extended offset

The active period is extended by the time difference between the old and the new offset:

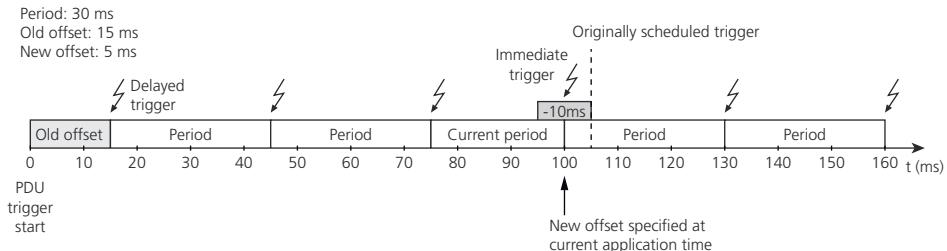


- Reduced offset

The active period is reduced by the time difference between the old and the new offset:



If the PDU would have had to be triggered in the past due to the reduced offset, the PDU is triggered immediately:



### Restrictions for using the PDU Cyclic Timing Control feature

You cannot use the PDU Cyclic Timing Control and the Frame Access feature in parallel. If you add both features to a PDU, conflicts occur. This also applies if the PDU is included in another PDU and the Frame Access feature is added to this PDU.

### Related topics

### References

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Specifying User-Defined Triggers for Transmitting PDUs

### Introduction

You can specify user-defined triggers to trigger the transmission of [basic PDUs](#) that are assigned to the Simulated ECUs part of a bus configuration. Via the user-defined triggers, you can trigger the transmission of PDUs continuously or asynchronously to a cyclic timing, for example.

### Conditions for specifying user-defined triggers for transmitting PDUs

To specify user-defined triggers for transmitting a PDU at run time, you must add the PDU Trigger feature to the PDU. In general, you can add the PDU Trigger feature to the following [basic PDUs](#):

- CAN TX PDUs that are not configured as the static part of a [multiplexed IPDU](#)
- LIN TX PDUs that are included in event-triggered or sporadic frames

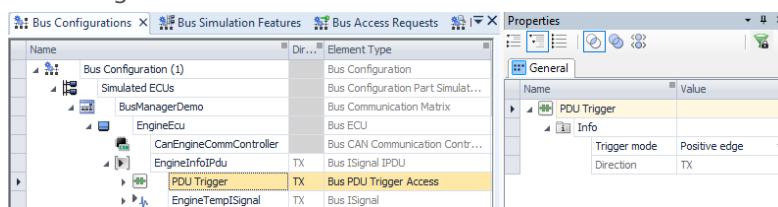
However, you can add the feature to a TX general-purpose PDU only if the PDU does not contain a global time domain in the bus configuration.

The PDU Trigger feature is independent of cyclic timings specified in the [communication matrix](#) or via the PDU Cyclic Timing Control feature. Therefore, you can add the PDU Trigger feature to a PDU regardless of whether cyclic timings are available for the PDU.

### Effects of adding the PDU Trigger feature

Adding the PDU Trigger feature to a PDU has the following effects:

- A Trigger mode property is available for the PDU that lets you specify the triggering mode for the PDU. You can access the property in two ways:
  - In the Properties Browser when you select the PDU Trigger node in the Bus Configurations table.



- In the Bus Simulation Features table.

PDU Features									
Name	Element Type	Di...	PDU R...	PDU Use...	User Co...	SecOC	PDU Trigger	Trigger Mode	Frame Ac...
EngineInfoIPdu	Bus ISignal IPDU	TX	Disabled	Disabled	Enabled		Enabled	Positive edge	Disabled
GearBoxInfoIPdu	Bus ISignal IPDU	RX	Disabled	Disabled					

You cannot change the triggering mode during run time. Therefore, no function port is available for the triggering mode.

- A Trigger function port is available for the PDU. Via the function port, you can trigger the transmission of the PDU at run time. The following illustration is an example of the function port as displayed in the Bus Configurations table:

EngineInfoIPdu	TX	Bus ISignal IPDU
PDU Trigger	TX	Bus PDU Trigger Access
EngineInfoIPdu Trigger		

### Specifying user-defined triggers

Specifying user-defined triggers for the transmission of a PDU comprises the following basic steps:

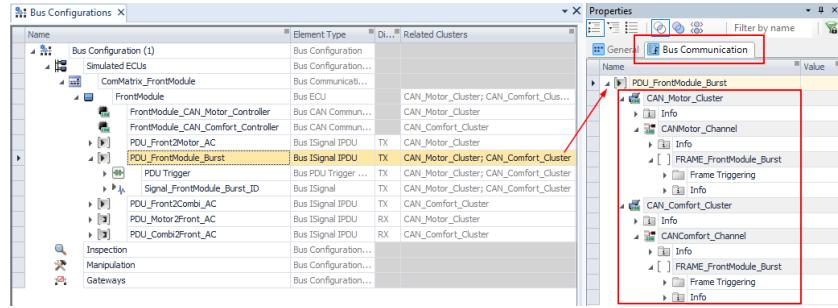
1. Add the PDU Trigger feature to the PDU.
2. Select a triggering mode via the Trigger mode property.
3. Configure the Trigger function port.

For more information, refer to [Triggering the Transmission of PDUs and Frames via User-Defined Triggers](#) on page 128.

### Effects of triggering the transmission of PDUs via the PDU Trigger feature

When you trigger the transmission of a PDU at run time, the PDU is transmitted via all involved frames and communication clusters.

To identify the involved frames and communication clusters, you can select the PDU in the Bus Configurations table or Bus Simulation Features table. If you do this, the Bus Communication page of the Properties Browser displays all the involved frames and clusters, as shown in the following example.



#### Tip

If you do not want to trigger the transmission of the PDU via a specific frame, you can do the following: Remove the PDU from the bus configuration. Then, drag the PDU from the Communication Matrices by Clusters view of the Buses Browser to assign it to the bus configuration in the context of a specific communication cluster. Refer to [Assigning Communication Matrix Elements to Bus Configurations](#) on page 75.

The timing for transmitting the PDU depends on the related Bus Configuration task and the specifications in the communication matrix. Refer to [Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager](#) on page 94.

### Restrictions for using the PDU Trigger feature

You cannot use the PDU Trigger and the Frame Access feature in parallel. If you add both features to a PDU, conflicts occur. This also applies if the PDU is included in another PDU and the Frame Access feature is added to this PDU.

### Related topics

#### Basics

Limitations for Bus Configuration Handling.....	328
Limitations for LIN Communication.....	325

#### References

<a href="#">Function Import Properties (Bus Configuration) (ConfigurationDesk Function Block Properties)</a>
--

## Accessing the Payload Length of CAN PDUs

### Introduction

You can access the payload length of CAN PDUs that are assigned to the Simulated ECUs part of a bus configuration at run time, e.g., to simulate dynamic length CAN PDUs.

### Conditions for accessing the payload length of CAN PDUs

To access the payload length of a CAN PDU at run time, you must add the PDU Length feature to the PDU. You can add the PDU Length feature to [basic PDUs](#) that fulfill all of the following conditions:

- The basic PDUs are included only in CAN frames.
- The basic PDUs are not configured in any of the following ways:
  - Authentic PDU, i.e., neither included in a secured IPDU nor protected by a cryptographic IPDU
  - Contained IPDU
  - Static part or dynamic part of a multiplexed IPDU
  - TX general-purpose PDU that contains a global time domain in the bus configuration

### Effects of adding the PDU Length feature

Depending on the direction of the PDU ([TX](#) or [RX](#)), a TX Length function port or RX Length function port is available that lets you access the payload length of the PDU at run time. The following illustration is an example of the function port as displayed in the Bus Configurations table:

	GeneralInfoIPdu	TX	Bus ISignal IPDU
	PDU Length	TX	Bus PDU Length Access
	GeneralInfoIPdu Length		
	CarLockControlIPdu	RX	Bus ISignal IPDU
	PDU Length	RX	Bus PDU Length Access
	CarLockControlIPdu Length		

### Accessing the payload length

The Length function port provides the payload length in bytes as follows:

- For TX PDUs, the function port specifies the payload length with which the PDU is transmitted on the bus.
- For RX PDUs, the function port provides the received payload length of the PDU, e.g., to a mapped behavior model.

By default, the initial function port value is the PDU length that is specified in the [communication matrix](#). Via the function port's Initial value property, you can specify a user-defined initial payload length in the range `0 .... <PDU length specified in the communication matrix>`.

The maximum valid function port value changes if the PDU length changes in the communication matrix. This can happen in the following cases:

- You modify the communication matrix. Refer to [Basics on Modifying Communication Matrices](#) on page 267.

- You replace the assigned communication matrix. Refer to [Replacing Assigned Communication Matrices](#) on page 291.

If required, you must adjust the initial payload length manually.

To access the payload length at run time, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

---

#### Run-time effects of changed payload lengths

Changed payload lengths have the following effects:

- If the payload length of a TX PDU changes at run time and the new length exceeds the maximum valid TX Length function port value, the length is automatically saturated.
- If a CAN FD PDU is transmitted (i.e., a TX PDU) and the specified payload length is invalid according to CAN FD, the length is automatically extended by padding bytes to the next higher valid CAN FD length. The value of each padding byte is 255.
- Depending on the direction of the PDU (i.e., TX or RX), a reduced payload length has the following effects on ISignals:
  - TX PDU: The Bus Manager includes only ISignals in the PDU that completely fit into the specified payload length. ISignals that do not completely fit into the specified payload length are not transmitted. This results in unused bits.
  - RX PDU: The Bus Manager decodes ISignals only if they are received completely. If an ISignal is only partially received because of a reduced payload length, the ISignal is not decoded at all. As a result, the raw data of the affected payload bits might change but the value of the ISignal remains unchanged (e.g., if you access the value via the ISignal Value feature).
- Unused bits of the payload are filled with the value of the unused bit pattern that is specified for the PDU. For example, bits might be unused in the following cases:
  - ISignals are not included in a TX PDU because they do not completely fit into the payload.
  - The payload length of a TX PDU is extended via the TX Length function port. However, the unused bit pattern does not apply to padding bytes that are added to the payload of TX PDUs if the specified payload length is invalid according to CAN FD.

You can specify the unused bit pattern. Refer to [Configurable Settings of PDUs](#) on page 281.

---

#### Specifying saturation values for TX Length function ports

For TX Length function ports, you can specify saturation values to limit the minimum and maximum payload length of TX PDUs. If you want to do this, you must first set the bus configuration's Saturation usage property to User min/max value. Refer to [Specifying saturation values for function imports](#) on page 123.

The valid saturation values of the TX Length function port might change if the PDU length changes in the communication matrix. This can happen in the following cases:

- You modify the communication matrix. Refer to [Basics on Modifying Communication Matrices](#) on page 267.
- You replace the assigned communication matrix. Refer to [Replacing Assigned Communication Matrices](#) on page 291.

If required, you must adjust the specified saturation values manually.

#### Related topics

#### References

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Observing the Status of Received PDUs

#### Introduction

During run time, you can observe the status of RX PDUs that are assigned to the Simulated ECUs or Inspection part of a bus configuration.

#### Conditions for observing the status of received PDUs

To observe the status of a received PDU [?](#) at run time, you must add the PDU RX Status feature to the PDU. You can add the feature to any RX PDU.

#### Effects of adding the PDU RX Status feature

When you add the PDU RX Status feature to a PDU, the following function ports are available for the PDU:

- PDU Counter function port
- PDU State function port
- PDU Time function port
- PDU Delta Time function port

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

	RawSensorsIPdu	RX	Bus ISignal IPDU
	PDU RX Status	RX	Bus PDU RX Status Inspection
	RawSensorsIPdu Counter		
	RawSensorsIPdu State		
	RawSensorsIPdu Time		
	RawSensorsIPdu Delta Time		

**Observing the status of received PDUs via function ports**

The function ports let you observe the status of a received PDU. When you enable model access or test automation support for a function port, you can use the data that is provided by the function port in a mapped behavior model or in experiment software, respectively. For more information on enabling model access and test automation support, refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

**Data provided by the PDU Counter function port** This function port provides the value of the counter that is implemented for the PDU when you add the PDU RX Status feature. Refer to [Counter values available for the PDU](#) on page 168.

**Data provided by the PDU State function port** This function port provides the state of the PDU, i.e., whether the PDU is received in the current sampling step:

- 0: The PDU is not received in the current sampling step.
- 1: The PDU is received in the current sampling step.

Via the function port's Initial value property, you can specify an initial state for the PDU:

- False (0): The PDU is not received. This is the default state.
- True (1): The PDU is received.

**Data provided by the PDU Time function port** This function port provides the time (in seconds) of the executable application when the PDU was received. By default, the initial function port value is 0 until the PDU is received for the first time. Via the function port's Initial value property, you can specify a user-defined initial value. The function port value is reset to the initial value each time the executable application starts, regardless of the setting of the Initial value usage property.

**Data provided by the PDU Delta Time function port** This function port provides the time difference (in seconds) between the current and the previous reception of the PDU. By default, the initial function port value is 0 until the PDU is received for the first time. Via the function port's Initial value property, you can specify a user-defined initial value. The function port value is reset to the initial value each time the executable application starts, regardless of the setting of the Initial value usage property.

Because there is no actual time difference when the PDU is received for the first time, the function port provides the time of the executable application in this case. With the second reception of the PDU, the function port provides the actual time difference.

For more information on initial function port values, refer to [Specifying initial values](#) on page 120.

---

**Counter values available for the PDU**

**Basics on the counter** When you add the PDU RX Status feature to a PDU, a counter is implemented for this PDU. The counter's data type is UInt32 and the

counter's initial value is 0. The counter value is incremented by 1 each time the PDU is received on the bus. If an overflow occurs, the counter value is reset to 0.

**Values provided by the PDU Counter function port** You can access the counter value via the PDU Counter function port. For the PDU Counter function port, you can specify an initial value via the function port's **Initial value** property.

**Note**

Do not confuse the counter's initial value with the function port's initial value.

The counter's initial value is always 0, regardless of the value specified for the function port's **Initial value** property. The value of the **Initial value** property applies only to the function port. It is used until the related **Com\_MainFunction** runnable function is executed for the first time. Then, the function port provides the counter value (i.e., the counter's initial value or the number of received PDUs). For more information on the runnable functions, refer to [Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager](#) on page 94.

**Counter behavior when restarting an executable application** When you stop and restart the executable application, the counter behavior depends on the setting specified for the **Initial value usage** property of the bus configuration. The specified setting also influences the value provided by the PDU Counter function port, as shown in the following table:

Initial Value Usage Property Set to	Effect
At first application start	The counter continues counting even if the application is stopped and restarted, i.e., the counter value is not reset. The value provided by the PDU Counter function port is not reset either. Therefore, the function port provides the last received counter value when you restart the application.
At every application start	The counter stops counting and is reset to 0 when you restart the application. The value of the PDU Counter function port is reset to the value specified for the function port's <b>Initial value</b> property.

---

**Restrictions for using the PDU RX Status feature** You cannot use the PDU RX Status and the Frame Access feature in parallel. If you add both features to a PDU, conflicts occur. This also applies if the PDU is included in another PDU and the Frame Access feature is added to this PDU.

---

**Related topics**

**References**

[Common Function Block Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)  
[Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Applying User Code to PDUs

### Introduction

You can apply user code to PDUs that are assigned to the Simulated ECUs, Inspection, or Manipulation part of a bus configuration, for example, to write checksum values that are calculated according to user-specific algorithms to ISignals of the related PDUs.

For basic information on user code, refer to [Working with User Code](#) on page 84.

### Conditions for applying user code to PDUs

To apply user code to a PDU, you must add the PDU User Code feature to the selected PDU. In general, you can add the feature to the following PDU types:

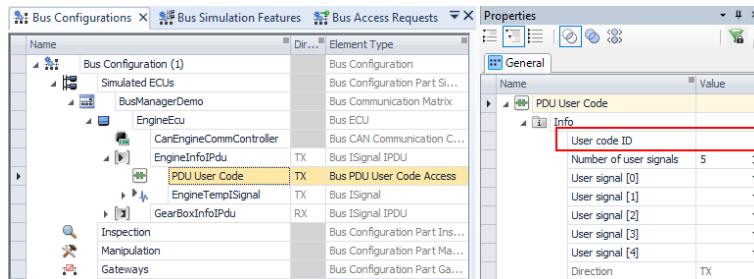
- [Basic PDUs](#)
- [Secured IPDUs](#)

However, you can add the feature to a TX general-purpose PDU only if the PDU does not contain a global time domain in the bus configuration.

### Effects of adding the PDU User Code feature

Adding the PDU User Code feature to a PDU has the following effects:

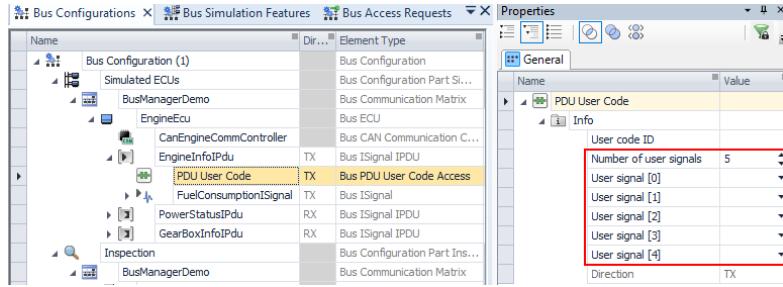
- A User code ID property is available that lets you reference the user code ID of the required user code implementation. You can access the property in the following ways:
  - In the Properties Browser when you select the PDU User Code node in the Bus Configurations table.



- In the related bus configuration features table, i.e., Bus Simulation Features, Bus Inspection Features, or Bus Manipulation Features table, as shown in the following example.

Name	Element Type	Bus Configuration	Communication...	ECU	PDU R...	PDU User Code	User Code ID
DoorLeftControlPdu	Bus ISignal IP...	TX	Bus ManagerDemo	BodyContr...	Disabled	Enabled	
DoorRightControlPdu	Bus ISignal IP...	TX	Bus ManagerDemo	BodyContr...	Disabled	Disabled	
CarLockControlPdu	Bus ISignal IP...	TX	Bus ManagerDemo	BodyContr...	Disabled	Enabled	

- A Number of user signals property and, depending on the specified number of user signals, User signal [<number>] properties are available that let you specify the user signals. You can access the properties when you select the PDU User Code node in the Bus Configurations table.



- If you add the PDU User Code feature to an RX PDU, a Result function port is available. The function port can provide data of the user code, for example, the verification result of a received checksum value that was evaluated by the user code. The following illustration is an example of the function port as displayed in the Bus Configurations table:

	PowerStatusIPdu	RX	Bus ISignal IPDU
	PDU User Code	RX	Bus PDU User Code Access
	PowerStatusIPdu Result		

- If you add the PDU User Code feature to a PDU that is assigned to the Manipulation part, additional manipulation-specific properties and function ports are available. Refer to [Manipulation-specific properties and function ports](#) on page 173.

## Workflow for applying user code to a PDU

Applying user code to a PDU consists of the following workflow steps:

- Specify the user code, i.e., the required user-specific algorithms, according to your requirements in C code (\*.c, \*.h) or C++ code (\*.cpp, \*.hpp).
- Extend the user code by functions for initializing the functionalities of the user code and for exchanging data with the Bus Manager, such as accessing properties of PDUs or writing checksum values to ISignals of PDUs. Use functions of the `DsBusCustomCode` and `DsBusCustomCode_PduUserCode` modules of the Bus Custom Code interface for this purpose.  
For more information, refer to [Bus Custom Code Interface Handling](#).
- Add the code files, i.e., the user code implementation, to the ConfigurationDesk application. Refer to [Implementing user code in ConfigurationDesk applications](#) on page 84.
- Add the PDU User Code feature to a PDU. Refer to [Adding bus configuration features](#) on page 118.
- Reference the required user code. To do so, type the user code ID as specified in the user code implementation into the edit field of the User code ID property. In the user code implementation, the user code ID is specified via the `DS_BUS_CUSTOM_FEATURE_NAME` definition of the related source file (\*.c, \*.cpp).

Depending on the PDU to which you add the PDU User Code feature, you can additionally specify the following settings:

- If you add the feature to a PDU that contains ISignals, you can specify user signals. Refer to [Specifying user signals](#) on page 172.

- If you add the feature to an RX PDU, you can specify an initial value for the Result function port. Refer to [Data provided by the Result function port](#) on page 172.

**Specifying user signals**

If you add the PDU User Code feature to a PDU that contains ISignals, you can specify user signals. User signals allow the user code to directly access ISignals of the PDU. You can specify user signals via the following properties:

Property	Value Range	Description
Number of user signals	0 ... 20	Lets you specify the maximum number of user signals, i.e., the maximum number of signals that can be accessed by the user code. The specified number determines the available number of User signal [<number>] properties.
User signal [0] ... User signal [<max. number of user signals>]	<available ISignals according to communication matrix>	Lets you map an ISignal that is specified for the PDU in the communication matrix to the user signal, regardless of whether the ISignal is assigned to the bus configuration. At run time, the user code can directly access the mapped ISignal, e.g., to use the ISignal value for calculating checksum values or to overwrite the original ISignal value with a calculated value. If a user signal is not mapped to an ISignal, the user signal is not configured, i.e., the user code cannot access this user signal at run time.

**Run-time behavior of applied user code**

Depending on the direction of the PDU ([TX](#) or [RX](#)), applying user code to a PDU results in the following behavior at run time:

- In case of a TX PDU, the PDU data is provided to the user code each time the transmission of the TX PDU is triggered. On the basis of the PDU data, the user-specified algorithms are executed and the calculated values are written to the PDU. Then, the PDU is transmitted on the bus.  
However, if you add the PDU User Code feature to a TX PDU that is assigned to the Manipulation part, this behavior applies only if the feature is enabled at run time. Refer to [Configuring the run-time behavior for manipulation](#) on page 174.  
The timing for executing the user code depends on the related Bus Configuration task. Refer to [Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager](#) on page 94.
- In case of an RX PDU, the PDU data is provided to the user code each time the RX PDU is received on the bus. On the basis of the received PDU data, the user-specified algorithms are executed. For example, a received checksum value can be passed to the user code to be evaluated by the related verification algorithms.

**Data provided by the Result function port**

For RX PDUs, a Result function port is available. The Result function port can provide data of the user code, for example, the verification result of a received and evaluated checksum value. However, the specific data that is provided by the Result function port depends on the definitions in the related user code implementation.

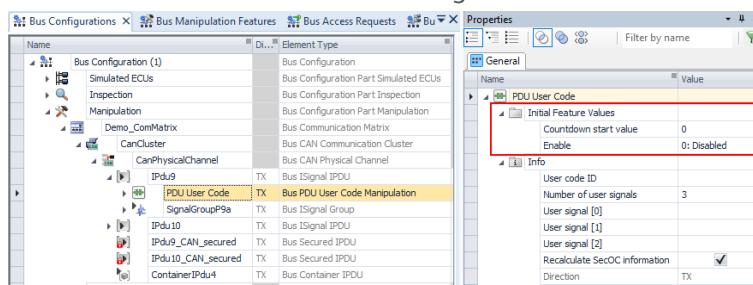
By default, the initial function port value is 0. Via the function port's Initial value property, you can specify a user-defined initial value in the range

$0 \dots \text{UInt32}_{\max}$ . To access the function port value at run time, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

## Manipulation-specific properties and function ports

In addition to the feature-specific properties and function ports, the following manipulation-specific properties and function ports are available if you add the PDU User Code feature to a PDU that is assigned to the Manipulation part:

- A Countdown start value and an Enable property are available that let you configure the run-time behavior for applying the user code to the PDU. You can access the properties in following ways:
- You can access both properties in the Properties Browser when you select the PDU User Code node in the Bus Configurations table.



- You can also access the Countdown start value property in the Bus Manipulation Features table.

PDU Features							
Name	Element Type	Bus Conf...	Comm...	Comm...	Countdown Start Value	User Code ID	PDU User Code Recalculate SecOC Information (User Code)
ContainerIPdu4	Bus Container IPDU	TX	Bus Conf...	Demo_Co...	CanCluster	0	Enabled <input checked="" type="checkbox"/>
IPdu9	Bus Signal IPDU	TX	Bus Conf...	Demo_Co...	CanCluster	0	Disabled
IPdu9_CAN_secured	Bus Secured IPDU	TX	Bus Conf...	Demo_Co...	CanCluster	0	Disabled
IPdu10	Bus Signal IPDU	TX	Bus Conf...	Demo_Co...	CanCluster	0	Disabled
IPdu10_CAN_secured	Bus Secured IPDU	TX	Bus Conf...	Demo_Co...	CanCluster	0	Disabled
ContainerIPdu4	Bus Container IPDU	TX	Bus Conf...	Demo_Co...	CanCluster	0	Enabled <input checked="" type="checkbox"/>

- If the PDU is involved in secure onboard communication, a Recalculate SecOC information checkbox is available. You can access the checkbox in the following ways:
  - In the Properties Browser when you select the PDU User Code node in the Bus Configurations table.
  - In the Recalculate SecOC Information (User Code) column of the Bus Manipulation Features table.

If the checkbox is selected, the authentication information of the affected secured IPDU is recalculated before transmission. For more information, refer to [Recalculating End-to-End Protection and Authentication Information](#) on page 135.

- The following function ports are available:
  - Current Countdown Value function port
  - Countdown Start Value function port
  - Enable function port
  - Enable State function port

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

	IPdu9	TX	Bus ISignal IPDU
	PDU User Code	TX	Bus PDU User Code Manipulation
	IPdu9 Current Countdown Value		
	IPdu9 Countdown Start Value		
	IPdu9 Enable		
	IPdu9 Enable State		

### Configuring the run-time behavior for manipulation

When you add the PDU User Code feature to a TX PDU that is assigned to the Manipulation part, the user code applies to the PDU only if the feature is enabled at run time.

You can enable the feature via the **Enable** property or the **Initial value** and **Initial substitute value** properties of the **Enable** function port as follows:

- Set the related property to 1: Permanently enabled.  
The user code applies to the PDU until you disable the feature explicitly, e.g., via experiment software.
- Set the related property to 2: Temporarily enabled.  
The user code applies to the PDU for a defined number of times. You can specify the number of times via the **Countdown start value** property or the **Initial value** and **Initial substitute value** properties of the **Countdown Start Value** function port.

You can access the current enable state and countdown value via the **Enable State** and **Current Countdown Value** function ports, respectively. For more information, refer to [Basics on Bus Manipulation Features](#) on page 131.

By default, you can change the specified settings via experiment software at run time. If you want to change the settings via a behavior model or provide function port values to a behavior model, you must enable model access for the related function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

### Restrictions for using the PDU User Code feature

You cannot use the PDU User Code and the Frame Access feature in parallel. If you add both features to a PDU, conflicts occur. This also applies if the PDU is included in another PDU and the Frame Access feature is added to this PDU.

### Related topics

### References

- [Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)
- [Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Verifying the Authentication Information of Received Secured IPDUs

### Introduction

You can verify the authentication information of received [secured IPDUs](#) that are assigned to the Simulated ECUs or Inspection part of a bus configuration.

### Conditions for verifying the authentication information of received secured IPDUs

To verify the authentication information of a received secured IPDU, you must do the following:

- Enable SecOC support for the bus configuration and provide user code for secure onboard communication. The user code must contain the algorithms for verifying the authentication information.  
Refer to [Implementing Secure Onboard Communication in Executable Applications](#) on page 87.
- Add the SecOC feature to the RX secured IPDU.

### Effects of adding the SecOC feature

Depending on the bus configuration part, the following function ports are available when you add the SecOC feature to a secured IPDU:

Function Port	Available for	Purpose
Enable Verification	▪ Simulated ECUs ▪ Inspection	Lets you enable and disable the verification of the authentication information.
State	▪ Simulated ECUs ▪ Inspection	Can provide the state of the verification, for example, whether the verified authentication information is valid.
Authenticator Value	Inspection	Provides the authenticator value that is received on the bus.
Freshness Value	Inspection	Provides the freshness value that is received on the bus.
Calculated Freshness Value	Inspection	Can provide the freshness value that is calculated in the user code.

The following illustration is an example of the function ports as displayed in the Bus Configurations table:



### Run-time behavior of verifying authentication information

When a secured IPDU for which the SecOC feature was added is received, the PDU data and the value of the Enable Verification function port are provided to the user code.

The definitions in the user code must determine which value of the Enable Verification function port enables the verification. If the verification is enabled,

the PDU data can be evaluated by verification algorithms that are specified in the user code. The State function port can provide data of the user code, for example, to indicate that a received freshness value was invalid.

When you add the SecOC feature to a secured IPDU that is assigned to the Inspection part, you can additionally do the following:

- Access the authenticator value and freshness value that is received with the secured IPDU via the Authenticator Value and Freshness Value function port, respectively.
- Access data of the user code via the Calculated Freshness Value function port. For example, the function port can provide the complete freshness value that was calculated in the user code. However, the actual provided data depends on the definitions in the user code.

However, to access function port values at run time, you must enable model access or test automation support for the related function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

The Bus Manager processes received secured IPDUs even if the verification indicates an error. If you want to reject a secured IPDU in this case, you must model the rejection in a behavior model, e.g., by using data that is provided by the State function port.

#### **Recommended usage of specific function port values**

The specifications in the user code determine how the values of the Enable Verification and State function ports are evaluated. The following table provides an overview of the recommended usage of specific function port values.

Function Port	Value	Use For
Enable Verification	0	Disable verification.
	1	Enable verification.
State	0	<code>SECOC_VERIFICATIONSUCCESS</code> : Verification was successful.
	1	<code>SECOC_VERIFICATIONFAILURE</code> : Verification failed, for example, because the related verification function of the user code indicates an error.
	2	<code>SECOC_FRESHNESSFAILURE</code> : Verification failed because of an unexpected freshness value.
	62	<code>SECOC_MESSAGELINKERFAILURE</code> : Verification failed because of an unexpected message linker value.
	63	<code>SECOC_VERIFICATIONNOTEXECUTED</code> : Verification is not executed because it is disabled.

#### **Specifying initial function port values**

When you add the SecOC feature to a secured IPDU, the function ports provide default initial values. Via the Initial value property of each function port, you can specify a user-defined initial value. The value range depends on the related function port, as shown in the following table.

Function Port	Initial Value	Value Range
Enable Verification	0	0 ... UInt32 <sub>max</sub>
State	63	0 ... UInt32 <sub>max</sub>
Authenticator Value	<ul style="list-style-type: none"> <li>▪ Vector, vector size determined by the value of the Authentication TX length property of the secured IPDU</li> <li>▪ Initial value of each vector element: 0</li> </ul>	0 ... 255 for each vector element
Freshness Value	0	0 ... UInt64 <sub>max</sub>
Calculated Freshness Value	0	0 ... UInt64 <sub>max</sub>

For more information on initial values, refer to [Specifying initial values](#) on page 120.

## Related topics

## References

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

[Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

# Invalidating the Authenticator of Secured IPDUs

## Introduction

You can invalidate the authenticator of secured IPDUs that are assigned to the Manipulation part of a bus configuration.

## Conditions for invalidating the authenticator of secured IPDUs

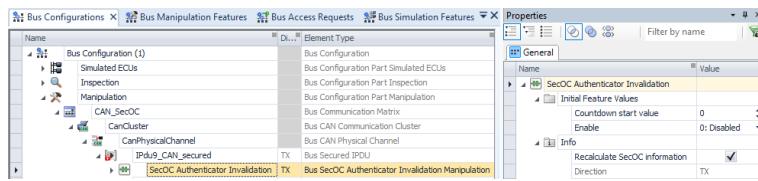
To invalidate the authenticator of a secured IPDU, you must do the following:

- Enable SecOC support for the bus configuration and provide user code for secure onboard communication.  
Refer to [Implementing Secure Onboard Communication in Executable Applications](#) on page 87.
- Add the SecOC Authenticator Invalidation feature to the secured IPDU.

### Effects of adding the SecOC Authenticator Invalidiation feature

Adding the SecOC Authenticator Invalidation feature to a secured IPDU has the following effects:

- A Countdown start value and an Enable property are available that let you configure the run-time behavior for invalidating the authenticator. You can access the properties in following ways:
- You can access both properties in the Properties Browser when you select the SecOC Authenticator Invalidation node in the Bus Configurations table.



- You can also access the Countdown start value property in the Bus Manipulation Features table.

PDU Features					
Name	Element Type	Bus Conf...	Communic...	Countdown Start Value	SecOC Authenticator Invalidatio...
IPdu10_CAN_secured	Bus Secured IPDU TX	Bus Confi...	CAN_SecOC	CanCluster	Disabled
IPdu5_CAN_secured	Bus Secured IPDU TX	Bus Confi...	CAN_SecOC	CanCluster_0	Enabled

- A Recalculate SecOC information checkbox is available. You can access the checkbox in the following ways:
  - In the Properties Browser when you select the SecOC Authenticator Invalidation node in the Bus Configurations table.
  - In the Recalculate SecOC Information (Authenticator Invalidiation) column of the Bus Manipulation Features table.

#### Note

Selecting this checkbox might result in unintended behavior at run time. Refer to [Recalculating authentication information](#) on page 179.

- The following function ports are available:
  - Current Countdown Value function port
  - Countdown Start Value function port
  - Enable function port
  - Enable State function port

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

IPdu5_CAN_secured	TX	Bus Secured IPDU
SecOC Authenticator Invalidatio...	TX	Bus SecOC Authenticator Invalidatio...
IPdu5_CAN_secured Current Countdown Value		
IPdu5_CAN_secured Countdown Start Value		
IPdu5_CAN_secured Enable		
IPdu5_CAN_secured Enable State		

### Configuring the run-time behavior for invalidating the authenticator

As long as the SecOC Authenticator Invalidation feature is enabled at run time, the authenticator is invalidated: After the authentication information is calculated in the user code and written to the secured IPDU, the value of the most significant bit of the authenticator is inverted.

You can enable the feature via the **Enable** property or the **Initial value** and **Initial substitute value** properties of the **Enable** function port as follows:

- Set the related property to 1: **Permanently enabled**.

The most significant bit of the authenticator is invalidated until you disable the feature explicitly, e.g., via experiment software.

- Set the related property to 2: **Temporarily enabled**.

The most significant bit of the authenticator is invalidated for a defined number of times. You can specify the number of times via the **Countdown start value** property or the **Initial value** and **Initial substitute value** properties of the **Countdown Start Value** function port.

You can access the current enable state and countdown value via the **Enable State** and **Current Countdown Value** function ports, respectively. For more information, refer to [Basics on Bus Manipulation Features](#) on page 131.

By default, you can change the specified settings via experiment software at run time. If you want to change the settings via a behavior model or provide function port values to a behavior model, you must enable model access for the related function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

### Recalculating authentication information

In general, if the **Recalculate SecOC information** checkbox is selected, the authentication information of the secured IPDU is recalculated before the secured IPDU is transmitted.

#### Note

You cannot change this setting at run time. Therefore, if the checkbox is selected, the invalidated authenticator will never be transmitted.

In most cases, it is therefore recommended to clear the **Recalculate SecOC information** checkbox.

However, if the secured IPDU is affected by other bus manipulation features for which the **Recalculate SecOC information** checkbox is selected, the authentication information might be recalculated even if you clear the checkbox for this feature. For more information, refer to [Recalculating End-to-End Protection and Authentication Information](#) on page 135.

### Specifying saturation values

For the **Countdown Start Value** and **Enable** function ports, you can specify user-defined saturation values to limit the minimum and maximum values of the function ports. If you want to do this, you must first set the bus configuration's **Saturation usage** property to **User min/max value**. Refer to [Specifying saturation values for function imports](#) on page 123.

**Related topics****References**

- [Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)
- [Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Overwriting the Freshness Value of Secured IPDUs

**Introduction**

You can overwrite the freshness value of secured IPDUs that are assigned to the Manipulation part of a bus configuration.

**Conditions for overwriting the freshness value of secured IPDUs**

To overwrite the freshness value of a secured IPDU, you must do the following:

- Enable SecOC support for the bus configuration and provide user code for secure onboard communication.
- Refer to [Implementing Secure Onboard Communication in Executable Applications](#) on page 87.
- Add the SecOC Freshness Overwrite Value feature to the secured IPDU.

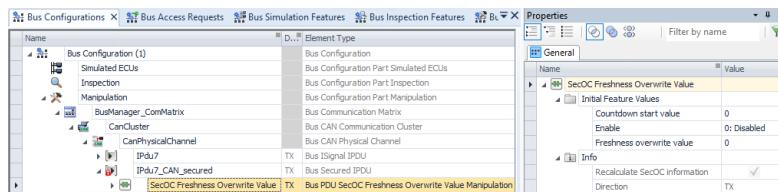
**Effects of adding the SecOC Freshness Overwrite Value feature**

Adding the SecOC Freshness Overwrite Value feature to a secured IPDU has the following effects:

- The following properties are available:
  - A Countdown start value and an Enable property are available that let you configure the run-time behavior for overwriting the freshness value.
  - A Freshness Overwrite Value property that lets you specify an initial value for overwriting the freshness value.
  - A Recalculate SecOC information property, indicating that the authentication information of the secured IPDU is recalculated before the PDU is transmitted on the bus.

You can access the properties in following ways:

- You can access all properties in the Properties Browser when you select the SecOC Freshness Overwrite Value node in the Bus Configurations table.



- You can also access the Countdown start value and Freshness Overwrite Value properties in the Bus Manipulation Features table.

PDU Features				
Name	Immuni...	Countdown Start Value	SecOC Freshness Overwrite Value	Freshness Overwrite Value
IPdu7	nCluster	0	Enabled	0
IPdu7_CAN_secured	nCluster	0	Enabled	0

- The following function ports are available:
  - Current Countdown Value function port
  - Countdown Start Value function port
  - Enable function port
  - Enable State function port
  - Freshness Overwrite Value function port

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

IPdu7_CAN_secured	TX	Bus Secured IPDU
SecOC Freshness Overwrite Value	TX	Bus PDU SecOC Freshness Overwrite Value Manipulation
IPdu7_CAN_secured Current Countdown Value		
IPdu7_CAN_secured Countdown Start Value		
IPdu7_CAN_secured Enable		
IPdu7_CAN_secured Enable State		
IPdu7_CAN_secured Freshness Overwrite Value		

#### Configuring the run-time behavior for overwriting the freshness value

As long as the SecOC Freshness Overwrite Value feature is enabled at run time, the original freshness value is overwritten with a specified overwrite value.

You can enable the feature via the Enable property or the Initial value and Initial substitute value properties of the Enable function port as follows:

- Set the related property to 1: Permanently enabled.  
The freshness value is overwritten until you disable the feature explicitly, e.g., via experiment software.
- Set the related property to 2: Temporarily enabled.  
The freshness value is overwritten for a defined number of times. You can specify the number of times via the Countdown start value property or the Initial value and Initial substitute value properties of the Countdown Start Value function port.

You can access the current enable state and countdown value via the Enable State and Current Countdown Value function ports, respectively. For more information, refer to [Basics on Bus Manipulation Features](#) on page 131.

By default, you can change the specified settings via experiment software at run time. If you want to change the settings via a behavior model or provide function port values to a behavior model, you must enable model access for the related function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

#### Specifying overwrite values

You can specify an overwrite value via the Freshness overwrite value property in the value range 0 ... UInt64<sub>max</sub>. The specified value applies to the Initial value

and Initial substitute value properties of the Freshness Overwrite Value function port automatically.

By default, you can change the specified overwrite value via experiment software at run time. If you want to change the overwrite value via a behavior model, you must enable model access for the Freshness Overwrite Value function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

#### Effects of overwriting the freshness value

For the SecOC Freshness Overwrite Value feature, the Recalculate SecOC information checkbox is selected and read-only. Therefore, when the feature is enabled at run time, the authentication information of the secured IPDU is always recalculated before the PDU is transmitted on the bus.

**Effects on the secured IPDU** When the SecOC Freshness Overwrite Value feature is enabled, the authentication information of the secured IPDU is calculated as follows:

- The authenticator is recalculated by using the specified overwrite value.
- If the authentication information that is included in the secured IPDU consists of the authenticator and the freshness value, the specified overwrite value is included in the secured IPDU. However, depending on the Freshness value TX length setting of the secured IPDU, the overwrite value might be truncated.

For basic information on the authentication information, refer to [Secure onboard communication for PDUs](#) on page 41. For more information on recalculating the authentication information, refer to [Recalculating End-to-End Protection and Authentication Information](#) on page 135.

**Effects on the SecOC Authenticator Invalidation feature** If the SecOC Freshness Overwrite Value and the SecOC Authenticator Invalidation features are enabled at the same time, the authentication information is recalculated after the authenticator was invalidated by the SecOC Authenticator Invalidation feature. Therefore, the invalidated authenticator is not transmitted on the bus. Refer to [Invalidating the Authenticator of Secured IPDUs](#) on page 177.

#### Specifying saturation values

For the Countdown Start Value, Enable, and Freshness Overwrite Value function ports, you can specify user-defined saturation values to limit the minimum and maximum values of the function ports. If you want to do this, you must first set the bus configuration's Saturation usage property to User min/max value. Refer to [Specifying saturation values for function imports](#) on page 123.

#### Related topics

#### References

- [Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)
- [Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Triggering the Execution of Functions in a Behavior Model via RX Interrupts

### Introduction

You can use CAN RX PDUs that are assigned to the Simulated ECUs part of a bus configuration to trigger the execution of functions in a behavior model via RX interrupts.

### Conditions for triggering the execution of functions in a behavior model via RX interrupts

To trigger the execution of functions in a behavior model via RX interrupts, you must add the PDU RX Interrupt feature to a CAN RX PDU. You can add the PDU RX Interrupt feature to a CAN RX PDU only if it fulfills all of the following conditions:

- The CAN RX PDU is directly included in one frame and not affected by secured IPDUs. Therefore, the PDU can be one of the following types:
  - [Container IPDU](#) that is neither included in a secured IPDU nor contains secured IPDUs
  - [Multiplexed IPDU](#) that is neither included in a secured IPDU, nor configured as a contained IPDU or J1939-compliant PDU
  - [Basic PDU](#) that is neither included in a secured IPDU, nor configured in any of the following ways:
    - Contained IPDU
    - Static part or dynamic part of a multiplexed IPDU
    - J1939-compliant PDU
- If the PDU is a container IPDU or multiplexed IPDU, all its included PDUs must be assigned to the same ECU connector port as the PDU itself.
- The frame in which the PDU is included has exactly one frame triggering.

### Effects of adding the PDU RX Interrupt feature

When you add the PDU RX Interrupt feature to a CAN RX PDU, the following ports are available for the PDU:

- PDU RX Interrupt Enable function port

This function port lets you enable and disable the PDU RX Interrupt feature.

- PDU Received event port

This event port provides an [event](#) (i.e., the RX interrupt) each time the related CAN RX PDU is received.

The following illustration is an example of the ports as displayed in the Bus Configurations table:

	PowerStatusIPdu	RX	Bus ISignal IPDU
	PDU RX Interrupt	RX	Bus PDU RX Interrupt Access
	PowerStatusIPdu RX Interrupt Enable		
	PowerStatusIPdu Received		

---

<b>Use scenario for working with RX interrupts</b>	<p>RX interrupts let you work independently of the cyclic Bus Configuration task. This has the following characteristics:</p> <ul style="list-style-type: none"><li>▪ The timing for triggering functions in the behavior model does not depend on the Bus Configuration task. Instead, the functions can be triggered by the RX interrupt, i.e., by the event that is generated each time the related RX PDU is received on the bus.</li><li>▪ The data that is available for the function ports of the RX PDU and the involved elements is updated with the reception of the PDU, i.e., the timing for updating the function port values does not depend on the Bus Configuration task.</li></ul> <p>The involved elements are elements that are included in the PDU and that are assigned to the Simulated ECUs part of the same bus configuration. Elements that are included in the RX PDU can be ISignals, contained IPDUs, or static or dynamic part IPDUs.</p>
<b>Required steps for using RX interrupts</b>	<p>To use RX interrupts in the behavior model, you must do the following:</p> <ul style="list-style-type: none"><li>▪ Add the PDU RX Interrupt feature to a CAN RX PDU.</li><li>▪ Map the PDU Received event port to a <a href="#">runnable function port</a> .</li><li>▪ Refer to <a href="#">Mapping event ports to runnable function ports</a> on page 185.</li><li>▪ Model the triggering behavior in the behavior model.</li></ul> <p>The modeling in the behavior model depends on your requirements and the type of the behavior model. For information on modeling the triggering behavior in a MATLAB/Simulink model, refer to <a href="#">Modeling the triggering behavior in a MATLAB/Simulink behavior model</a> on page 186.</p>
<b>Enabling and disabling the PDU RX Interrupt feature</b>	<p>The value of the PDU RX Interrupt Enable function port determines the enable state of the PDU RX Interrupt feature.</p> <p><b>Specifying the initial state</b> Via the Initial value property of the function port, you can specify the initial state of the PDU RX Interrupt feature as follows:</p> <ul style="list-style-type: none"><li>▪ False (0): The PDU RX Interrupt feature is disabled.</li><li>▪ True (1): The PDU RX Interrupt feature is enabled. This is the default state.</li></ul> <p><b>Changing the enable state at run time</b> To change the enable state at run time, you must enable model access or test automation support for the function port. Refer to <a href="#">Configuring Function Ports for Bus Configuration Features</a> on page 120.</p>

**Note**

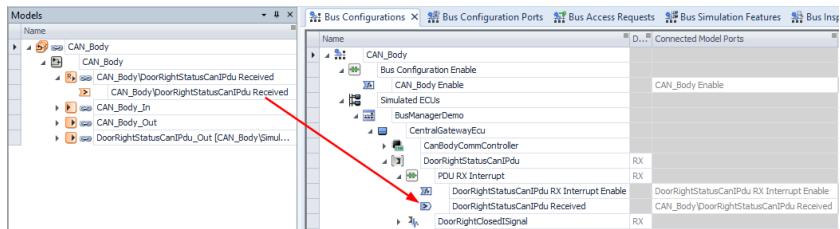
To ensure optimum run-time behavior, it is recommended to enable model access for the PDU RX Interrupt Enable function port and to use it in the behavior model by mapping it to a [model port block](#). In the behavior model, make sure that the related model port block is executed according to a periodic task, i.e., a task that is executed cyclically. For information on modeling this behavior in a MATLAB/Simulink behavior model, refer to [Modeling the triggering behavior in a MATLAB/Simulink behavior model](#) on page 186.

**Effects of disabling the PDU RX Interrupt feature** When you add the PDU RX Interrupt feature to a PDU, the processing of the PDU data according to the cyclic Bus Configuration task is disabled, regardless of whether you enable or disable the PDU RX Interrupt feature. Thus, disabling the PDU RX Interrupt feature does not enable the cyclic processing of the PDU data according to the Bus Configuration task. Instead, if you disable the PDU RX Interrupt feature, the processing of the PDU data is disabled completely: When the PDU is received on the bus, the data of the related function ports is not updated, no event is generated (i.e., no RX interrupt), and the affected functions of the behavior model are not triggered.

### Mapping event ports to runnable function ports

To use the PDU RX Interrupt feature at run time, you must map the PDU Received event port to a runnable function port.

Runnable function ports are available when you add a behavior model to the ConfigurationDesk application and this behavior model provides [Runnable Function blocks](#). To map an available runnable function port to an event port, you can drag the runnable function port from the Model Browser to the event port in the Bus Configurations or Bus Configuration Ports table, for example.



For a detailed example, refer to [Example of Mapping Model Ports to Bus Configuration Ports via Tables](#) on page 249.

When you map a runnable function port to the event port, the related [Runnable Function](#) and [I/O Event](#) are assigned to a task. For more information, refer to [Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager](#) on page 94.

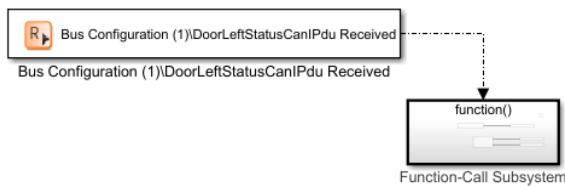
If you do not map the PDU Received event port to a runnable function port, a conflict occurs. If you do not resolve the conflict, the PDU RX Interrupt feature is inactive and you cannot use the feature at run time. Instead, the PDU data is processed according to the Bus Configuration task.

**Tip**

If you work with a MATLAB/Simulink behavior model and the model does not provide Runnable Function blocks, you can create the required blocks from within the ConfigurationDesk application. Refer to [Points to note for creating model port blocks](#) on page 187.

### Modeling the triggering behavior in a MATLAB/Simulink behavior model

If you work with a MATLAB/Simulink behavior model, you can model the triggering behavior by using a [Runnable Function block](#) and a function-call subsystem. To do this, you can map the Runnable Function block that provides the event of the PDU RX Interrupt feature to the function $\square$  port of a function-call subsystem. In this case, the subsystem is executed each time the event is generated. Therefore, the functions that you model in the subsystem are executed each time the related RX PDU is received on the bus.



In the following cases, no event can be generated:

- The PDU RX Interrupt feature is disabled.
- The related communication controller is disabled via the Communication Controller Enable feature.
- The related bus configuration is disabled via the Bus Configuration Enable feature.
- The RX PDU cannot be received because the transmission of the related TX PDU is disabled via the PDU Enable feature, the Communication Controller Enable feature, or the Bus Configuration Enable feature.

**Note**

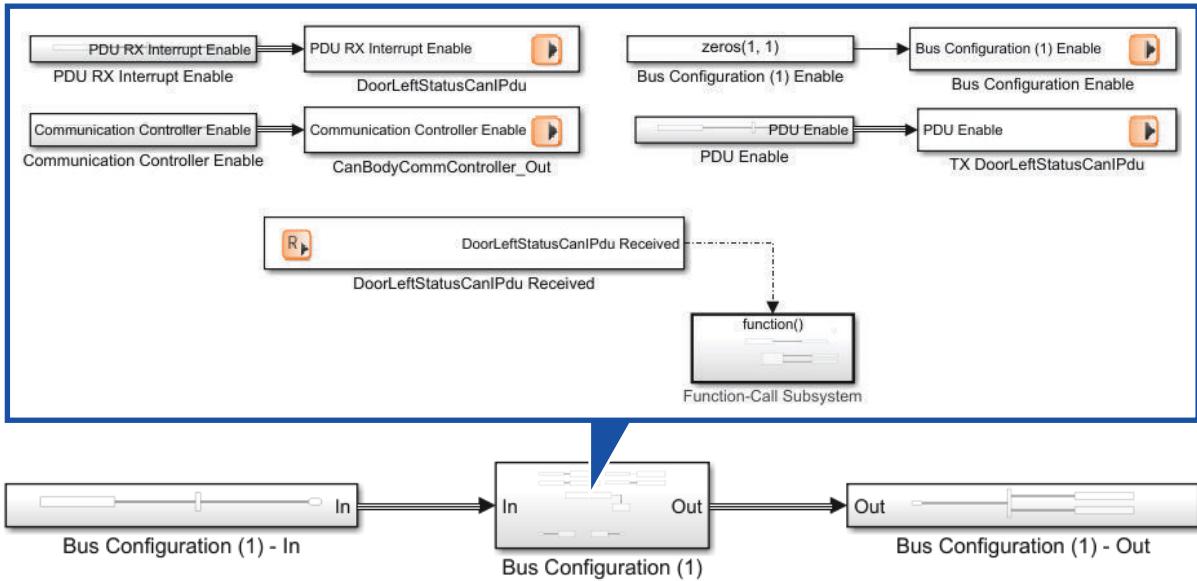
Depending on the modeling in the behavior model and the specified settings in the ConfigurationDesk application, you cannot enable the features again at run time. In this case, the event generation is permanently disabled and the function-call subsystem, including its functions, will never be executed at run time.

To ensure that you can enable the event generation again at run time, do the following:

- In the ConfigurationDesk application, enable model access for the PDU RX Interrupt Enable function port and map it to a [data outport](#).
- In the Simulink model, ensure that the related [model port block](#) is executed according to a periodic task, i.e., do not place it in a function-call subsystem. Instead, place it in a subsystem that is executed according to the Bus Configuration task, for example.

- If you use model port blocks that are mapped to the Enable function port of the Communication Controller Enable feature, Bus Configuration Enable feature, or PDU Enable feature, ensure that these model port blocks are also executed according to a periodic task.

In the following example, the model port blocks that are mapped to Enable function ports are included in the Bus Configuration (1) subsystem, which is executed according to the Bus Configuration task. Because the function-call subsystem is placed on the same level as the model port blocks, it does not affect their execution.



If your model does not have suitable model port blocks, you can create the required blocks from within the ConfigurationDesk application. However, there are some points to note.

#### Points to note for creating model port blocks

For unmapped event ports and function ports of bus configurations, you can create [model port blocks](#), i.e., Runnable Function blocks and data port blocks. However, the default settings of bus configurations prohibit the creation of Runnable Function blocks.

**Basics on creating model port blocks** In general, when you select a bus configuration, you can create the required model port blocks in a new or an existing Simulink model via the **Generate New Simulink Model Interface** or **Propagate to Simulink Model** commands.

**Note**

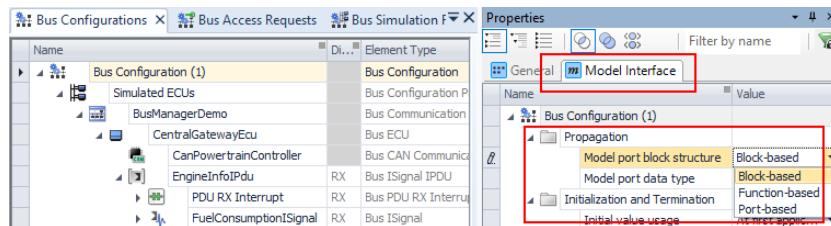
If ports of the bus configuration are already mapped to model ports and the related Simulink model is available in the Model Browser, you change the model when you use the **Propagate to Simulink Model** command. In Simulink, you cannot undo the changes. Refer to [Basics of Connecting ConfigurationDesk and Simulink Behavior Models \(ConfigurationDesk Real-Time Implementation Guide\)](#).

To create Runnable Function blocks, you must change the default setting for the structure of model port blocks.

**Specifying the structure of model port blocks** Each bus configuration provides a Model port block structure property that lets you specify the structure of model port blocks when you create the blocks from within the ConfigurationDesk application. By default, the property is set to Block-based. This has the following effects when you create model port blocks for the bus configuration:

- Up to two data port blocks are created:
  - If the bus configuration has unmapped function imports, one data port block providing [data outputs](#) is created.
  - If the bus configuration has unmapped function exports, one data port block providing [data imports](#) is created.
- No Runnable Function blocks are created, even if the bus configuration has unmapped event ports.

To create Runnable Function blocks for unmapped event ports and separate data port blocks for unmapped function ports, you must set the Model port block structure property to Function-based or Port-based.

**Note**

Setting the Model port block structure property to Function-based or Port-based can significantly increase the number of created model port blocks. This might reduce performance.

To optimize the number of created model port blocks, it is recommended to create model port blocks according to the following workflow:

1. Configure the bus communication via bus configuration features according to your requirements but without the PDU RX Interrupt feature.
2. Enable model access for the function ports whose data you want to use in the behavior model but for which you do not need separate model port blocks.

3. Use the default setting of the Model port block structure property, i.e., Block-based, and create model port blocks for the bus configuration.
4. Add the PDU RX Interrupt feature to the required PDUs. Enable model access for the function ports whose data you want to use in the behavior model and for which you need separate model port blocks.
5. Set Model port block structure to Function-based or Port-based and create the remaining model port blocks.

For more information, refer to [Specifying Options for Creating Model Port Blocks \(ConfigurationDesk Real-Time Implementation Guide](#) .

---

#### Restrictions for using the PDU RX Interrupt feature

If you use the PDU RX Interrupt feature, the following restrictions apply:

- You cannot use the PDU RX Interrupt and the Frame Access feature in parallel. If you add both features to a PDU, conflicts occur.
- If you use RX interrupts in the behavior model to trigger functions that provide data to bus configurations that is to be transmitted on the bus, the timing for transmitting the data on the bus depends on the cyclic Bus Configuration task.
- In general, you can use one event to trigger the execution of multiple runnable functions by mapping one event port to multiple runnable function ports. However, if you generate bus simulation containers, it depends on the specific simulation platform (e.g., VEOS) whether this triggering behavior is supported.

---

#### Related topics

##### Basics

[Basics on Configuring Tasks \(ConfigurationDesk Real-Time Implementation Guide](#) 

[Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager](#) ..... 94

[Working with Simulink Behavior Models \(ConfigurationDesk Real-Time Implementation Guide](#) 

##### References

[Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties](#) 

# Bus Configuration Features Available for Frames

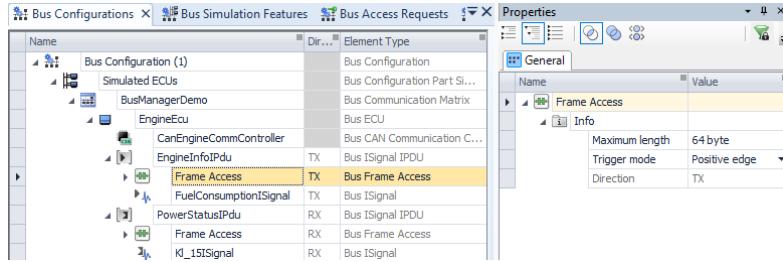
Where to go from here	Information in this section
	<ul style="list-style-type: none"> <li><a href="#">Accessing CAN Frame Settings.....</a> 190</li> <li><a href="#">Manipulating the Payload Length of CAN Frames.....</a> 197</li> <li><a href="#">Suspending the Transmission of Frames.....</a> 200</li> </ul>

## Accessing CAN Frame Settings

<b>Introduction</b>	You can access the settings of CAN frames that are assigned to the Simulated ECUs part of a bus configuration and change settings at run time.
<b>Conditions for accessing CAN frame settings</b>	<p>To access and change settings of a CAN frame at run time, you must add the Frame Access feature to a PDU that is included in the frame. You can add the Frame Access feature to a PDU only if the PDU and the related frame fulfill all of the following conditions:</p> <ul style="list-style-type: none"> <li>▪ The PDU is a CAN PDU that is directly included in one frame. Therefore, the PDU can be of one of the following types: <ul style="list-style-type: none"> <li>▪ <a href="#">Container IPDU</a> that is not included in a secured IPDU</li> <li>▪ <a href="#">Multiplexed IPDU</a> that is neither included in a secured IPDU, nor configured as a contained IPDU or J1939-compliant PDU</li> <li>▪ <a href="#">Secured IPDU</a> that is not configured as a contained IPDU</li> <li>▪ <a href="#">Basic PDU</a> that is neither included in a secured IPDU, nor configured in any of the following ways: <ul style="list-style-type: none"> <li>▪ Contained IPDU</li> <li>▪ Static part or dynamic part of a multiplexed IPDU</li> <li>▪ J1939-compliant PDU</li> <li>▪ General-purpose PDU that contains a global time domain in the bus configuration.</li> </ul> </li> </ul> </li> <li>▪ The PDU has exactly one frame triggering and this triggering is a CAN frame triggering.</li> <li>▪ The related CAN frame has exactly one PDU-to-frame mapping.</li> </ul>
<b>Effects of adding the Frame Access feature</b>	<p>Adding the Frame Access feature to a PDU has the following effects:</p> <ul style="list-style-type: none"> <li>▪ A Maximum length and Trigger mode property are available that let you specify the frame's maximum length and the triggering mode for the frame's</li> </ul>

transmission, respectively. The Trigger mode property is available only for TX PDUs. You can access the properties in two ways:

- In the Properties Browser when you select the Frame Access node in the Bus Configurations table.



- In the Bus Simulation Features table.

PDU Features										
Name	Element Type	Dir...	PDU R...	PDU ...	Use...	SecOC	PDU...	Trigger Mode	Frame Access	Maximum Length
EngineInfoPdu	Bus ISignal IPDU	TX	Disabled	Disabled	Dis...		Dis...	Positive edge	Enabled	64 byte
PowerStatusPdu	Bus ISignal IPDU	RX	Disabled	Disabled				Enabled		64 byte
GearBoxInfoPdu	Bus ISignal IPDU	RX	Disabled	Enabled						

- Depending on the direction of the PDU (TX or RX), the following function ports are available:

Function Ports Available for											
TX PDUs						RX PDUs					
Trigger function port						State function port					
TX Length function port						RX Length function port					
TX Raw Data function port						RX Raw Data function port					
Identifier function port						Identifier function port					
Extended Addressing function port						Extended Addressing function port					
CAN FD Frame Support function port						CAN FD Frame Support function port					
Bit Rate Switch function port						Bit Rate Switch function port					

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

 EngineInfoIPdu	TX	Bus ISignal IPDU
 Frame Access	TX	Bus Frame Access
 EngineInfoFrame Trigger		
 EngineInfoFrame Length		
 EngineInfoFrame Raw Data		
 EngineInfoFrame Identifier		
 EngineInfoFrame Extended Addressing		
 EngineInfoFrame CAN FD Frame Support		
 EngineInfoFrame Bit Rate Switch		
 PowerStatusIPdu	RX	Bus ISignal IPDU
 Frame Access	RX	Bus Frame Access
 PowerStatusFrame State		
 PowerStatusFrame Length		
 PowerStatusFrame Raw Data		
 PowerStatusFrame Identifier		
 PowerStatusFrame Extended Addressing		
 PowerStatusFrame CAN FD Frame Support		
 PowerStatusFrame Bit Rate Switch		

#### Accessing CAN frames via the Frame Access feature

Adding the Frame Access feature to a PDU lets you do the following:

- You can trigger the transmission of the frame. Refer to [Triggering the frame transmission](#) on page 192.
- You can access the length of the frame. Refer to [Accessing the frame length](#) on page 192.
- You can access the payload of the frame in raw data format. Refer to [Accessing the payload of the frame in raw data format](#) on page 193.
- You can access the frame triggering. Refer to [Accessing the frame triggering](#) on page 195.
- You can access the state of the frame. Refer to [Accessing the frame state](#) on page 196.

#### Triggering the frame transmission

When you add the Frame Access feature to a TX PDU, you cannot transmit the PDU according to cyclic timings, e.g., as specified in the communication matrix. To transmit the PDU and the related frame, you must specify the transmission of the frame in the following way:

1. Add the Frame Access feature to the TX PDU.
2. Select a triggering mode via the Trigger mode property.
3. Configure the Trigger function port.

For more information, refer to [Triggering the Transmission of PDUs and Frames via User-Defined Triggers](#) on page 128.

#### Accessing the frame length

When you add the Frame Access feature to a TX or RX PDU, you can specify the maximum payload length of the frame and access the payload length at run time.

**Specifying the maximum payload length** The Maximum length property lets you specify the maximum payload length of the frame in bytes. Valid values are 1 ... 8, 12, 16, 20, 24, 32, 48, and 64 bytes. The default maximum length is 64 bytes. The specified maximum length determines:

- The valid maximum value of the Length function port.
- The data width of the Raw Data function port.

You cannot change the specified maximum length during run time.

**Accessing the frame length at run time** The Length function port provides the payload length of the frame as follows:

- For TX PDUs, the function port value specifies the payload length with which the related frame is transmitted on the bus.
- For RX PDUs, the function port provides the received payload length of the related frame, e.g., to a mapped behavior model.

By default, the initial function port value is the frame's payload length that is specified in the communication matrix. Via the function port's Initial value property, you can specify a user-defined initial payload length in the range 0 ... <maximum length>. In the following cases, the payload length is automatically adjusted at run time:

- For classic CAN frames, the maximum length is saturated to 8 bytes.
- A payload length that is invalid according to CAN FD is extended by padding bytes to the next higher valid length. The value of each padding byte is 255. Refer to [CAN FD protocol](#) on page 47.

To access the payload length at run time, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

#### Accessing the payload of the frame in raw data format

When you add the Frame Access feature to a TX or RX PDU, you can access the payload of the related frame (including the PDU) in raw data format only. The Raw Data function port provides the raw data of the payload as follows:

- For TX PDUs, the function port value specifies the raw data with which the related frame is transmitted on the bus.
- For RX PDUs, the function port provides the received raw data of the related frame, e.g., to a mapped behavior model.

**Configuring the Raw Data function port** The value of the Raw Data function port is a vector. The default value of each vector element is 0. Via the function port's Initial value property, you can specify a user-defined initial raw data in the range 0 ... 255 for each vector element.

To access the raw data at run time, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

**Effects of the Maximum length property** The data width of the Raw Data function port is derived from the Maximum length property. If you change the Maximum length property, the function port's data width adjusts accordingly. Because the data width determines the vector size of the function port value,

this has the following effects on the function port's Initial value and Initial substitute value properties:

- If the data width is reduced, the vector size of both properties reduces accordingly. The vector elements are deleted from right to left and the values specified for these elements are lost, as shown in the following example.

FuelRateEngineFrame Raw Data	
Initialization and Termination	
Initial value	10; 15; 25; 30; 35; 40; 45; 50
Behavior Model	
Test Automation Support	
Activate test automation...	Enabled
Initial switch setting	Model signal
Initial substitute value	10; 15; 25; 30; 35; 40; 45; 50
Info	
Description	
Port type	In
Data type	UInt8
Data width	8

FuelRateEngineFrame Raw Data	
Initialization and Termination	
Initial value	10; 15; 25; 30
Behavior Model	
Test Automation Support	
Activate test automation...	Enabled
Initial switch setting	Model signal
Initial substitute value	10; 15; 25; 30
Info	
Description	
Port type	In
Data type	UInt8
Data width	4

- If the data width is increased, the vector size of both properties increases accordingly. The new vector elements are added to the right of the existing vector elements. The default value of the new vector elements is 0.

**Data provided by the Raw Data function port** The vector size of Raw Data function ports determines the maximum number of bytes that are available for the related Raw Data function port. The available bytes are used as follows:

- The frame data is written to the Raw Data function port bytes from left to right.
- For TX frames, the frame length that is specified via the Length function port determines the number of the Raw Data function port bytes that transmit raw data. If the specified frame length is shorter than the vector size of the Raw Data function port, the values of the unused function port bytes are not transmitted on the bus.
- For RX frames, the received frame length determines the number of function port bytes that are updated with the received frame data. If the received frame length is shorter than the vector size of the Raw Data function port, the unused function port bytes keep their last value.

The value provided by a TX Raw Data function port might not be the value that is transmitted on the bus. This can happen if the length of a TX frame is invalid according to CAN FD. In this case, padding bytes are added to the frame length. The padding bytes' value (i.e., 255) is transmitted on the bus but not available at the TX Raw Data function port.

**Example for accessing the frame's payload via the Frame Access feature** In the following example, the Frame Access feature is added to a TX and an RX instance of a PDU. The vector size of both Raw Data function ports (TX, RX) is 16 bytes. For the TX Raw Data function port, the raw data of all bytes is 40. The actual length of the related TX frame is 9 bytes. Therefore, only the raw data of the first 9 bytes is transmitted on the bus. Because 9 bytes is an invalid length according to CAN FD but 12 bytes is a valid CAN FD length, the frame length is extended by 3 padding bytes with a value of 255 each. The following table provides an overview of this example.

	<b>Byte</b>															
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>	<b>16</b>
Data of TX Raw Data function port	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
Length of TX frame	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-
Padding byte	-	-	-	-	-	-	-	-	255	255	255	-	-	-	-	-
Data transmitted on the bus	40	40	40	40	40	40	40	40	40	255	255	255	-	-	-	-

The data that is transmitted on the bus is received by the RX Raw Data function port of the related RX PDU instance. The data is received as the RX frame data. In the following example, the old data of the RX Raw Data function port was 125. Then, the 12 bytes of the RX frame are received. The first 12 bytes of the RX Raw Data function port are updated with the received data. The last 4 bytes of the function port keep their previous value, as shown in the following table.

	<b>Byte</b>															
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>	<b>16</b>
Old data of RX Raw Data function port	125	125	125	125	125	125	125	125	125	125	125	125	125	125	125	125
New data of RX frame	40	40	40	40	40	40	40	40	40	255	255	255	-	-	-	-
New data of RX Raw Data function port	40	40	40	40	40	40	40	40	40	255	255	255	125	125	125	125

### Tip

On RX side, it is not possible to identify the origin of the received frame data, i.e., it cannot be identified whether the values of the bytes 10, 11, and 12 are padding byte values or values of the frame's original payload.

## Accessing the frame triggering

When you add the Frame Access feature to a TX or RX PDU, you can access the frame triggering via the following function ports:

Function Port	Purpose	Value Range
Extended Addressing function port	Lets you access the identifier format of the frame.	<ul style="list-style-type: none"> <li>▪ 0: Standard identifier format (11-bit)</li> <li>▪ 1: Extended identifier format (29-bit)</li> </ul>
Identifier function port	Lets you access the identifier of the frame.	<p>Depends on the identifier format specified for the Extended Addressing function port:</p> <ul style="list-style-type: none"> <li>▪ Standard identifier format: 0 ... <math>2^{11} - 1</math></li> <li>▪ Extended identifier format: 0 ... <math>2^{29} - 1</math></li> </ul>
CAN FD Frame Support function port	Lets you access the state of the frame's CAN FD support. If CAN FD support is enabled, the payload length of the frame can be up to 64 bytes and the data	<ul style="list-style-type: none"> <li>▪ 0: CAN FD support disabled, i.e., the frame is a classic CAN frame</li> <li>▪ 1: CAN FD support enabled</li> </ul>

Function Port	Purpose	Value Range
	phase of the frame can be transmitted with a higher bit rate.	
Bit Rate Switch function port	Lets you access the state of the bit rate switch. If switching the bit rate is enabled, the data phase of the frame can be transmitted with a higher bit rate. The function port value applies only to the related frame if CAN FD frame support is enabled.	<ul style="list-style-type: none"> <li>▪ 0: Bit rate switch disabled, i.e., the data phase of the frame is transmitted with the bit rate that is specified for the Baud rate property of the related CAN communication cluster.</li> <li>▪ 1: Bit rate switch enabled, i.e., the data phase of the frame is transmitted with the bit rate that is specified for the Data phase baud rate property of the related CAN communication cluster.</li> </ul>

The default values of the function ports are derived from the related frame settings that are specified in the communication matrix. Via the Initial value property of each function port, you can specify a user-defined initial function port value. To access the function port values at run time, you must enable model access or test automation support for the function ports. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

For the Identifier function port, you can additionally specify user-defined saturation values to limit the minimum and maximum values of the function port. For more information, refer to [Specifying saturation values for function imports](#) on page 123.

#### Accessing the frame state

When you add the Frame Access feature to an RX PDU, a State function port is available that provides the state of the frame as follows:

- 0: The frame is not received in the current sampling step. This is the default state of the function port.
- 1: The frame is received in the current sampling step.

Via the function port's Initial value property, you can specify a user-defined initial value in the range 0 ... 255. To access the function port value at run time, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

#### Restrictions for using the Frame Access feature

If you use the Frame Access feature, the following restrictions apply:

- You cannot use the Frame Access feature with any other PDU- or ISignal-related bus configuration feature in parallel. This applies to the PDU to which you add the Frame Access feature and to its included elements, i.e.:
  - Contained IPDUs if you add the feature to a container IPDU.
  - ISignal PDUs that are configured as the static or dynamic part if you add the feature to a multiplexed IPDU.
  - PDUs that are used as authentic IPDUs if you add the feature to a secured IPDU.
  - ISignals that are directly included in the PDU or in any of its included PDUs.

If you use the Frame Access feature with any other PDU- or ISignal-related feature in parallel, conflicts occur.

- Cyclic timings that are specified in the communication matrix are ignored. Instead, the transmission of the PDU is only triggered according to the settings specified for the Frame Access feature.
- Bit patterns for unused bits that are specified in the communication matrix are ignored. Instead, 0 is used as bit pattern for unused bits. This applies to the PDU to which you add the Frame Access feature and to its included PDUs.

**Related topics****References**

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)  
[Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Manipulating the Payload Length of CAN Frames

**Introduction**

You can manipulate the payload length of CAN frames that are assigned to the Manipulation part of a bus configuration.

**Conditions for manipulating the payload length of CAN frames**

To manipulate the payload length of a CAN frame, you must add the Frame Length feature to a PDU that is included in the frame. You can add the Frame Length feature to a PDU only if the PDU and the related frame fulfill all of the following conditions:

- The PDU is a CAN PDU that is directly included in one frame. Therefore, the PDU can be one of the following types:
  - [Container IPDU](#) that is not included in a secured IPDU
  - [Multiplexed IPDU](#) that is neither included in a secured IPDU, nor configured as a contained IPDU
  - [Secured IPDU](#) that is not configured as a contained IPDU
  - [Basic PDU](#) that is neither configured as a contained IPDU, nor as static part or dynamic part of a multiplexed IPDU, and not included in a secured IPDU
- The PDU has exactly one frame triggering for the channel of the communication cluster for which the PDU is assigned to the Manipulation part.
- The related frame has exactly one PDU-to-frame mapping.

**Effects of adding the Frame Length feature**

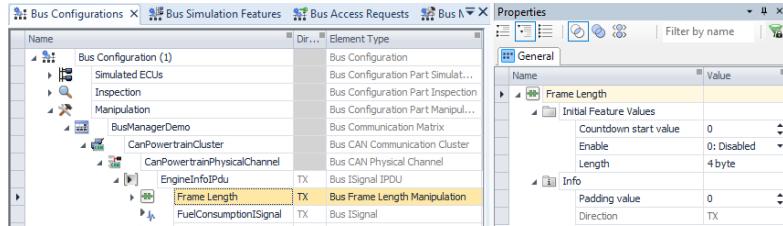
Adding the Frame Length feature to a PDU has the following effects:

- The following properties are available:
  - A Countdown start value and an Enable property that let you configure the run-time behavior for manipulating the frame length.

- A Length and a Padding value property that let you specify an initial payload length and a value for padding bytes, respectively.

You can access the properties in following ways:

- You can access all properties in the Properties Browser when you select the Frame Length node in the Bus Configurations table.



- You can also access the Countdown start value, Length, and Padding value properties in the Bus Manipulation Features table.

Name	Element Type	Bus Configuration	Bus Configuration Part Simul...	Bus Configuration Part Inspection	Bus Configuration Part Manipul...	Bus Communication Matrix	Bus CAN Communication Cluster	Bus CAN Physical Channel	Bus Frame Length Manipulation
Frame Length	TX	Bus ISignal IPDU							

- The following function ports are available:
  - Current Countdown Value function port
  - Countdown Start Value function port
  - Enable function port
  - Enable State function port
  - Length function port

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

EngineInfoIPdu	TX	Bus ISignal IPDU
Frame Length	TX	Bus Frame Length Manipulation
EngineInfoFrame Current Countdown Value		
EngineInfoFrame Countdown Start Value		
EngineInfoFrame Enable		
EngineInfoFrame Enable State		
EngineInfoFrame Length		

### Configuring the run-time behavior for manipulating the payload length

As long as the Frame Length feature is enabled at run time, the payload length of the frame is manipulated.

You can enable the feature via the Enable property or the Initial value and Initial substitute value properties of the Enable function port as follows:

- Set the related property to 1: Permanently enabled.  
The length of the frame is manipulated until you disable the feature explicitly, e.g., via experiment software.
- Set the related property to 2: Temporarily enabled.  
The length of the frame is manipulated for a defined number of times. You can specify the number of times via the Countdown start value property or the Initial value and Initial substitute value properties of the Countdown Start Value function port.

You can access the current enable state and countdown value via the Enable State and Current Countdown Value function ports, respectively. For more information, refer to [Basics on Bus Manipulation Features](#) on page 131.

By default, you can change the specified settings via experiment software at run time. If you want to change the settings via a behavior model or provide function port values to a behavior model, you must enable model access for the related function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

#### Specifying payload lengths

When you add the Frame Length feature to a PDU, the initial payload length is derived from the frame's payload length that is specified in the communication matrix. Via the Length property, you can specify a user-defined payload length in the following range:

- Classic CAN frames: 0 byte ... 8 bytes
- CAN FD frames: 0 byte ... 64 bytes

The specified value applies to the Initial value and Initial substitute value properties of the Length function port automatically.

By default, you can change the specified length via experiment software at run time. If you want to change the length via a behavior model, you must enable model access for the Length function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

#### Specifying a padding value for added bytes

If you extend the length of a frame via the Frame Length feature, the added bytes are filled with a padding value. The default padding value depends on the specifications in the communication matrix as follows:

- If the communication matrix specifies a bit pattern for unused bits of the related PDU, this value is used as padding value.
- If the communication matrix does not specify a bit pattern for unused bits of the related PDU, 0 is used as padding value.

Via the Padding value property, you can specify a user-defined padding value in the range 0 ... 255. You cannot change the specified padding value during run time.

##### Note

The value of the Padding value property does not apply to padding bytes that are used to extend the frame length to a valid CAN FD length. Padding bytes are always filled with 255.

For example, a frame has a length of 8 bytes with a value of 125 each. Via the Frame Length feature, you add 2 bytes and specify a Padding value of 55. The next valid frame length according to CAN FD is 12 bytes. This results in the following frame value:

Byte	1	2	3	4	5	6	7	8	9	10	11	12
Value	125	125	125	125	125	125	125	125	55	55	255	255

For more information on valid CAN FD frame lengths, refer to [CAN FD protocol](#) on page 47.

<b>Specifying saturation values</b>	For the Countdown Start Value, Enable, and Length function ports, you can specify user-defined saturation values to limit the minimum and maximum values of the function ports. If you want to do this, you must first set the bus configuration's Saturation usage property to User min/max value. Refer to <a href="#">Specifying saturation values for function imports</a> on page 123.
-------------------------------------	---

**Related topics****References**

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

[Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Suspending the Transmission of Frames

**Introduction**

You can suspend the transmission of frames that are assigned to the Manipulation part of a bus configuration.

**Conditions for suspending the transmission of frames**

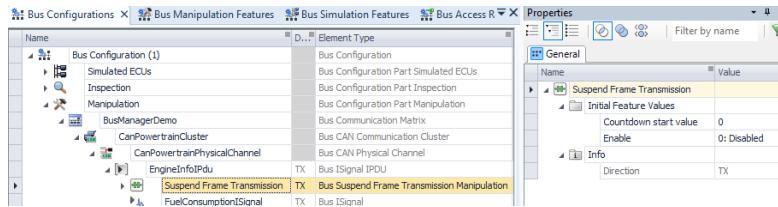
To suspend the transmission of a frame, you must add the Suspend Frame Transmission feature to a PDU that is included in the frame. You can add the Suspend Frame Transmission feature to a PDU only if the PDU and the related frame fulfill all of the following conditions:

- The PDU is directly included in one frame. Therefore, the PDU can be one of the following types:
  - [Container IPDU](#) that is not included in a secured IPDU
  - [Multiplexed IPDU](#) that is neither included in a secured IPDU, nor configured as a contained IPDU
  - [Secured IPDU](#) that is not configured as a contained IPDU
  - [Basic PDU](#) that is neither configured as a contained IPDU, nor as a static part or dynamic part of a multiplexed IPDU, and not included in a secured IPDU
- The PDU has exactly one frame triggering for the channel of the communication cluster for which the PDU is assigned to the Manipulation part.
- The related frame has exactly one PDU-to-frame mapping.

### Effects of adding the Suspend Frame Transmission feature

Adding the Suspend Frame Transmission feature to a PDU has the following effects:

- A Countdown start value and an Enable property are available that let you configure the run-time behavior for suspending the frame transmission. You can access the properties in following ways:
- You can access both properties in the Properties Browser when you select the Suspend Frame Transmission node in the Bus Configurations table.



- You can also access the Countdown start value property in the Bus Manipulation Features table.

PDU Features							
Name	Element Type	...	Bus Config...	Communica...	Communi...	Countdown Start Value	Suspend Frame Transmission
EngineInfoIPdu	Bus ISignal IPDU	TX	Bus Config...	Bus Manage...	CanPower...	0	Enabled
PowerStatusIPdu	Bus ISignal IPDU	TX	Bus Config...	Bus Manage...	CanPower...		Disabled
GearBoxInfoIPdu	Bus ISignal IPDU	TX	Bus Config...	Bus Manage...	CanPower...		Disabled

- The following function ports are available:
  - Current Countdown Value function port
  - Countdown Start Value function port
  - Enable function port
  - Enable State function port

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

EngineInfoIPdu	TX	Bus ISignal IPDU
Suspend Frame Transmission	TX	Bus Suspend Frame Transmission Manipulation
EngineInfoFrame Current Countdown Value		
EngineInfoFrame Countdown Start Value		
EngineInfoFrame Enable		
EngineInfoFrame Enable State		

### Configuring the run-time behavior for suspending the frame transmission

As long as the Suspend Frame Transmission feature is enabled at run time, the transmission of the frame is suspended.

You can enable the feature via the Enable property or the Initial value and Initial substitute value properties of the Enable function port as follows:

- Set the related property to 1: Permanently enabled.  
The transmission of the frame is suspended until you disable the feature explicitly, e.g., via experiment software.
- Set the related property to 2: Temporarily enabled.  
The transmission of the frame is suspended for a defined number of times. You can specify the number of times via the Countdown start value property or the Initial value and Initial substitute value properties of the Countdown Start Value function port.

You can access the current enable state and countdown value via the Enable State and Current Countdown Value function ports, respectively. For more information, refer to [Basics on Bus Manipulation Features](#) on page 131.

By default, you can change the specified settings via experiment software at run time. If you want to change the settings via a behavior model or provide function port values to a behavior model, you must enable model access for the related function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

---

#### Effects of suspending the transmission of a frame

Suspending the transmission of a frame has the following effects on the active bus communication:

- If you suspend the transmission of a frame while the frame is being transmitted, the transmission is not interrupted. The transmission of the frame is suspended after the frame is completely transmitted.
- If you suspend the transmission of the frame, the frame is not queued, i.e., the suspended transmissions are discarded and lost. The suspended transmissions are not processed when the transmission of the frame is enabled again. This applies to all suspended transmission, regardless of whether they are cyclic transmissions or triggered transmission.

---

#### Specifying saturation values

For the Countdown Start Value and Enable function ports, you can specify user-defined saturation values to limit the minimum and maximum values of the function ports. If you want to do this, you must first set the bus configuration's Saturation usage property to User min/max value. Refer to [Specifying saturation values for function imports](#) on page 123.

---

#### Related topics

#### References

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

[Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

# Bus Configuration Features Available for Communication Controllers

## Where to go from here

## Information in this section

<a href="#">Enabling and Disabling Communication Controllers.....</a>	203
<a href="#">Sending Wake-Up Signals on a LIN Bus.....</a>	205
<a href="#">Working with LIN Schedule Tables.....</a>	207
<a href="#">Enabling and Disabling J1939 Network Management.....</a>	211

## Enabling and Disabling Communication Controllers

### Introduction

You can enable and disable communication controllers that are assigned to the Simulated ECUs part of a bus configuration.

### Conditions for enabling and disabling communication controllers

Each [network node](#) of a communication cluster has one communication controller (CAN communication controller, [LIN slave](#) communication controller, or [LIN master](#) communication controller).

To enable or disable a communication controller, you must add the Communication Controller Enable feature to a selected communication controller.

### Effects of adding the Communication Controller Enable feature

When you add the Communication Controller Enable feature, the following function ports are available for the communication controller:

- Communication Controller Enable function port  
This function port lets you enable and disable the communication controller.
- Communication Controller Enable State function port  
This function port provides the current state of the communication controller (e.g., to a mapped behavior model).

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

	CanEngineCommController	Bus CAN Communication Controller
	Communication Controller Enable	Bus Communication Controller Enable Access
	CanEngineCommController Enable	
	CanEngineCommController Enable State	

---

**Specifying the enable state**

The Communication Controller Enable function port lets you specify the enable state of the communication controller as follows:

- False (0): The communication controller is disabled.
- True (1): The communication controller is enabled. This is the default state.

The Communication Controller Enable State function port provides the currently active enable state.

Via the Initial value properties of the function ports, you can change the default enable state. To change the enable state of the communication controller at run time and to access the function port values, you must enable model access or test automation support for the function ports. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

---

**Effects of disabling a communication controller**

**General effects on the bus communication** When you disable a communication controller, you stop the communication of the related network node. The network node does not transmit frames on the bus and received frames are not evaluated. This lets you, for example, exclude the network node from being simulated without adapting the configured bus communication.

If you disable a LIN master communication controller, the LIN master is disabled and therefore the communication of the entire LIN cluster stops. Disabling a LIN slave or CAN communication controller does not directly affect other network nodes.

**Effects on the currently active bus communication** The following effects apply to the currently active bus communication:

- If you disable a communication controller while a frame is being transmitted/received, the frame is not interrupted. The communication controller is disabled after the frame is completely transmitted/received. This behavior also applies to LIN master headers and slave responses.
- If the transmission of a frame is triggered while the related communication controller is disabled, the trigger is discarded and lost, i.e., the trigger is not processed when the communication controller is active again.
- Disabling a LIN master communication controller has the following effects on the LIN schedule tables:
  - Disabling the communication controllers suspends the active LIN schedule table at the end of the currently active slot.
  - If a request to switch the active LIN schedule table occurs while the communication controller is disabled, the request is discarded and lost, i.e., the request is not processed when the communication controller is active again.

For more information on working with LIN schedule tables, refer to [Working with LIN Schedule Tables](#) on page 207.

**Effects of re-enabling communication controllers**

Re-enabling a communication controller at run time (i.e., switching the state from False to True) has the following effects:

Element	Available for	Effects
End-to-end protection counters	End-to-end-protected ISignal groups (according to the corresponding end-to-end protection profile)	The counters of end-to-end-protected ISignal groups are reset to 0.
Counter signals	ISignals with enabled Counter Signal feature	Counter signals are reset to the specified initial counter value.
Cyclic timings	PDUs	The cyclic timings of PDUs are reset to the values specified for the time period and the time offset.
Container IPDUs	CAN communication	<ul style="list-style-type: none"> <li>▪ The container timeout of TX container IPDUs is reset to its initial state.</li> <li>▪ The queues of contained IPDUs in container IPDUs with a dynamic container layout are cleared, i.e., no contained IPDUs are queued.</li> <li>▪ The send buffer of contained IPDUs in container IPDUs with a static container layout is cleared, i.e., no contained IPDUs are buffered.</li> </ul>
TX and RX global time domains	Network nodes that are configured as time masters or time slaves in global time synchronization scenarios	<ul style="list-style-type: none"> <li>▪ The counters of time masters and time slaves are reset to 0.</li> <li>▪ The timings of time masters are reset to their initial states.</li> <li>▪ The message queues are cleared, i.e., follow-up (FUP) messages are lost if their related synchronization (SYNC) messages were transmitted before the communication controller was disabled.</li> </ul>
Transport protocol address	Communication controllers with enabled J1939 Network Management Enable feature	<ul style="list-style-type: none"> <li>▪ Only if the J1939 Network Management Enable feature is set to the 2: Dynamic address handling enable state: The transport protocol address of the communication controller is reset to the address value that is specified in the communication matrix.</li> <li>▪ The communication controller sends an address claim on the CAN bus.</li> </ul>
LIN schedule tables	LIN masters	<p>The active LIN schedule table and the Index function port are not reset, i.e.:</p> <ul style="list-style-type: none"> <li>▪ The LIN schedule table is continued with the next pending slot, i.e., no schedule table slot is lost or sent twice.</li> <li>▪ The Index function port keeps its last value.</li> </ul>

**Related topics****References**

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

[Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Sending Wake-Up Signals on a LIN Bus

**Introduction**

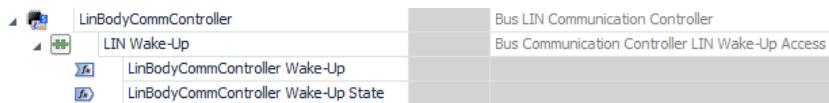
You can send wake-up signals via LIN communication controllers that are assigned to the Simulated ECUs part of a bus configuration.

**Conditions for sending wake-up signals on a LIN bus** To send a wake-up signal on a LIN bus, you must add the LIN Wake-Up feature to a LIN communication controller. You can add the feature to LIN master and LIN slave communication controllers.

**Effects of adding the LIN Wake-Up feature** When you add the LIN Wake-Up feature, the following function ports are available for the communication controller:

- **Wake-Up function port**  
This function port lets you specify whether the communication controller sends wake-up signals on the bus.
- **Wake-Up State function port**  
This function port indicates whether the communication controller has detected a wake-up signal on the bus.

The following illustration is an example of the function ports as displayed in the Bus Configurations table:



#### Sending wake-up signals

The Wake-Up function port lets you specify whether the communication controller sends wake-up signals:

- **False (0):** No wake-up signal is sent. This is the default state.
- **True (1):** A wake-up signal is sent.

Via the function port's **Initial value** property, you can change the default state. To change the state at run time, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

#### Processing of wake-up signals

Wake-up signals are only processed if the LIN bus is in sleep mode. If a wake-up signal is sent while the LIN bus is in any other mode (e.g., active mode or initialization mode), the wake-up signal is ignored by the bus members.

The LIN bus switches to sleep mode if it is inactive for at least 4 seconds. The LIN bus is inactive if there is no bus communication, no wake-up signals are sent, etc.

If the LIN communication is inactive directly after an [executable application](#) starts, the LIN bus cannot switch to sleep mode. If a wake-up signal is sent in this case, it is ignored by the bus members. The LIN bus can switch to sleep mode only after there was LIN bus communication (e.g., a frame was sent on the bus). If then the bus is inactive for at least 4 seconds, wake-up signals can be processed by the bus members.

**Wake-up signals detected by the communication controller**

The value of the Wake-Up State function port indicates whether a wake-up signal is detected by the communication controller:

- 0: No wake-up signal is detected, e.g., because no wake-up signal is sent on the bus or the communication controller is already active.
- 1: A wake-up signal is detected.

By default, the initial function port value is 0. Via the function port's Initial value property, you can specify a user-defined initial value in the range 0 ... 255. To access the function port value at run time, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

**Restrictions for using the LIN Wake-Up feature**

If you use the LIN Wake-Up feature, the following restrictions apply:

- Communication controllers that are disabled via the Communication Controller Enable feature cannot send or receive wake-up signals. Before you have such communication controllers send or receive wake-up signals, you must enable the controllers via the Communication Controller Enable feature.
- If you set the Initial value or Initial substitute value property of the Wake-Up function port to True, a wake-up signal is sent directly after an executable application starts. However, this wake-up signal cannot be processed by the bus members. Directly after starting an executable application the LIN bus is never in sleep mode, even if no bus communication is active. Refer to [Processing of wake-up signals](#) on page 206.

**Related topics****References**

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

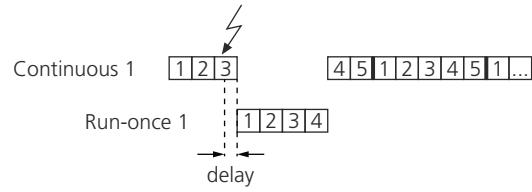
[Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Working with LIN Schedule Tables

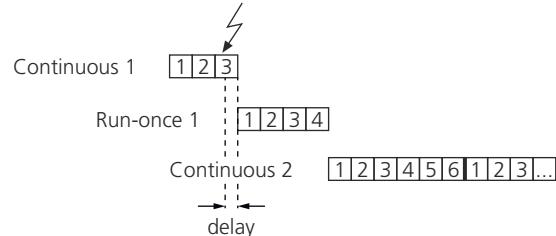
**Introduction**

Each [LIN master](#) provides schedule tables that are used to regulate the communication of a LIN bus. For LIN master communication controllers that are assigned to the Simulated ECUs part of a bus configuration, you can specify an initial schedule table and change the active schedule table during run time.

<b>Basics on schedule tables</b>	<p>Schedule tables define the sequence in which the LIN master transmits frame headers on the bus and requests the answers from the connected <a href="#">LIN slaves</a>. A <a href="#">communication matrix</a> can define one or more schedule tables for a LIN master. In general, there are two types of schedule tables:</p> <ul style="list-style-type: none"> <li>▪ Continuous schedule tables</li> </ul> <p>A continuous schedule table is repeated until it is stopped by another continuous schedule table or interrupted by a run-once schedule table.</p> <ul style="list-style-type: none"> <li>▪ Run-once schedule tables</li> </ul> <p>A run-once schedule table is processed only once and can be neither stopped nor interrupted. If a run-once schedule table ends and no other schedule table is pending, the bus communication stops.</p>
<b>Switch behavior between different schedule tables</b>	<p>Depending on the schedule table type, a switch request to start a new schedule table has different effects on the currently active schedule table:</p> <ul style="list-style-type: none"> <li>▪ Active frame slots of continuous schedule tables cannot be stopped. If a switch request is sent while a frame slot of a continuous schedule table is active, the switch is delayed. The new schedule table starts at the end of this frame slot.</li> </ul> <ul style="list-style-type: none"> <li>▪ When a continuous schedule table is stopped by another continuous schedule table, the new continuous schedule table is processed until it is stopped or interrupted itself.</li> <li>▪ When a continuous schedule table is interrupted by a run-once schedule table, the run-once schedule table is processed until it ends: <ul style="list-style-type: none"> <li>▪ If no other schedule table is pending in the request queue, the original continuous schedule table is processed again. Depending on the schedule table's resume position, which is defined in the communication matrix, the schedule table starts either from the beginning or from the point where it was interrupted. The following illustration is an example for the latter case.</li> </ul> </li> </ul>

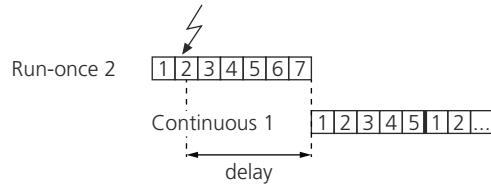


- If another schedule table is pending in the request queue, this schedule table starts.



The request queue can store up to 5 schedule tables. When the request queue is full, no further schedule tables can be added. These schedule tables are lost.

- If a switch request is sent while a run-once schedule table is active, the switch is delayed until the run-once schedule table finished.

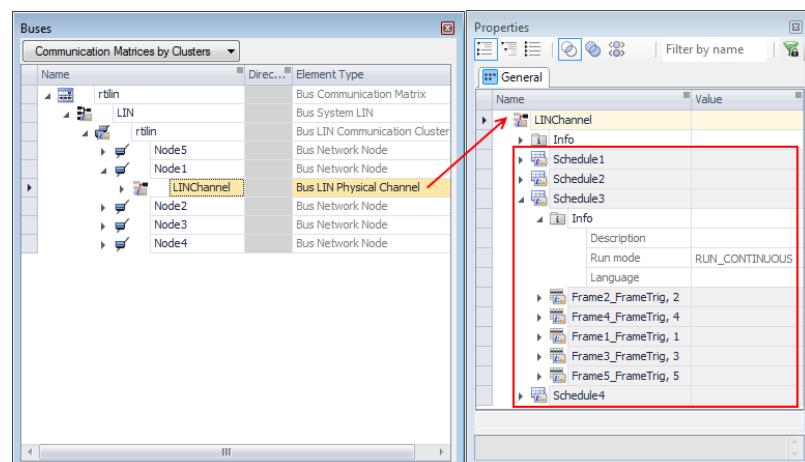


## Accessing schedule tables via the Bus Manager

You can access schedule tables in two ways:

- Via the LIN bus channel

When you select the LIN bus channel (e.g., in the Communication Matrices by Clusters view of the Buses Browser), the Properties Browser displays all the schedule tables that are defined for the LIN master.



- Via PDUs

When you select a LIN PDU (e.g., in the Buses Browser), the Bus Communication page of the Properties Browser displays all the schedule tables that contain a frame triggering of the selected PDU.

### Working with the LIN Schedule Table feature

To each LIN master communication controller that is assigned to the Simulated ECUs part of a bus configuration, the LIN Schedule Table feature is added automatically. Therefore, one LIN Schedule Index function port is available for each LIN master communication controller. Via this function port, you can specify an initial schedule table for the related LIN master. If you enable model access and/or test automation support, you can change the active LIN schedule table during run time.

#### Note

By default, no initial schedule table is specified. Therefore, LIN communication is disabled. Since model access and test automation support of the function port are disabled by default as well, you cannot select a schedule table during run time. To enable LIN communication, you must either specify an initial schedule table or enable model access and/or test automation support for the function port.

For more information on enabling model access and test automation support, refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

The following illustration is an example of the function port as displayed in the Bus Configurations table:

	LinBodyCommController		Bus LIN Communication Controller
	LIN Schedule Table	TX	Bus Communication Controller LIN Schedule Table Access
	LinBodyCommController Schedule Index		

### Notes on specifying an initial schedule table

The Initial value property of a LIN Schedule Index function port references the position of a schedule table in the communication matrix: For example, an Initial value of 2 references the second schedule table in the communication matrix specified for the LIN master. This schedule table is used as the initial schedule table.

#### Tip

By default, the Properties Browser displays the schedule tables in the order in which they are specified in the communication matrix when you select a LIN bus channel or a LIN PDU. If you sorted the tree view of the Properties Browser in ascending or descending order, clear the sorting to display the schedule tables in the order in which they are specified in the communication matrix.

If the Initial value is set to 0, no initial schedule table is used and no frames are transmitted or received directly after the executable application is started. If the

Initial value is higher than the number of schedule tables defined for the LIN master, a conflict occurs.

---

<b>Limitations</b>	When you work with LIN schedule tables, some limitations apply. For details, refer to <a href="#">Limitations for LIN Communication</a> on page 325.
--------------------	--

---

Related topics	References
Function Import Properties (Bus Configuration) (ConfigurationDesk Function Block Properties 	

## Enabling and Disabling J1939 Network Management

---

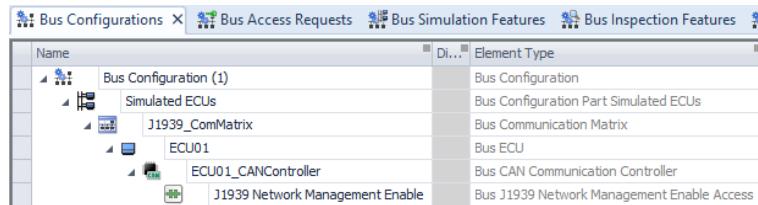
<b>Introduction</b>	You can enable, disable, and specify the enable state of J1939 network management for CAN communication controllers that are assigned to the Simulated ECUs part of a bus configuration. If J1939 network management is enabled, the related <a href="#">network node</a>  participates in the J1939 address claiming procedure. This procedure ensures that a network node participates in J1939 communication only if its transport protocol address is unique in the communication cluster. For more information on the address claiming procedure, refer to <a href="#">J1939 protocol</a> on page 48.
---------------------	---

---

<b>Conditions for enabling and disabling J1939 network management</b>	To enable or disable J1939 network management for a CAN communication controller, you must add the J1939 Network Management Enable feature to the communication controller. You can add the feature to any CAN communication controller that participates in J1939 communication.
---	---

---

<b>Effects of adding the J1939 Network Management Enable feature</b>	When you add the J1939 Network Management Enable feature, a J1939 Network Management Enable node is added to the communication controller in the Bus Configurations table, as shown in the following example.
--	---



The node provides an **Enable** property, which lets you specify the network management enable state as follows:

- **0: Disabled:** J1939 network management is disabled.
- **1: Static address handling:** Static J1939 network management is enabled. This is the default state.
- **2: Dynamic address handling:** Dynamic J1939 network management is enabled.

You cannot change the specified enable state at run time.

The J1939 Network Management Enable feature does not provide any settings that can be configured at run time. Therefore, no function ports are available for this feature.

---

#### **Effects of disabling J1939 network management**

If you set the **Enable** property to **0: Disabled**, the network node neither transmits address claims nor responds to received address claims at run time. This results in the following:

- The network node participates in the J1939 communication regardless of whether its transport protocol address is unique in the communication cluster.
- Other members of the communication cluster are not informed about the used transport protocol address. Therefore, they cannot detect whether there is an address conflict with their own transport protocol address.

---

#### **Effects of enabling J1939 network management**

If you set the **Enable** property to **1: Static address handling** or **2: Dynamic address handling**, the network node transmits an address claim in the following situations at run time:

- At application start if the bus configuration and the communication controller are enabled.
- The state of the bus configuration changes, i.e., it is set to enabled via the **Bus Configuration Enable** feature, while the communication controller is enabled.
- The state of the communication controller changes, i.e., it is set to enabled via the **Communication Controller Enable** feature, while the bus configuration is enabled.

However, if the **2: Dynamic address handling** enable state is selected for a network node, the following applies: Before the network node transmits the address claim, its transport protocol address is reset to the address value that is specified in the communication matrix. This way, the network node always tries to claim its initial transport protocol address.

The reactions on received address claims differ for the **1: Static address handling** and **2: Dynamic address handling** enable states:

<b>Enable State</b>	<b>Reaction on Received Address Claims</b>
1: Static address handling	<ul style="list-style-type: none"> <li>▪ If the network node receives an address claim that conflicts with its own transport protocol address, the network node responds as follows:</li> </ul>

Enable State	Reaction on Received Address Claims
	<ul style="list-style-type: none"> <li>▪ If the network node has a higher priority, it transmits an address claim itself and participates in the J1939 communication. According to J1939-81, the network node that transmitted the conflicting address claim must either claim another address or stop its J1939 communication.</li> <li>▪ If the network node has a lower priority, it transmits a <i>cannot claim address</i> message and stops its J1939 communication. This applies even if its <b>arbitrary address capable</b> parameter is set to <b>1</b>.</li> <li>▪ If the network node receives address claims indicating that other network nodes have claimed new transport protocol addresses, the network node does not adjust its J1939 communication: It transmits data to and receives data from the transport protocol addresses that are specified in the communication matrix. This applies regardless of whether the addresses are used by the network nodes specified by the communication matrix as the receiving and transmitting network nodes.</li> </ul>
2: Dynamic address handling	<ul style="list-style-type: none"> <li>▪ If the network node receives an address claim that conflicts with its own transport protocol address, the network node responds as follows: <ul style="list-style-type: none"> <li>▪ If the network node has a higher priority, it transmits an address claim itself and participates in the J1939 communication. According to J1939-81, the network node that transmitted the conflicting address claim must either claim another address or stop its J1939 communication.</li> <li>▪ If the network node has a lower priority, the response depends on the value of the <b>arbitrary address capable</b> parameter: <ul style="list-style-type: none"> <li>▪ <b>Arbitrary address capable</b> set to <b>0</b>: The network node transmits a <i>cannot claim address</i> message and stops its J1939 communication.</li> <li>▪ <b>Arbitrary address capable</b> set to <b>1</b>: The network node claims a new transport protocol address in the range 128 ... 247 and communicates the claimed address by sending an address claim on the bus. The network node transmits a <i>cannot claim address</i> message and stops its J1939 communication only if the network node cannot claim a new address, e.g., because all addresses in the value range are used by network nodes that have a higher priority.</li> </ul> </li> <li>▪ If the network node receives address claims indicating that other network nodes have claimed new transport protocol addresses, the network node adjusts its J1939 communication: It transmits data to and receives data from the transport protocol addresses that are used by the network nodes the communication matrix specifies as the receiving and transmitting network nodes.</li> </ul> </li> </ul>

When the enable state is set to 2: Dynamic address handling, the network node additionally claims a new address in the following situation:

- The **arbitrary address capable** parameter is set to **1**.
- The initial transport protocol address is set to 254.

## Related topics

### Basics

[Limitations for CAN Communication.....](#) 323

# Bus Configuration Features Available for Global Time Domains

## Where to go from here

## Information in this section

- |   |     |
|---|-----|
| Controlling the Timing of Time Synchronization.....               | 214 |
| Accessing the Time Base Data of Time Masters and Time Slaves..... | 216 |
| Accessing Validity Checks for Time Synchronization Messages.....  | 218 |

## Controlling the Timing of Time Synchronization

### Introduction

You can control the timing of the time synchronization of time masters that are assigned to the Simulated ECUs part of a bus configuration.

### Conditions for controlling the timing of time synchronization

To control the timing of the time synchronization of a time master, you must add the GTS Transmission Control feature to the related TX global time domain. When you do this, you can specify the synchronization period or trigger the time synchronization asynchronously.

### Effects of adding the GTS Transmission Control feature

When you add the GTS Transmission Control feature to a TX global time domain, the following function ports are available:

- Period function port  
This function port lets you specify the synchronization period for cyclic time synchronization.
- Trigger function port  
This function port lets you trigger the time synchronization asynchronously.

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

	GTS_CanFrame_TD0_PDU	TX	Bus General-Purpose PDU
	GTD0_CAN	TX	Bus Global Time Domain
	GTS Transmission Control	TX	Bus GTS Transmission Control Access
	GTD0_CAN Period		
	GTD0_CAN Trigger		

### Specifying the synchronization period

The synchronization period determines the time that is available for transmitting a synchronization message and its related follow-up message. For more information, refer to [Run-time behavior regarding synchronization periods](#) on page 90.

The Period function port lets you specify the synchronization period in seconds for cyclic time synchronization. By default, the initial function port value is the synchronization period that is specified in the communication matrix. If the communication matrix does not specify a synchronization period, 0 is used as the initial function port value. Via the function port's Initial value property, you can specify a user-defined initial synchronization period.

#### Note

If the period is 0, cyclic time synchronization is disabled.

To change the synchronization period at run time, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

The actual timing of the time synchronization depends on the related Bus Configuration task. For more information on the Bus Configuration task, refer to [Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager](#) on page 94.

#### Specifying saturation values for the synchronization period

For the Period function port, you can specify user-defined saturation values to limit the minimum and maximum values of the synchronization period. If you want to do this, you must first set the bus configuration's Saturation usage property to User min/max value. Refer to [Specifying saturation values for function imports](#) on page 123.

#### Triggering time synchronization asynchronously

The Trigger function port lets you trigger time synchronization, e.g., asynchronously to cyclic time synchronization. Each time a rising edge is detected, i.e., the function port value changes from 0 to 1, a synchronization message and its related follow-up message are transmitted once.

The default settings of the Trigger function port prohibit triggering time synchronization asynchronously during run time. To trigger time synchronization during run time, you must at least enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

By default, the initial function port value is 0. Via the function port's Initial value property, you can specify the following values:

- False (0): During run time, the first change of the function port value results in a rising edge. Therefore, time synchronization is triggered asynchronously with the first value change.

#### Tip

If the function port value was changed via experiment software during run time, the function port value is automatically reset to 0 after the time synchronization was triggered.

- True (1): During run time, the first change of the function port value results in a falling edge. Therefore, time synchronization is not triggered asynchronously with the first value change but with the second value change.

The actual timing of the asynchronous time synchronization depends on the related Bus Configuration task. For more information on the Bus Configuration task, refer to [Basics on Tasks, Events, and Runnable Functions Provided by the Bus Manager](#) on page 94.

**Related topics****Basics**

[Implementing Global Time Synchronization in Executable Applications.....](#) 88

**References**

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Accessing the Time Base Data of Time Masters and Time Slaves

**Introduction**

You can access the time base data of time masters and time slaves that are assigned to the Simulated ECUs part of a bus configuration.

**Conditions for accessing the time base data of time masters and time slaves**

To access the time base data of a time master or a time slave, you must add the GTS Time Base Data feature to the related TX or RX global time domain.

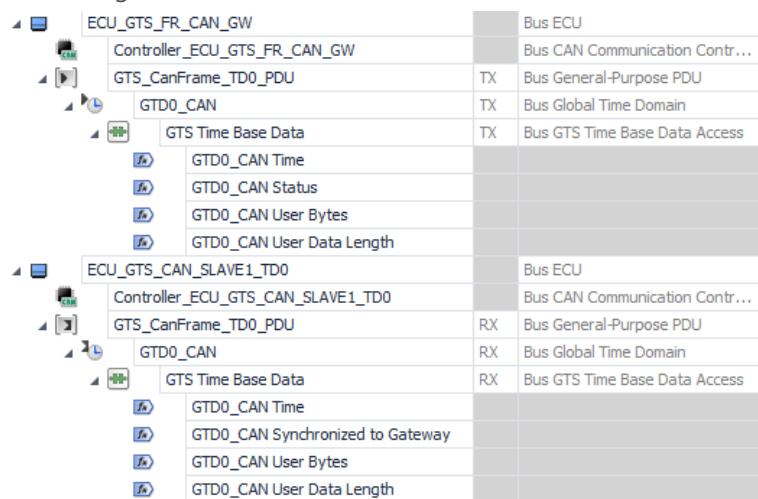
**Effects of adding the GTS Time Base Data feature**

When you add the GTS Time Base Data feature to a global time domain, the following function ports are available:

Function Port	Available for	Purpose
Time	▪ Time master ▪ Time slave	Provides the time when the time synchronization message is transmitted by the time master or received by the time slave, respectively. This is the time to which the time master and time slave are synchronized. The function port provides the time information of the synchronization (SYNC) message and its related follow-up (FUP) message in seconds.
Status	Time master	Provides the synchronization status of the time base. Refer to <a href="#">Status information provided by the Status function port</a> on page 217.
Synchronized to Gateway	Time slave	Provides the received synchronization state of the time base, i.e, the value of the SGW bit of the follow-up message: ▪ 0: The time base is directly synchronized with the global time master.

Function Port	Available for	Purpose
		<ul style="list-style-type: none"> <li>1: The time base is synchronized with a time gateway, i.e., it is not directly synchronized with the global time master. Instead, it is synchronized with a time master subordinate to the global time master.</li> </ul>
User Bytes	<ul style="list-style-type: none"> <li>Time master</li> <li>Time slave</li> </ul>	<p>Provides the values of the user bytes.</p> <p>The function port always provides three bytes, regardless of the actual number of used user bytes. The actual number of used user bytes is provided by the User Data Length function port.</p>
User Data Length	<ul style="list-style-type: none"> <li>Time master</li> <li>Time slave</li> </ul>	Provides the actual number of used user bytes.

The following illustration is an example of the function ports as displayed in the Bus Configurations table:



#### Specifying initial function port values and accessing the time base data at run time

By default, the value of all function ports is 0. Via the Initial value property of the each function port, you can specify a user-defined initial function port value. To access the time base data that is provided by the function ports at run time, you must enable model access or test automation support for the function ports. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

#### Status information provided by the Status function port

The Status function port that is available for time masters provides the following status information:

Bit Position	Bit Name	Bit Values
Bit 0 (LSB)	TIMEOUT	<ul style="list-style-type: none"> <li>0: No synchronization timeout.</li> <li>1: Synchronization timeout.</li> </ul> <p>The time base was not synchronized for a longer period than specified for the Sync loss timeout parameter in the communication matrix.</p>
Bit 2	SYNC_TO_GATEWAY (GTW)	<ul style="list-style-type: none"> <li>0: The time base is directly synchronized with the global time master.</li> <li>1: The time base is synchronized with a time gateway, i.e., it is not directly synchronized with the global time master. Instead, it is synchronized with a time master subordinate of the global time master.</li> </ul>

Bit Position	Bit Name	Bit Values
Bit 3	GLOBAL_TIME_BASE	<ul style="list-style-type: none"> <li>▪ 0: The time base has never been synchronized with the global time master, e.g., because it is based on a local clock.</li> <li>▪ 1: The time base has been synchronized with the global time master at least once.</li> </ul> <p><b>Note</b></p> <p>The time master can transmit time synchronization messages only if this bit is set to 1, i.e., time synchronization with this time base is disabled as long as the bit value is 0.</p>
Bit 4	TIMELEAP_FUTURE	<ul style="list-style-type: none"> <li>▪ 0: No leap into the future within the received time for the time base, i.e., the received time does not exceed the threshold specified by the <b>Time leap future threshold</b> parameter in the communication matrix.</li> <li>▪ 1: Leap into the future, i.e., the received time exceeds the threshold specified by the <b>Time leap future threshold</b> parameter in the communication matrix.</li> </ul> <p>The bit value is reset to 0 after the time has been received correctly for the number of times specified via the <b>Time leap healing counter</b> parameter in the communication matrix.</p>
Bit 5	TIMELEAP_PAST	<ul style="list-style-type: none"> <li>▪ 0: No leap into the past within the received time for the time base, i.e., the received time does not exceed the threshold specified by the <b>Time leap past threshold</b> parameter in the communication matrix.</li> <li>▪ 1: Leap into the past, i.e., the received time exceeds the threshold specified by the <b>Time leap past threshold</b> parameter in the communication matrix.</li> </ul> <p>The bit value is reset to 0 after the time has been received correctly for the number of times specified via the <b>Time leap healing counter</b> parameter in the communication matrix.</p>

All other bits of the Status function port are reserved and set to 0.

## Related topics

### Basics

[Implementing Global Time Synchronization in Executable Applications.....88](#)

### References

[Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties !\[\]\(a871441f7b6f1c1e38b45b15cc279177\_img.jpg\)](#)

## Accessing Validity Checks for Time Synchronization Messages

### Introduction

You can access the validity checks that are performed for time synchronization messages received by time slaves that are assigned to the Simulated ECUs part of a bus configuration.

**Conditions for accessing validity checks for time synchronization messages**

To access the validity checks that are performed for the time synchronization messages received by a time slave, you must add the GTS Validation feature to the related RX global time domain.

**Effects of adding the GTS Validation feature**

When you add the GTS Validation feature to an RX global time domain, the following function ports are available:

- Result function port  
This function port provides the results of the validity checks that are performed for a time synchronization message, i.e., the synchronization (SYNC) message and its related follow-up (FUP) message.
- Partial Validation function port  
This function port lets you exclude certain validity checks from the validation of time synchronization messages.

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

▲ [x] GTS_CanFrame_TD1_PDU	RX	Bus General-Purpose PDU
▲ [b] GTD1_CAN	RX	Bus Global Time Domain
▲ [g] GTS Validation	RX	Bus GTS Validation Access
▲ [f] GTD1_CAN Partial Validation		
▲ [r] GTD1_CAN Result		

**Accessing the results of validity checks**

For each time synchronization message that is received by a time slave, several validity checks are performed. In most cases, the time base of the time slave is not synchronized if a validation check indicates an error. The result of each validity check is written to a specific bit of the Result function port. A bit value of 1 indicates an error.

To access the results of the validity checks, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

The following table provides an overview of the Result function port bits that provide results of validity checks, including the related validity checks. Bits that are not listed in the table are reserved for future use. The value of these bits is always 0. The table also shows which validity checks affect the synchronization of the slave's time base.

Bit Position	Validity Check	Bit Value	Affects Time Base Synchronization
Bit 0	Received two consecutive synchronization messages. The first received synchronization message is ignored.	▪ 0: No ▪ 1: Yes	No
Bit 2	Received synchronization message with wrong message type. Example: A synchronization message is CRC-secured but its message type is 0x10.	▪ 0: No ▪ 1: Yes	Yes
Bit 4	Received synchronization message whose message length is neither 8 bytes nor 16 bytes.	▪ 0: No ▪ 1: Yes	Yes
Bit 5	Received synchronization message whose message length is shorter than 8 bytes.	▪ 0: No	Yes

Bit Position	Validity Check	Bit Value	Affects Time Base Synchronization
		▪ 1: Yes	
Bit 6	Received synchronization message whose message length differs from the message length specified in the communication matrix	▪ 0: No ▪ 1: Yes	Yes
Bit 8	Received synchronization message whose sequence counter is either not incremented or incremented by a value greater than specified via the <code>sequence counter jump width</code> attribute.	▪ 0: No ▪ 1: Yes	Yes
Bit 9	Received synchronization message whose sequence counter was not incremented by 1.	▪ 0: No ▪ 1: Yes	No
Bit 10	Received CRC-secured synchronization message whose <code>CRC validated</code> attribute is set to <code>CRC optional</code> or <code>CRC validated</code> and whose checksum value is incorrect.	▪ 0: No ▪ 1: Yes	Yes
Bit 11	Received CRC-secured synchronization message whose <code>CRC validated</code> attribute is set to <code>CRC optional</code> , but no data IDs are available.	▪ 0: No ▪ 1: Yes	Yes
Bit 16	Received two consecutive follow-up messages.	▪ 0: No ▪ 1: Yes	Yes
Bit 18	Received follow-up message with incorrect message type. Example: A follow-up message is CRC-secured but its message type is 0x18.	▪ 0: No ▪ 1: Yes	Yes
Bit 19	Received follow-up message whose message type does not correspond to the message type of the related synchronization message. Example: A follow-up message is CRC-secured but the related synchronization message is not CRC-secured.	▪ 0: No ▪ 1: Yes	Yes
Bit 20	Received follow-up message whose message length is neither 8 bytes nor 16 bytes.	▪ 0: No ▪ 1: Yes	Yes
Bit 21	Received follow-up message whose message length is shorter than 8 bytes.	▪ 0: No ▪ 1: Yes	Yes
Bit 22	Received follow-up message whose message length differs from the message length specified in the communication matrix.	▪ 0: No ▪ 1: Yes	Yes
Bit 24	Received follow-up message whose sequence counter value differs from the sequence counter value of the related synchronization message.	▪ 0: No ▪ 1: Yes	Yes
Bit 26	Received CRC-secured follow-up message whose <code>CRC validated</code> attribute is set to <code>CRC optional</code> or <code>CRC validated</code> and whose checksum value is incorrect.	▪ 0: No ▪ 1: Yes	Yes
Bit 27	Received CRC-secured follow-up message whose <code>CRC validated</code> attribute is set to <code>CRC optional</code> , but no data IDs are available.	▪ 0: No ▪ 1: Yes	Yes
Bit 28	Received follow-up message too late, i.e., the maximum time span between the receipt of the synchronization message and the follow-up message has elapsed.	▪ 0: No ▪ 1: Yes	Yes

By default, the initial function port value is 0. Via the function port's `Initial value` property, you can specify a user-defined initial value in the range `UInt32min ... UInt32max`. Refer to [Specifying initial values](#) on page 120.

#### Excluding validity checks via partial validation

**Enabling and disabling partial validation** The Partial Validation function port lets you enable and disable partial validation as follows:

- False (0): Partial validation is disabled. This is the default state.
- True (1): Partial validation is enabled.

Via the Initial value property of the function port, you can change the default enable state. To change the enable state at run time, you must enable model access or test automation support. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

**Effects of enabling partial validation** When you enable partial validation, certain validity checks are excluded from the validation of time synchronization messages. The time base of the time slave is synchronized even if any of the excluded validity checks indicates an error.

### Tip

- Validity checks that do not affect the time base synchronization are not affected by partial validation.
- The results of all validity checks are written to the Result function port, regardless of whether they are affected by partial validation.

When you enable partial validation, only the following validity checks are used for validating time synchronization messages:

Validity Check	Bit Position of Result Function Port
Received synchronization message whose message length is shorter than 8 bytes.	Bit 5
Received two consecutive follow-up messages.	Bit 16
Received follow-up message whose message length is shorter than 8 bytes.	Bit 21

---

## Related topics

### Basics

[Implementing Global Time Synchronization in Executable Applications](#).....88

### References

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)  
[Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

# Bus Configuration Features Available for Frame Captures and Frame Gateways

## Where to go from here

## Information in this section

<a href="#">Accessing the Data of Captured Frames.....</a>	222
<a href="#">Specifying the Direction of CAN Frame Gateways.....</a>	226
<a href="#">Controlling Filters of Frame Captures and Frame Gateways.....</a>	227

## Accessing the Data of Captured Frames

### Introduction

You can access the data of frames that are captured by frame captures of the Inspection part of a bus configuration.

For more information on frame captures, refer to [Capturing CAN Frames](#) on page 107.

### Basics on accessing the data of captured frames

Each frame capture lets you capture received CAN frames independently from a communication matrix. Via the Frame Capture Data feature, you can access the captured frame data.

When you add a frame capture to the Inspection part, the Frame Capture Data feature is added automatically. This feature provides the following function ports that let you access captured data:

- State function port
- Time function port
- Length function port
- Raw Data function port
- Identifier function port
- Extended Addressing function port
- CAN FD Frame Support function port
- Bit Rate Switch function port

The following illustration is an example of the function ports as displayed in the Bus Configurations table:

	Inspection	Bus Configuration Part Inspection
Frame Capture (1)	RX	Bus Frame Capture
Frame Capture Data	RX	Bus Frame Capture Data Inspection
Frame Capture (1) State		
Frame Capture (1) Identifier		
Frame Capture (1) Extended Addressing		
Frame Capture (1) CAN FD Frame Support		
Frame Capture (1) Bit Rate Switch		
Frame Capture (1) Time		
Frame Capture (1) Length		
Frame Capture (1) Raw Data		

#### Accessing captured data via function ports

The function ports of the Frame Capture Data feature let you access the captured data of all captured frames. For this purpose, the value of each function port is a vector: Except for the Raw Data function port, there is one vector element for each frame that can be captured. The required number of vector elements is determined by the Maximum number of captured frames property of the related frame capture. Refer to [Configuring frame captures](#) on page 108. For information on the vector elements of the Raw Data function port, refer to [Data provided by the Raw Data function port](#) on page 224.

The following table provides an overview of the data that is provided by the function ports.

Function Port	Description
State function port	Each vector element indicates whether a frame is captured in the current sampling step: <ul style="list-style-type: none"> <li>▪ False (0): No frame is captured in the current sampling step.</li> <li>▪ True (1): A frame is captured in the current sampling step.</li> </ul> For example, the vector size of the function port is 10, i.e., a maximum of 10 frames can be captured in one sampling step. In the current sampling step, only 6 frames are captured. As a result, the value of the first 6 vector elements is 1, the value of the remaining vector elements is 0.
Time function port	Each vector element provides the time (in seconds) of the <a href="#">executable application</a> when a captured frame was received.
Length function port	Each vector element provides the payload length (in bytes) of a captured frame.
Raw Data function port	Provides the raw data of captured frames. Refer to <a href="#">Data provided by the Raw Data function port</a> on page 224.
Identifier function port	Each vector element provides the frame identifier of a captured frame.
Extended Addressing function port	Each vector element indicates whether a captured frame uses the extended identifier format: <ul style="list-style-type: none"> <li>▪ False (0): The frame uses the standard identifier format (11-bit).</li> <li>▪ True (1): The frame uses the extended identifier format (29-bit).</li> </ul>
CAN FD Frame Support function port	Each vector element indicates whether a captured frame is a CAN FD frame: <ul style="list-style-type: none"> <li>▪ False (0): The frame is a classic CAN frame.</li> <li>▪ True (1): The frame is a CAN FD frame.</li> </ul>
Bit Rate Switch function port	Each vector element indicates whether a captured CAN FD frame supports switching the bit rate: <ul style="list-style-type: none"> <li>▪ False (0): The frame does not support switching the bit rate, i.e., the data phase and the arbitration phase of the frame are transmitted with the same bit rate.</li> </ul>

Function Port	Description
	<ul style="list-style-type: none"> <li>True (1): The frame supports switching the bit rate, i.e., the data phase of the frame can be transmitted with a higher bit rate than the arbitration phase.</li> </ul> <p>For more information on the bit rate switch, refer to <a href="#">CAN FD protocol</a> on page 47.</p>

The default values of the function ports are False or 0. Via the Initial value property of each function port, you can specify a user-defined initial function port value. To access the captured data at run time, you must enable model access or test automation support for the function ports. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

#### Data provided at run time

At run time, frames are captured in the order in which they are received. The captured data is written to the vector elements of the function ports from left to right and in descending chronological order, i.e., data of the latest captured frame is written to the first vector element.

For example, Maximum number of captured frames of the related frame capture is set to 6. In the first sampling step of the [executable application](#), only 3 frames are captured, in the second sampling step 4. Except for the Raw Data function port, this results in the following function port data:

Sampling Step	Function Port	Data of Vector Element					
		1	2	3	4	5	6
Sampling step 1	State function port	1	1	1	0	0	0
	All other function ports, except Raw Data	Captured frame 3	Captured frame 2	Captured frame 1	Initial function port value	Initial function port value	Initial function port value
Sampling step 2	State function port	1	1	1	1	0	0
	All other function ports, except Raw Data	Captured frame 4 of sampling step 2	Captured frame 3 of sampling step 2	Captured frame 2 of sampling step 2	Captured frame 1 of sampling step 2	Captured frame 3 of sampling step 1	Captured frame 2 of sampling step 1

#### Data provided by the Raw Data function port

The Raw Data function port provides the raw data of all captured frames. For this purpose, the number of vector elements of the function port is determined by the Maximum number of captured frames and Maximum frame length properties of the related frame capture as follows: <Value of Maximum number of captured frames> · <value of Maximum frame length>

For example, the following settings are specified:

- Maximum number of captured frames: 4
- Maximum frame length: 3 bytes

In this case, the value of the Raw Data function port is a vector with 12 elements. The elements 1 ... 3 provide the raw data of the latest captured frame,

the elements 4 ... 6 the raw data of the previously captured frame, etc. The mapping of vector elements to the frames is static. This results in the following:

- If the payload length of a captured frame exceeds the specified maximum frame length, the raw data of the exceeding number of bytes is not captured.
- If the payload length of a captured frame is smaller than the specified maximum frame length, vector elements are unused. These elements keep their old data.

For example, in the first sampling step of the executable application, only one frame with a payload length of 6 bytes is captured. In the second sampling step, three frames are captured. The payload length of the third captured frame is 2 bytes. With the specified settings above, this results in the following function port data:

	Raw Data Function Port: Data of Vector Element							
	1	2	3	4	5	6	7 ... 9	10 ... 12
Sampling step 1	Captured frame 1, byte 0	Captured frame 1, byte 1	Captured frame 1, byte 2	Initial function port value	Initial function port value			
Sampling step 2	Captured frame 3 of sampling step 2, byte 0	Captured frame 3 of sampling step 2, byte 1	Old data: Captured frame 1 of sampling step 1, byte 2	Captured frame 2 of sampling step 2, byte 0	Captured frame 2 of sampling step 2, byte 1	Captured frame 2 of sampling step 2, byte 2	Captured frame 3 of sampling step 2, byte 0 ... 2	Captured frame 1 of sampling step 2, byte 0 ... 2

### Tip

The Length function port provides the payload length of each captured frame. Therefore, you can use the Length function port to identify whether the Raw Data function port provides the raw data of all bytes of a frame and/or if vector elements are unused and provide old data.

#### Effects of deleting the Frame Capture Data feature

If you delete the Frame Capture Data feature, the related frame capture is not deleted automatically. As a result, frames are captured at run time but you cannot access the captured data. For optimum run-time performance, it is therefore recommended to directly delete a frame capture and not its related Frame Capture Data feature if you do not need the frame capture.

#### Related topics

#### References

[Function Outport Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Specifying the Direction of CAN Frame Gateways

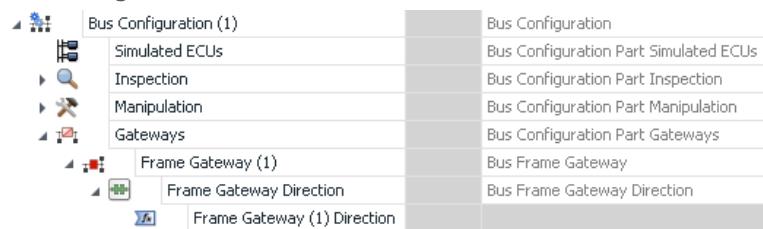
### Introduction

You can specify the gateway direction for frame gateways that are added to the Gateways part of a bus configuration.

### Basics on specifying the direction of CAN frame gateways

Each frame gateway that is added to the Gateways part of a bus configuration lets you exchange CAN communication between two [communication clusters](#). When you add a frame gateway to the Gateways part, the Frame Gateway Direction feature and its Direction function port are added automatically. The function port lets you specify the gateway direction.

The following illustration is an example of the function port as displayed in the Bus Configurations table:



For basic information on specifying gateways via frame gateways, refer to [Specifying CAN Gateways](#) on page 109.

### Specifying the gateway direction

The Direction function port lets you specify the gateway direction as follows:

- **Disabled (0):** Gatewaying is disabled.
- **Routing cluster 1 to 2 (1):** The bus communication is routed from cluster 1 to cluster 2.
- **Routing cluster 2 to 1 (2):** The bus communication is routed from cluster 2 to cluster 1.
- **Bidirectional (3):** The bus communication is routed from cluster 1 to cluster 2 and vice versa (bidirectional gateway). This is the default state.

Via the function port's Initial value property, you can change the default state. To change the gateway direction at run time, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

### Specifying saturation values

For the Direction function port, you can specify user-defined saturation values to limit the minimum and maximum values of the function port. If you want to do this, you must first set the bus configuration's Saturation usage property to User min/max value. Refer to [Specifying saturation values for function imports](#) on page 123.

**Effects of deleting the Frame Gateway Direction feature**

If you delete the Frame Gateway Direction feature from a frame gateway, the frame gateway itself is not deleted automatically. Therefore, the bus communication is still gatewayed at run time: The gateway direction is bidirectional and you can neither change the gateway direction nor disable the gateway.

**Related topics****References**

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties\)](#)

## Controlling Filters of Frame Captures and Frame Gateways

**Introduction**

You can enable, disable, and specify the filter mode of filters that are available for the following elements:

- Frame captures of the Inspection part of a bus configuration.
- Frame gateways of the Gateways part of a bus configuration.

For basic information on capturing and gatewaying frames, refer to [Capturing CAN Frames](#) on page 107 and [Specifying CAN Gateways](#) on page 109, respectively.

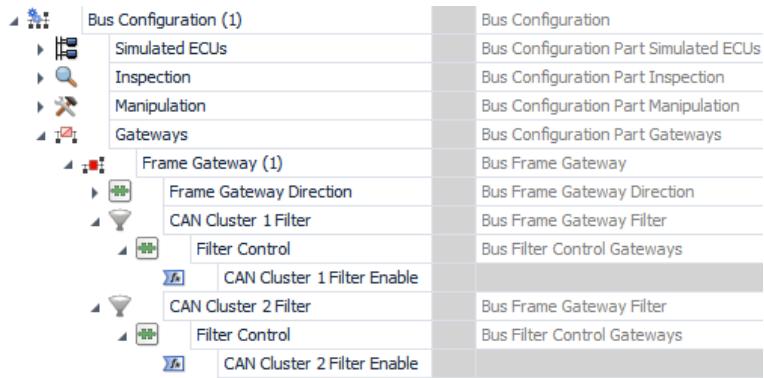
**Basics on controlling filters**

To each filter, the Filter Control feature is added automatically. This has the following effects:

- For the Filter Control feature node, a Filter mode property is available. This property lets you select whether the CAN frames that pass the related filter are included in or excluded from capturing or gatewaying.



- An Enable function port is available that lets you specify the enable state of the related filter. The following illustration is an example of the function port as displayed in the Bus Configurations table:



However, to filter CAN frames, you must add at least one filter rule to each filter. Refer to [Specifying Filters for Frame Captures and Frame Gateways](#) on page 111.

#### Specifying the filter mode

The Filter mode property of the Filter Control feature node lets you select the following filter modes:

- Exclude: The CAN frames that pass a filter rule of the related filter are not captured or gatewayed. Instead, all other CAN frames of the communication cluster are captured/gatewayed. This is the default mode.
- Include: Only the CAN frames that pass a filter rule of the related filter are captured/gatewayed.

The specified filter mode applies to all the filter rules of the related filter, i.e., you cannot specify different filter modes for individual filter rules of one filter. Additionally, you cannot change the specified filter mode during run time.

#### Specifying the enable state of a filter

The Enable function port lets you specify the enable state of the related filter as follows:

- False (0): The filter is disabled, i.e., all CAN frames of the cluster can be captured/gatewayed.
- True (1): The filter is enabled and the CAN frames can be filtered for capturing/gatewaying. This is the default state.

Via the function port's Initial value property, you can change the default state. To change the state at run time, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

#### Effects of deleting the Filter Control feature

If you delete the Filter Control feature from a filter, the filter and its filter rules are not deleted automatically. Therefore, the CAN frames of the cluster are still filtered for capturing/gatewaying:

- The filter is permanently enabled.
- All CAN frames that pass a specified filter rule are excluded from capturing/gatewaying.

**Related topics**

**References**

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties !\[\]\(1d229ba0de52e1ded5af744fd36cf38c\_img.jpg\)](#)

# Bus Configuration Features Available for Bus Configurations

## Enabling and Disabling Bus Configurations

### Introduction

You can enable and disable bus configurations to enable and disable the bus communication of a bus configuration completely or partly in one step.

### Basics on enabling and disabling bus configurations

To enable or disable a bus configuration, you must add the Bus Configuration Enable feature to the bus configuration. When you do this, a Bus Configuration Enable function port is available for the bus configuration that lets you specify the enable state.

The following illustration is an example of the function port as displayed in the Bus Configurations table:

	Bus Configuration (1)	Bus Configuration
	Bus Configuration Enable	Bus Configuration Enable Global
	Bus Configuration (1) Enable	

### Specifying the enable state of a bus configuration

The Bus Configuration Enable function port lets you specify the enable state of the bus configuration as follows:

- 0: Disabled: The complete bus configuration is disabled, i.e., no data is received or transmitted on the bus.
- 1: Enabled: The complete bus configuration is enabled, i.e., data can be received and transmitted on the bus. This is the default state.
- 2: RX enabled only: The bus configuration is partly enabled: Data is only received on the bus but no data can be transmitted on the bus.

Via the function port's Initial value property, you can change the default state. To change the bus configuration state at run time, you must enable model access or test automation support for the function port. Refer to [Configuring Function Ports for Bus Configuration Features](#) on page 120.

### Specifying saturation values

For the Bus Configuration Enable function port, you can specify user-defined saturation values to limit the minimum and maximum values of the function port. If you want to do this, you must first set the bus configuration's Saturation usage property to User min/max value. Refer to [Specifying saturation values for function imports](#) on page 123.

**Behavior of bus configurations in disabled or RX enabled only state**

Setting the state of a bus configuration to 0: Disabled or 2: RX enabled only results in the following behavior at run time:

State	Run-Time Behavior
0: Disabled	The entire bus communication of the bus configuration is disabled, i.e., no bus communication is simulated, inspected, manipulated, or gatewayed. Therefore, the bus configuration has no effects on the executable application. For example, this allows you to use the same bus access for variants of bus communication.
2: RX enabled only	The bus configuration only receives data on the bus, i.e., the reception of bus communication can be simulated and received bus communication can be inspected. The received data can be provided to a mapped behavior model, for example. In contrast, simulating the transmission of bus communication, manipulating bus communication, or gatewaying bus communication is not possible. This applies even if data must be transmitted, e.g., according to a used bus protocol. For example, no acknowledgment data is sent even though this might be required in J1939 communication.

**Switching the bus configuration state from disabled to RX enabled only**

Switching the bus configuration state at run time from 0: Disabled to 2: RX enabled only has the following effects:

Element	Available for	Effects
End-to-end protection counters	End-to-end-protected ISignal groups (according to the corresponding end-to-end protection profile)	The counters of end-to-end-protected ISignal groups are reset to 0.
Counter signals	ISignals with enabled Counter Signal feature	Counter signals are reset to the specified initial counter value.
PDU counters	PDUs with enabled PDU RX Status feature	PDU counters keep the last counter value and continue counting on the basis of this value.
RX global time domains	ECUs that are configured as time slaves in global time synchronization scenarios	The counters of time slaves are reset to 0.
Transport protocol address	Communication controllers with enabled J1939 Network Management Enable feature	Only if the J1939 Network Management Enable feature is set to the 2: Dynamic address handling enable state: The transport protocol address of the communication controller is reset to the address value that is specified in the communication matrix.

**Switching the bus configuration state from disabled or RX enabled only to enabled**

Switching the bus configuration state at run time from 0: Disabled or 2: RX enabled only to 1: Enabled has the following effects:

Element	Available for	Effects
End-to-end protection counters	End-to-end-protected ISignal groups (according to the corresponding end-to-end protection profile)	The counters of end-to-end-protected ISignal groups are reset to 0.
Counter signals	ISignals with enabled Counter Signal feature	Counter signals are reset to the specified initial counter value.
Cyclic timings	PDUs	The cyclic timings of PDUs are reset to the values specified for the time period and the time offset.
Container IPDUs	CAN communication	<ul style="list-style-type: none"> <li>▪ The container timeout of TX container IPDUs is reset to its initial state.</li> <li>▪ The queues of contained IPDUs in container IPDUs with a dynamic container layout are cleared, i.e., no contained IPDUs are queued.</li> <li>▪ The send buffer of contained IPDUs in container IPDUs with a static container layout is cleared, i.e., no contained IPDUs are buffered.</li> </ul>
PDU counters	PDUs with enabled PDU RX Status feature	PDU counters keep the last counter value and continue counting on the basis of this value.
TX and RX global time domains	ECUs that are configured as time masters or time slaves in global time synchronization scenarios	<ul style="list-style-type: none"> <li>▪ The counters of time masters and time slaves are reset to 0.</li> <li>▪ The timings of time masters are reset to their initial states.</li> <li>▪ The message queues are cleared, i.e., follow-up (FUP) messages are lost if their related synchronization (SYNC) messages were transmitted before the bus configuration was disabled.</li> </ul>
Transport protocol address	Communication controllers with enabled J1939 Network Management Enable feature	<ul style="list-style-type: none"> <li>▪ Only if the bus configuration state is switched from 0: Disabled to 1: Enabled and the J1939 Network Management Enable feature is set to the 2: Dynamic address handling enable state: The transport protocol address of the communication controller is reset to the address value that is specified in the communication matrix.</li> <li>▪ The communication controller sends an address claim on the CAN bus.</li> </ul>
LIN schedule tables	LIN masters	The active LIN schedule table starts from the beginning, regardless of the resume position that is specified in the communication matrix.
Manipulation counters	Enabled bus manipulation features	Manipulation counters keep the last counter value and continue counting on the basis of this value. This affects the countdown values of bus manipulation features: The countdown values are not reset but decremented with the next transmission or reception of the related bus configuration elements.
Frame gateways	Gateways bus configuration part with added frame gateways	The queues of frame gateways are cleared, i.e., no bus communication is queued for being gatewayed.

**Related topics**

**References**

[Function Import Properties \(Bus Configuration\) \(ConfigurationDesk Function Block Properties !\[\]\(7b5ac31091387d0784a7af62b48d4170\_img.jpg\)](#)

# Implementing Bus Communication in the Signal Chain Using the Bus Manager

---

## Where to go from here

## Information in this section

How to Add a Communication Matrix to a ConfigurationDesk Application.....	233
How to Add a Bus Configuration to a ConfigurationDesk Application.....	236
How to Assign Communication Matrix Elements to a Bus Configuration.....	238
How to Enable Model Access for Function Ports.....	242
How to Add Behavior Models to a ConfigurationDesk Application.....	244
Example of Selecting Multiple Similar Communication Matrix Elements at Once.....	247
Example of Mapping Model Ports to Bus Configuration Ports via Tables.....	249

## How to Add a Communication Matrix to a ConfigurationDesk Application

---

### Objective

To make bus network descriptions available in the ConfigurationDesk application, you must add one or more communication matrices to an application.

**Basics**

For basic information, refer to [Working with Communication Matrices](#) on page 58.

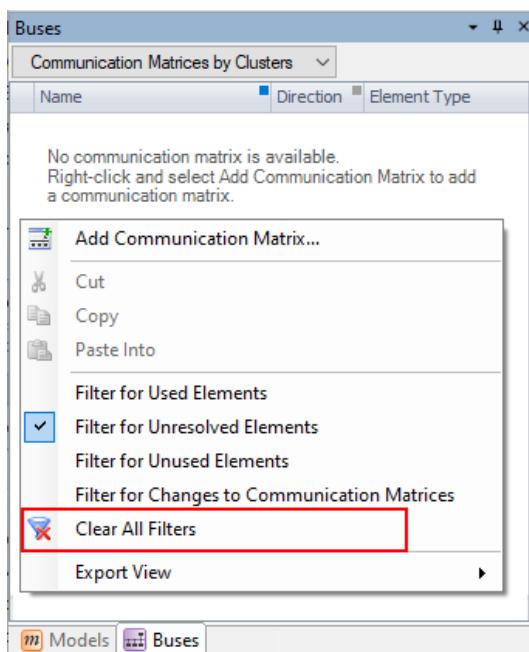
**Precondition**

A ConfigurationDesk project with an application is open.

**Method**

**To add a communication matrix to an application**

- 1 Switch to the Buses view set if necessary.
- 2 Right-click in the Buses Browser to open the context menu.
- 3 Click the Clear All Filters command if the command is available.



All active filters of the Buses Browser are cleared at once, i.e.:

- All active element filters, which are indicated by a check mark in the context menu and highlighted on the Home ribbon.
- All active column filters, which are indicated by a blue square in the column header.

**Tip**

- If the Buses Browser is closed, you can open it via Windows on the Home ribbon.
- Depending on the active filters, a communication matrix might be hidden when you import it. In this case, the Buses Browser displays a text, informing you that no elements match the filter conditions. By clearing all active filters you prevent the communication matrix from being hidden.

- 4 Right-click in the Buses Browser and select Add Communication Matrix from the context menu.

**Tip**

You can also access the command via the Home ribbon.

The Add Communication Matrix dialog opens.

- 5 In the dialog, navigate to the communication matrix you want to add and click Open.

The communication matrix is added to the application. The Message Viewer provides information on the import:

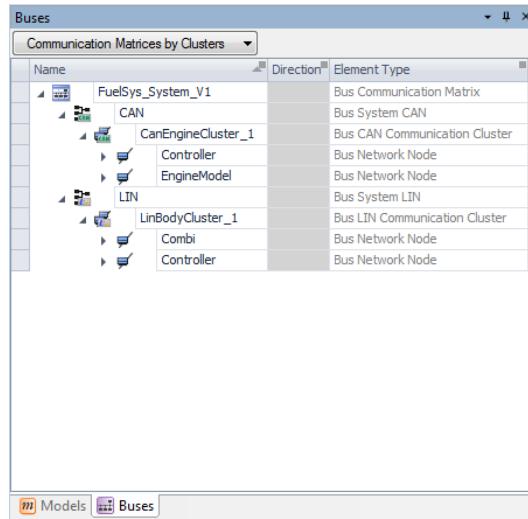
- If the Message Viewer displays an error message, the import failed, e.g., because you tried to add a communication matrix while a matrix with identical content already exists in the application.
- If the Message Viewer displays warning messages, inconsistent and/or invalid communication matrix settings were detected during the import. In this case, the communication matrix is imported but at least one of the following communication matrix conflicts occurs:
  - Communication matrix: Failed consistency checks during import of communication matrix
  - Communication matrix: Failed schema check during import of communication matrix

**Note**

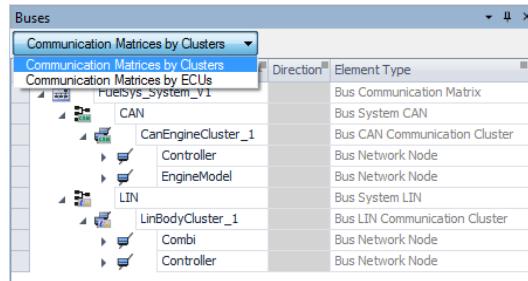
The inconsistent and invalid settings might result in unintended behavior in ConfigurationDesk or at run time. It is recommended that you correct the original communication matrix and work with the corrected matrix. For more information, refer to [Basics on Bus Manager-Related Conflicts](#) on page 99.

**Result**

The communication matrix is added to the application and displayed in the Buses Browser. Depending on the selected view, the communication matrix elements are sorted by clusters or ECUs.



The drop-down list of the Buses Browser lets you change the view.



#### Next step

To make bus communication available in the signal chain, you can now add a bus configuration to the application. Refer to [How to Add a Bus Configuration to a ConfigurationDesk Application](#) on page 236.

## How to Add a Bus Configuration to a ConfigurationDesk Application

#### Objective

To be able to implement bus communication in the [signal chain](#), you must add one or more bus configurations to a ConfigurationDesk application.

#### Precondition

A ConfigurationDesk project with an application is open.

#### Method

##### To add a bus configuration to an application

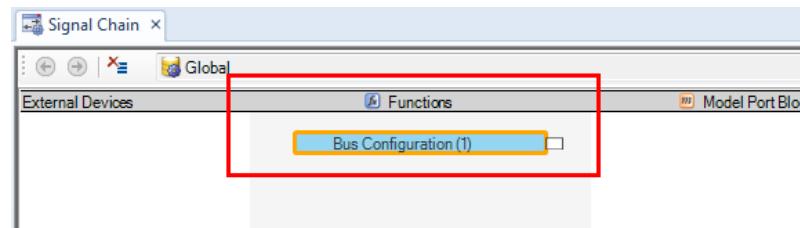
- 1 Switch to the Buses view set if necessary.

**2** On the Home ribbon, click .

A new bus configuration with a unique default name is added. The Bus Configurations table displays the bus configuration and its Simulated ECUs, Inspection, Manipulation, and Gateways parts.

Name	Direction	Element Type	Related Clusters	Related TX ECUs
Bus Configuration (1)		Bus Configuration		
Simulated ECUs		Bus Configuration Part Simulated ECUs		
Inspection		Bus Configuration Part Inspection		
Manipulation		Bus Configuration Part Manipulation		
Gateways		Bus Configuration Part Gateways		

The Global working view displays the related Bus Configuration function block.



**Tip**

- If the Bus Configurations table is closed, you can open it via Windows on the Home ribbon.
- You can access the Global working view via the Signal Chain view set, for example.
- You can also add a bus configuration in the following ways:
  - Via context menu of the Bus Configurations table.
  - By dragging communication matrix elements to the Bus Configurations table if no bus configuration is available in the application yet.
  - Via the context menu of the Bus Configuration table, you can also add multiple bus configurations to the application at the same time.

**3** You are recommended to rename the bus configuration according to your requirements.

To do so, select the bus configuration in the Bus Configurations table and rename it.

**Tip**

You can also rename the bus configuration via the related function block, in the Properties Browser, or in other tables. Refer to [How to Rename Function Blocks \(ConfigurationDesk Real-Time Implementation Guide\)](#).

**Result**

You added a bus configuration to the application.

<b>Next steps</b>	To make bus communication available in the signal chain and configure it for simulation, inspection, or manipulation purposes, you can now assign communication matrix elements to the bus configuration. Refer to <a href="#">How to Assign Communication Matrix Elements to a Bus Configuration</a> on page 238.
-------------------	--

<b>Related topics</b>	Basics
	<a href="#">Basics on Bus Configurations</a> .....63

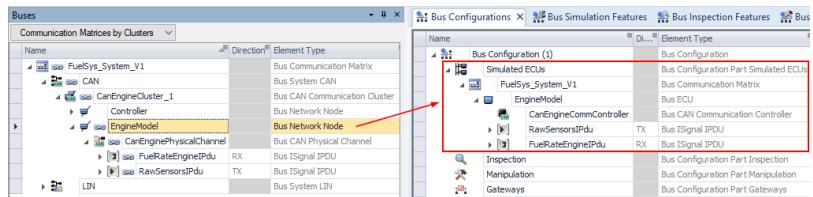
## How to Assign Communication Matrix Elements to a Bus Configuration

---

<b>Objective</b>	To make bus communication available in the signal chain, you must assign communication matrix elements to a bus configuration.  Additionally, it is recommended to rename the communication matrix nodes in the bus configuration, especially if you want to work with test automation scripts later on. If you work with DBC or LDF communication matrices, this also applies to the names of communication clusters. For more information, refer to <a href="#">Element identification via node names</a> on page 65.
<b>Basics</b>	Depending on the communication matrix element and the bus configuration part you assign the element to, related higher-level elements and subelements are assigned as well. For more information, refer to <a href="#">Assigning Communication Matrix Elements to Bus Configurations</a> on page 75.
<b>Precondition</b>	A communication matrix is available in the active ConfigurationDesk application.
<b>Method</b>	<b>To assign communication matrix elements to a bus configuration</b> <ol style="list-style-type: none"><li>1 Switch to the Buses view set if necessary.</li><li>2 In the Buses Browser, select a communication matrix element and drag it to the Bus Configurations table.</li></ol>

The selected element is assigned to a bus configuration in one of the following ways:

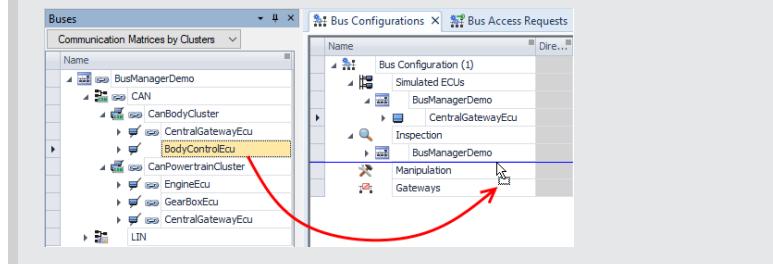
- If no bus configuration existed in the ConfigurationDesk application before, a new bus configuration is added and the selected communication matrix element is assigned to the Simulated ECUs part.



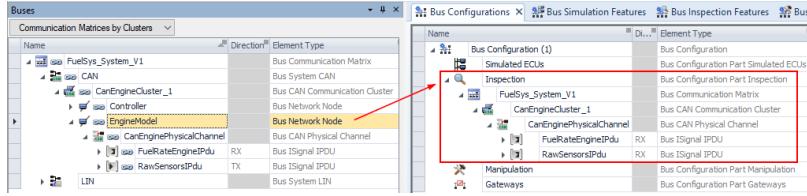
- If you dragged the communication matrix element to the bus configuration node or the Simulated ECUs node, or above or below any part node of an existing bus configuration, the communication matrix element is assigned to the Simulated ECUs part.

Tip

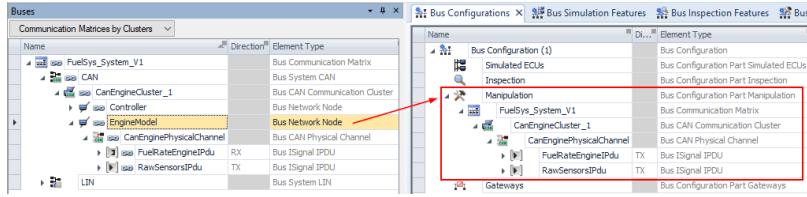
If you drag an element above or below a part node, a blue line indicates that the element will be assigned to the Simulated ECUs part when you drop it, as `BodyControlEcu` in the following example.



- If you dragged the communication matrix element to the **Inspection** node of an existing bus configuration, the communication matrix element is assigned to the **Inspection** part.



- If you dragged the communication matrix element to the **Manipulation** node of an existing bus configuration, the communication matrix element is assigned to the **Manipulation** part.



Relevant higher-level elements and subelements of the communication matrix element are assigned as well. All the assigned elements of the communication matrix are marked by a chain symbol.

- In the Bus Configurations table, select a communication matrix node and specify a meaningful name for it.

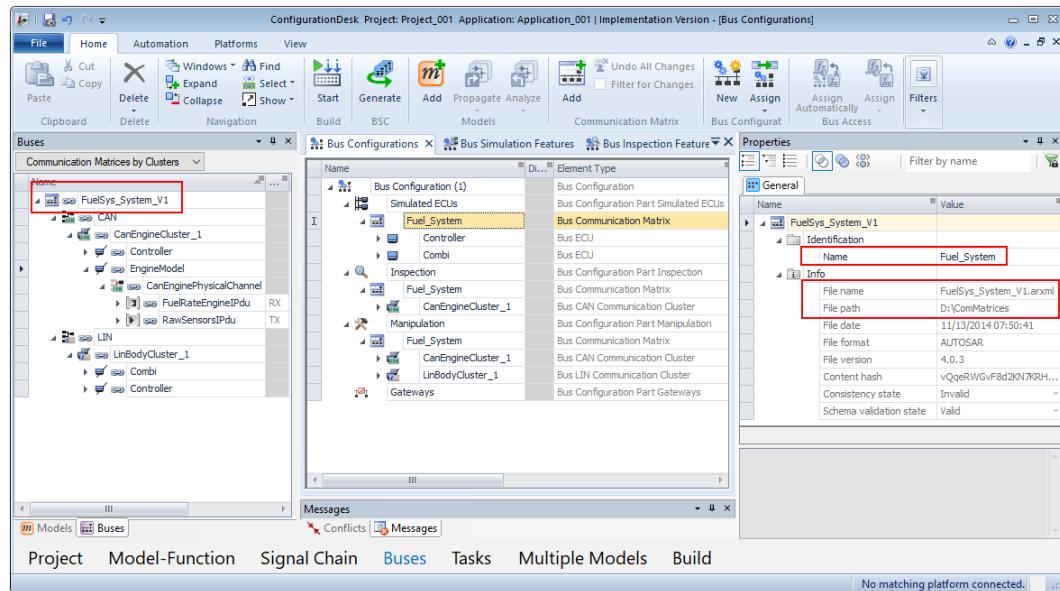
### Tip

Each communication matrix that is assigned to a bus configuration is represented by up to four communication matrix nodes. Changes you make for one of these nodes always apply to the other nodes as well.

The communication matrix node is renamed. The Properties Browser displays:

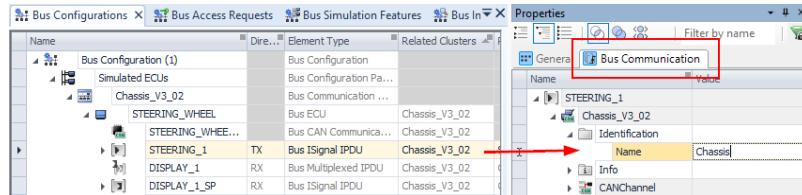
- The Name of the communication matrix nodes in the bus configuration structure
- The File name of the assigned communication matrix file
- The File path of the communication matrix file

The Buses Browser displays the communication matrix file name without file name extension.



- 4 If you assigned elements of a DBC or LDF communication matrix, specify a meaningful name for the communication cluster:

- If you assigned elements to the Simulated ECUs part, select an assigned PDU and edit the cluster name on the Bus Communication page of the Properties Browser.



- If you assigned elements to the Inspection or Manipulation part, rename the related communication cluster node in the Bus Configurations table.

Name	Element Type
Bus Configuration (1)	Bus Configuration
Simulated ECUs	Bus Configuration Part Simulated E...
Chassis_V3_02	Bus Communication ...
STEERING_WHEEL	Bus ECU
STEERING_WHEEL	Bus CAN Communication Cluster
STEERING_WHEEL	Bus ISignal IPDU
DISPLAY_1	TX
DISPLAY_1	RX
DISPLAY_1_SP	RX

- 5 Repeat the steps above to assign further elements of one or more communication matrices to the bus configuration.

**Tip**

You can use the communication matrix filters and the **Select Elements by Type** command to easily select multiple similar communication matrix elements at once and assign them to a bus configuration. For an example, refer to [Example of Selecting Multiple Similar Communication Matrix Elements at Once](#) on page 247.

---

**Result**

You assigned communication matrix elements to a bus configuration and renamed the communication matrix nodes of the bus configuration structure. If you work with a DBC or LDF communication matrix, you also renamed the communication cluster in the bus configuration.

---

**Next steps**

- You can now configure the bus communication of the bus configurations. For example, you can add bus configuration features or specify user-defined settings for communication matrix elements. For overviews of the configurable elements, refer to:
  - [Basics on Bus Configuration Features](#) on page 116
  - [Basics on Modifying Communication Matrices](#) on page 267
- If you want to work with a behavior model and the behavior model already exists, you can add the model to the ConfigurationDesk application. Refer to [How to Add Behavior Models to a ConfigurationDesk Application](#) on page 244
- If you want to work without a behavior model, you can assign the bus configurations to application processes. Refer to [Working Without Behavior Models](#) on page 263.

## How to Enable Model Access for Function Ports

---

**Objective**

To map bus configurations to a behavior model (e.g., to use dynamic signal values during run time), you must enable model access for function ports.

---

**Precondition**

- Communication matrix elements are assigned to a bus configuration.
  - Bus configuration features that provide function ports are added to the bus configuration.
- For more information on bus configuration features, refer to [Basics on Bus Configuration Features](#) on page 116.

**Method****To enable model access for function ports**

- 1 Switch to the Buses view set if necessary.
- 2 Open the Bus Configuration Ports table.  
The table displays all ports of all the bus configurations of the active application.
- 3 Select one or multiple function ports whose model access you want to enable.

**Tip**

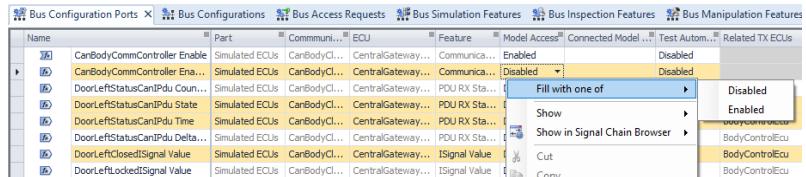
You can use the **Ctrl** and **Shift** keys for multiselection.

- 4 Enable the model access:

- If you selected one function port, select Enabled from the Model Access column.

Name	Part	Communi...	ECU	Feature	Model Access	Conn
I CanBodyCommController Enable	Simulated ECUs	CanBodyCl...	CentralGateway...	Communica...	Enabled	
I CanBodyCommController Ena...	Simulated ECUs	CanBodyCl...	CentralGateway...	Communica...	Disabled	
I DoorLeftStatusCanIpdu Coun...	Simulated ECUs	CanBodyCl...	CentralGateway...	PDU RX Sta...	Disabled	
I DoorLeftStatusCanIpdu State	Simulated ECUs	CanBodyCl...	CentralGateway...	PDU RX Sta...	Disabled	
I DoorLeftStatusCanIpdu Time	Simulated ECUs	CanBodyCl...	CentralGateway...	PDU RX Sta...	Disabled	

- If you selected multiple function ports, right-click the Model Access column of a selected function port and select Fill with one of - Enabled from the context menu.



You enabled model access for the selected function ports.

**Tip**

You can also enable model access for each function port in the Properties Browser.

**Result**

You enabled model access for the selected function ports. In working views, the enabled function ports are displayed for the Bus Configuration function blocks. Enabling model access allows you to map model ports to the function ports and to exchange dynamic signal values with the behavior model.

**Next steps**

You can now map the function ports to model ports:

- If model port blocks are already available in the ConfigurationDesk application, you can manually map them to the function ports. For an example, refer to

[Example of Mapping Model Ports to Bus Configuration Ports via Tables](#) on page 249. For basic information on model port mapping, refer to [Model Port Mapping \(ConfigurationDesk Real-Time Implementation Guide\)](#).

- If you already have a suitable behavior model but it is not available in the ConfigurationDesk application yet, add the model to the ConfigurationDesk application. When you do this, model port blocks are available in the Model Browser. You can then map the function ports to the available model ports. Refer to [How to Add Behavior Models to a ConfigurationDesk Application](#) on page 244.
- If you do not have a suitable behavior model yet but MATLAB/Simulink and [Model Interface Package for Simulink](#) are installed, you can generate the required model port blocks for ConfigurationDesk and a MATLAB/Simulink model. For more information, refer to [Working with Simulink Behavior Models \(ConfigurationDesk Real-Time Implementation Guide\)](#).

## How to Add Behavior Models to a ConfigurationDesk Application

---

### Objective

To exchange data between ConfigurationDesk and a behavior model, ConfigurationDesk requires information on the behavior model. Therefore, you must add the behavior model to the ConfigurationDesk application.

### Preconditions

- A ConfigurationDesk project with an application is open.
- A behavior model exists.

#### Note

If you want to generate bus simulation containers later on, the behavior model must be a Simulink implementation container (SIC file). For more information, refer to [Basics on Bus Simulation Containers](#) on page 253.

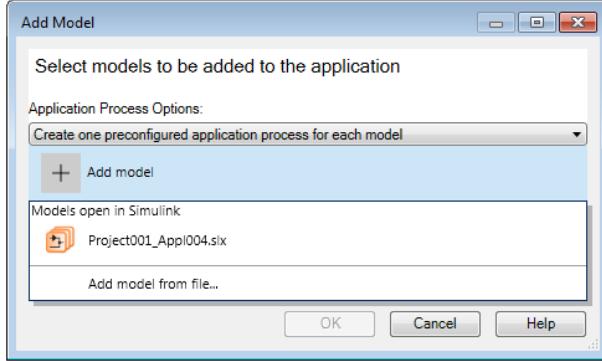
---

### Method

#### To add a behavior model to a ConfigurationDesk application

- 1 In ConfigurationDesk, select the Model Browser.
- 2 Right-click in the Model Browser and select Import - Add Model from the context menu.  
The Add Model dialog opens.

- 3 In the Add Model dialog, click Add model.



- 4 Select your behavior model:

- If it is displayed in the Models open in Simulink list, select it in the list.
- If it is not displayed, click Add model from file to open a standard Open dialog and select your behavior model (e.g., \*.mdl, \*.slx, \*.sic, or \*.fmu) from the file system.

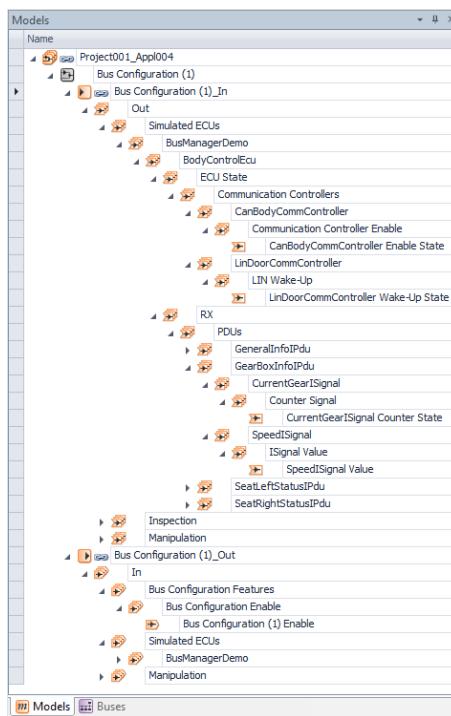
The selected behavior model is displayed in the Add Model dialog.

**Note**

If you want to generate bus simulation container (BSC) files later on, ensure that Create one preconfigured application process for each model is selected from the Application Process Options list. For more information, refer to [Basics on Bus Simulation Containers on page 253](#).

- 5 Click OK.

The dialog closes and the Model Browser displays the analyzed model topology. The following illustration is an example of a Simulink model that was generated for a bus configuration named 'Bus Configuration (1)'.



### Result

You added a behavior model to the application.

If the behavior model and the signal chain contain model port blocks whose identities match, these model port blocks of the signal chain become resolved. For more information, refer to [Basics of Connecting ConfigurationDesk and Simulink Behavior Models \(ConfigurationDesk Real-Time Implementation Guide\)](#).

For basic information on handling the model interface, refer to [Basics on the Model Interface \(ConfigurationDesk Real-Time Implementation Guide\)](#).

### Next steps

If function ports of bus configurations are not mapped to model ports yet, you can now map them. For an example, refer to [Example of Mapping Model Ports to Bus Configuration Ports via Tables](#) on page 249.

### Related topics

### References

[Add Model \(ConfigurationDesk User Interface Reference\)](#)

## Example of Selecting Multiple Similar Communication Matrix Elements at Once

### Introduction

You can easily select multiple similar communication matrix elements at once by using the communication matrix filters and the Select Elements by Type command. This is useful, for example, if you want to assign multiple similar communication matrix elements to a bus configuration.

This example shows you how to select all the TX ISignals of the CentralGatewayEcu in the Buses Browser at once.

Name	Direction	Element Type
BusManagerDemo		Bus Communication Matrix
CentralGatewayEcu		Bus ECU
GeneralInfoIPdu	TX	Bus ISignal IPDU
KI_15ISignal	TX	Bus ISignal
GearBoxInfoIPdu	TX	Bus ISignal IPDU
CurrentGearISignal	TX	Bus ISignal
SpeedISignal	TX	Bus ISignal
DoorLeftStatusCanIPdu	RX	Bus ISignal IPDU
DoorLeftClosedISignal	RX	Bus ISignal
DoorLeftLockedISignal	RX	Bus ISignal
DoorRightStatusCanIPdu	RX	Bus ISignal IPDU
DoorRightClosedISignal	RX	Bus ISignal
DoorRightLockedISignal	RX	Bus ISignal
CarLockControlIPdu	RX	Bus ISignal IPDU
LockCarISignal	RX	Bus ISignal
UnlockCarISignal	RX	Bus ISignal
PowerStatusIPdu	TX	Bus ISignal IPDU
KI_15ISignal	TX	Bus ISignal
GearBoxInfoIPdu	RX	Bus ISignal IPDU
CurrentGearISignal	RX	Bus ISignal
SpeedISignal	RX	Bus ISignal
EngineInfoIPdu	RX	Bus ISignal IPDU
FuelConsumptionISignal	RX	Bus ISignal
EngineSpeedISignal	RX	Bus ISignal
EngineTempISignal	RX	Bus ISignal
BodyControlEcu		Bus ECU
EngineEcu		Bus ECU

### Tip

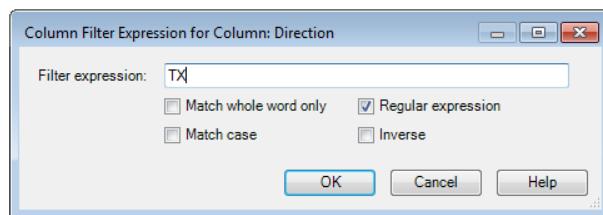
The CentralGatewayEcu is defined in the `BusManagerDemo.arxml` communication matrix which is available in the BusManagerDemo folder of your [Documents folder](#).

### Precondition

The Direction column is available in the Buses Browser. If it is not available, add it via the Column Chooser command.

**Method****To select multiple similar communication matrix elements at once**

- 1 Add the `BusManagerDemo.arxml` communication matrix to your ConfigurationDesk application. Refer to [How to Add a Communication Matrix to a ConfigurationDesk Application](#) on page 233.
- 2 In the Buses Browser, right-click the Direction column and select Set Column Filter from the context menu.  
A dialog opens for you to specify the column filter.
- 3 In the dialog, specify `TX` as the Filter expression.
- 4 Make sure that Regular expression is selected and Inverse is cleared.



- 5 Click OK.

The dialog closes and the Direction column is filtered for all TX elements. A blue square in the column header indicates the active filter.

Name	Direction	Element Type
BusManagerDemo		Bus Communication Matrix
CentralGatewayEcu		Bus ECU
GeneralInfoIPdu		Bus ISignal IPDU
Kl_15ISignal	TX	Bus ISignal
GearBoxInfoIPdu	TX	Bus ISignal IPDU
CurrentGearISignal	TX	Bus ISignal
SpeedISignal	TX	Bus ISignal
PowerStatusIPdu	TX	Bus ISignal IPDU
Kl_15ISignal	TX	Bus ISignal
BodyControlEcu		Bus ECU
EngineEcu		Bus ECU

- 6 Right-click the CentralGatewayEcu and select Select Elements by Type - Bus ISignal from the context menu.

All the TX ISignals of the CentralGatewayEcu are selected.

Name	Direction	Element Type
BusManagerDemo		Bus Communication Matrix
CentralGatewayEcu		Bus ECU
GeneralInfoIPdu	TX	Bus ISignal IPDU
Kl_15ISignal	TX	Bus ISignal
GearBoxInfoIPdu	TX	Bus ISignal IPDU
CurrentGearISignal	TX	Bus ISignal
SpeedISignal	TX	Bus ISignal
PowerStatusIPdu	TX	Bus ISignal IPDU
Kl_15ISignal	TX	Bus ISignal
BodyControlEcu		Bus ECU
EngineEcu		Bus ECU

<b>Result</b>	You selected all the TX ISignals of the CentralGatewayEcu at once by using communication matrix filters and the Select Elements by Type command.
<b>Next steps</b>	You can now assign the selected signals to a bus configuration, for example, by dragging them to a bus configuration in the Bus Configurations table. Refer to <a href="#">How to Assign Communication Matrix Elements to a Bus Configuration</a> on page 238.
<b>Related topics</b>	<b>References</b> <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <a href="#">Column Chooser (ConfigurationDesk User Interface Reference)</a> </div>

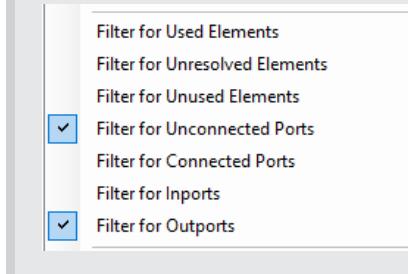
## Example of Mapping Model Ports to Bus Configuration Ports via Tables

<b>Introduction</b>	When a behavior model is available in the Model Browser and you must map <a href="#">model ports</a> to bus configuration ports manually, you can map the ports in a working view or via the following tables:
	<ul style="list-style-type: none"> <li>▪ Bus Configurations table</li> <li>▪ Bus Configuration Ports table</li> </ul> <p>Mapping model ports to bus configuration ports via tables is often more convenient because the tables provide a better overview of the available ports than a working view. Additionally, the Model Browser and the tables provide various column and port filter options that reduce the number of the displayed ports.</p>
	In the following example, model ports are mapped to bus configuration function ports via the Bus Configuration Ports table and by using some of the available filter options.
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>▪ A behavior model providing at least one unmapped model outport is available in the Model Browser.</li> <li>▪ At least one bus configuration provides an unmapped function inport.</li> </ul>
<b>Method</b>	<p><b>To map model ports to bus configuration ports via tables</b></p> <ol style="list-style-type: none"> <li>1 Switch to the Buses view set if necessary.</li> <li>2 In the Model Browser, right-click in a free area and select the following filters from the context menu: <ul style="list-style-type: none"> <li>▪ Filter for Unconnected Ports</li> <li>▪ Filter for Outports</li> </ul> </li> </ol>

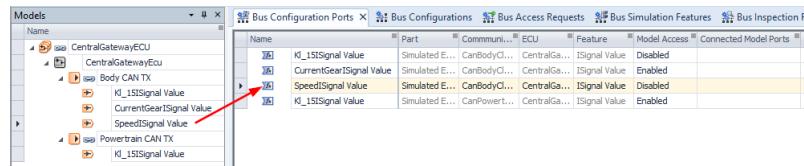
The selected filters are active. The Model Browser displays all the model outports that are not mapped to function ports.

### Tip

Active port filters are indicated by check marks in the context menu.



- 3 Open the Bus Configuration Ports table.
  - 4 In the Bus Configuration Ports table, right-click in a free area and select the following filters from the context menu:
    - Filter for Unconnected Ports
    - Filter for Imports
- The Bus Configuration Ports table displays all the function imports that are not mapped to model ports.
- 5 To map a model port to a function port, drag the model port from the Model Browser to a function port in the Bus Configuration Ports table.



### Tip

If you drag a model port to a function port with disabled model access, model access is automatically enabled for this function port.

The model port is mapped to the function port. Because Filter for Unconnected Ports is still active in the Model Browser and the Bus Configuration Ports table, none of the ports are displayed.

### Note

Selected port filters are active until you deactivate them explicitly. Active port filters are indicated only in the context menu and not in the Model Browser or Bus Configuration Ports table.

- 6 In the context menus of the Model Browser and Bus Configuration Ports table, clear Filter for Unconnected Ports.

The Model Browser and Bus Configuration Ports table display the mapped ports. The Connected Model Ports column of the Bus Configuration Ports table displays the model ports that are mapped to function ports.

**Tip**

If you mapped a model port to a function port by mistake, you can undo the mapping (e.g., via **Ctrl+Z**).

- 7 Repeat the steps above to map further model ports to bus configuration ports. If required, change the selected filter options, sort the displayed columns, or specify column filters, for example.

---

**Result**

You mapped model ports to bus configuration ports via the Bus Configuration Ports table by using some of the available filter options.

---

**Next steps**

You can inspect the mapping states of the specified mappings in a working view (e.g., to identify conflicting mappings). For an overview of the available mapping states, refer to [Mapping States and Their Effects \(ConfigurationDesk Real-Time Implementation Guide\)](#). If required, you can remove mappings by deleting mapping lines in the working view.

**Tip**

You can neither inspect the mapping states nor remove mappings in the Bus Configurations or Bus Configuration Ports table.



# Generating Bus Simulation Containers

---

## Where to go from here

## Information in this section

Basics on Bus Simulation Containers.....	253
Port Interface of Bus Simulation Containers.....	256
How to Generate Bus Simulation Containers.....	259

## Basics on Bus Simulation Containers

---

### Introduction

The Bus Manager lets you generate bus simulation containers. Bus simulation containers contain bus communication configured with the Bus Manager and let you use it independently from the Bus Manager on various dSPACE simulation platforms.

---

### Fields of application for bus simulation containers

The main fields of application for bus simulation containers are:

- Implementing bus communication in [offline simulation applications](#) ⓘ  
You can import bus simulation containers to the [VEOS Player](#) ⓘ. The VEOS Player lets you assign the bus communication to different communication clusters and implement it in an offline simulation application for [VEOS](#) ⓘ.
- Implementing bus communication in [real-time applications](#) ⓘ  
You can add bus simulation containers as a [model implementation](#) ⓘ to ConfigurationDesk applications. ConfigurationDesk lets you specify the access to real-time hardware and implement the bus communication in a real-time application for dSPACE SCALEXIO or MicroAutoBox III systems.

**Tip**

- Because the code related to the bus communication is already included in the bus simulation container, this code is not generated during the build process. This reduces the effort for building the real-time application.
- If you work with the Bus Manager in ConfigurationDesk, you can implement bus communication in real-time applications without using bus simulation containers.

Moreover, bus simulation containers let you handle the configured bus communication independently of the ConfigurationDesk project. For example, you can use bus simulation containers to:

- Archive the configured bus communication.
- Exchange the configured bus communication between different users.

<b>Working with Behavior Models</b>	To generate bus simulation containers, you can work with or without behavior models.
-------------------------------------	--

<b>Working with Behavior Models</b>	<b>Working Without Behavior Models</b>
If you work with a behavior model, the behavior model must be a Simulink implementation container (SIC) file.	<p>If you work without a behavior model, you must create application processes and assign bus configurations to the application processes manually.</p> <p>For an overview of a typical workflow, refer to <a href="#">Generating Bus Simulation Containers Without a Behavior Model</a> on page 34.</p>

**Note**

The Bus Manager lets you work with multiple behavior models in one application process. This is not supported for generating bus simulation containers. If you work with multiple behavior models, you must ensure that you create a separate application process for each model. For instructions, refer to [How to Add Behavior Models to a ConfigurationDesk Application](#) on page 244.

For an overview of a typical workflow, refer to [Generating Bus Simulation Containers with a Behavior Model](#) on page 33.

<b>Generating bus simulation containers</b>	You can generate bus simulation containers via one command. When you start generating bus simulation containers, one BSC file is generated for each application process that fulfills the following conditions: <ul style="list-style-type: none"> <li>▪ The application process contains no or only one behavior model. If the application process contains a behavior model, it must be a Simulink implementation container.</li> <li>▪ The application process contains at least one bus configuration.</li> </ul>
---	---

**Note**

During the generation of bus simulation containers, existing BSC files are deleted from the Generated Containers folder without notice. If you want to keep these BSC files, you must store them in another folder before you start the generation process.

The generated BSC files contain all the bus configurations that are part of the related application processes, regardless of whether the bus configurations are mapped to model ports of Simulink implementation containers. Only bus configurations for which conflicts exist that abort the code generation are not included in the BSC files. For more information, refer to [Basics on Bus Manager-Related Conflicts](#) on page 99.

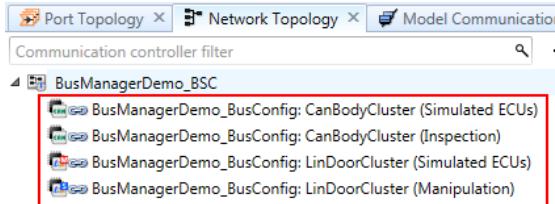
### Modifying bus communication in bus simulation containers

You cannot modify the bus communication that is included in a bus simulation container. If you want to do so (e.g., add ISignals to a bus configuration, specify bus configuration features etc.), you must modify the bus configurations for which you generated the bus simulation container. Then, you must generate a new bus simulation container.

### Availability of bus access requests in bus simulation containers

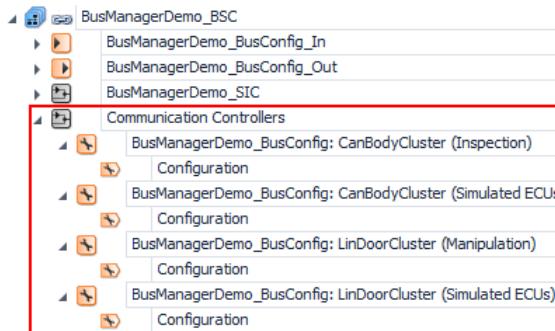
[Bus access requests](#) are included in the BSC files when you generate bus simulation containers. To access bus channels, you must assign the bus access requests to [bus accesses](#) in the VEOS Player or ConfigurationDesk as follows:

**Bus access requests in the VEOS Player** In the VEOS Player, the bus access requests are available as communication controllers:



To specify the bus access, you must connect the communication controllers to communication clusters.

**Bus access requests in ConfigurationDesk** In ConfigurationDesk, Configuration Port blocks are available providing Configuration ports:



To specify the bus access, you must map the Configuration ports to suitable bus function blocks (CAN, LIN) and assign hardware resources to the bus function blocks.

---

**Port interface of bus simulation containers**

Bus simulation containers provide a port interface that can include [model ports](#) of Simulink implementation containers and function ports of bus configurations. Ports that are included in the port interface can be used in the VEOS Player and ConfigurationDesk. Refer to [Port Interface of Bus Simulation Containers](#) on page 256.

---

**Using bus communication in offline simulations and real-time applications**

For bus communication that is configured via the Bus Manager, the paths of variable descriptions are the same in offline simulation applications and real-time applications. This lets you, for example, use the same ControlDesk layouts for offline simulation applications and real-time applications.

However, reusing bus communication in offline simulation applications and real-time applications limits the maximum number of application processes to the number of usable cores of the available processing hardware (e.g., SCALEXIO Processing Unit or DS1403 Processor Board). For an overview of the correlation between application processes and processing hardware, refer to [Terms and Definitions for Building Executable Applications \(ConfigurationDesk Real-Time Implementation Guide\)](#).

---

**Limitations for working with bus simulation containers**

When you work with bus simulation containers, some limitations apply. Refer to [Limitations for Bus Simulation Containers](#) on page 331.

## Port Interface of Bus Simulation Containers

---

**Introduction**

Bus simulation containers provide a port interface that can include [model ports](#) of Simulink implementation containers and function ports of bus configurations. Ports that are included in the port interface can be used in the VEOS Player and ConfigurationDesk.

In addition to the ports of the port interface, bus simulation containers provide elements for accessing bus channels when you import BSC files to the VEOS Player or ConfigurationDesk. Refer to [Availability of bus access requests in bus simulation containers](#) on page 255.

<b>Ports included in the port interface</b>	Model ports and function ports are included in the port interface of bus simulation containers if the following preconditions are fulfilled:
---	--

<b>Preconditions for</b>	
<b>Model Ports</b>	<b>Function Ports</b>
<ul style="list-style-type: none"> <li>The model ports are part of a Simulink implementation container that is assigned to the application process for which a bus simulation container is generated.</li> <li>The model ports are not mapped to ports of a bus configuration, i.e., they are unmapped or mapped to ports of other ConfigurationDesk function blocks.</li> </ul> <p><b>Tip</b></p> <p>This applies to <a href="#">data imports</a>, <a href="#">data outports</a>, and <a href="#">Runnable function ports</a> of the Simulink implementation container.</p>	<ul style="list-style-type: none"> <li>The function ports are part of a bus configuration that is assigned to the application process for which a bus simulation container is generated.</li> <li>Model access is enabled for the function ports but they are not mapped to model ports.</li> </ul>

The port interface is not valid for event ports, i.e., event ports of bus configurations are not included in the port interface, regardless of whether they are mapped to runnable function ports.

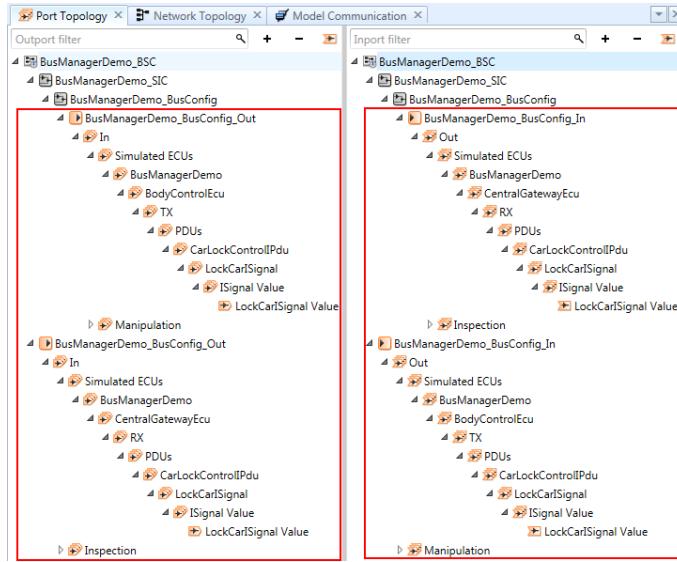
#### Note

For optimum run-time performance, it is recommended to include as few function ports as possible in the port interface of bus simulation containers, i.e.:

- Enable model access only for those bus configuration function ports whose data you want to exchange with a behavior model.
- Whenever reasonable, map bus configuration function ports with enabled model access to suitable model ports before you generate BSC files.

<b>Using ports of the port interface in the VEOS Player</b>	In the VEOS Player, data imports, data outports, and function ports of the port interface are available as signals. You can, for example, connect these signals to
---	--

signals of different plant models when you configure the bus communication in the VEOS Player.



#### Note

Runnable function ports are not available in the VEOS Player and the related [Runnable functions](#) cannot be executed at run time. If you import a BSC file that contains runnable function ports to the VEOS Player, the VEOS Player Messages pane displays warning messages.

#### Using ports of the port interface in ConfigurationDesk

In ConfigurationDesk, the following [model port blocks](#) can be available:

- Data port blocks

These blocks are available for the data imports, data exports, and function ports of the port interface.

- Runnable Function blocks

These blocks are available for runnable function ports of the port interface.



You can use the ports in the signal chain and map them to ConfigurationDesk function blocks or model ports of other behavior models.

#### Note

Mapping model ports in an unfavorable way might reduce the run-time performance. For optimum run-time performance, it is therefore recommended to observe specific mapping rules if you map model ports of BSC files to model ports of other behavior models. Refer to [Model Communication Recommendations for Model Port Blocks with Structured Data Ports \(ConfigurationDesk Real-Time Implementation Guide\)](#).

## How to Generate Bus Simulation Containers

### Objective

To use the bus communication you configured with the Bus Manager independently from the Bus Manager, you can generate bus simulation container (BSC) files.

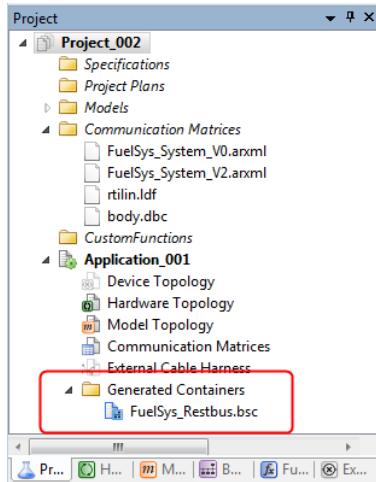
### Basics

For basic information, refer to [Basics on Bus Simulation Containers](#) on page 253.

---

<b>Preconditions</b>	<ul style="list-style-type: none"><li>▪ The ConfigurationDesk application you want to generate the BSC files for is active.</li><li>▪ Bus communication is configured in one or more bus configurations.</li><li>▪ If function ports of a bus configuration are mapped to model ports, the model ports are part of exactly one Simulink implementation container.</li><li>▪ A separate application process is available for each Simulink implementation container and no other <a href="#">model implementations</a> are assigned to the application process.</li><li>▪ Conflicts of the <b>Generate no code</b> category must not exist for bus configurations you want to include in the BSC files.</li><li>▪ Conflicts of the <b>Abort BSC file generation</b> category must not exist for application processes you want to generate the BSC files for.</li></ul>
<b>Method</b>	<p><b>To generate bus simulation containers</b></p> <ol style="list-style-type: none"><li>1 Switch to the Buses view set if necessary.</li><li>2 On the Home ribbon, click .</li></ol>
<b>Result</b>	<p>The generation of BSC files starts and the Message Viewer provides information on the generation progress. Each available application process is analyzed to check whether its configuration is valid for generating a BSC file. For each application process that passes the analysis, one BSC file is generated.</p> <p><b>Note</b></p> <div style="background-color: #f0f0f0; padding: 10px;"><p>If inconsistencies are detected during BSC file generation, you are informed about them in the Message Viewer. Due to these inconsistencies, BSC file generation might abort for the affected application processes.</p></div> <p>If the BSC file generation aborts due to multiple models that are assigned to one application process, you must change the assignment. Refer to <a href="#">How to Assign Behavior Models to Separate Application Processes</a> on page 344.</p>

The generated BSC files are displayed in the Project Manager below the Generated Containers node of the active application.



#### Next steps

- You can import the generated BSC files to the VEOS Player to implement the bus communication in an offline simulation application. For more information, refer to [Basics on Importing Bus Simulation Containers \(BSCs\) \(VEOS Manual\)](#).
- You can import the generated BSC files to ConfigurationDesk to implement the bus communication in a real-time application. For more information, refer to [Working with Bus Simulation Containers \(ConfigurationDesk Real-Time Implementation Guide\)](#)



# Working Without Behavior Models

## Where to go from here

## Information in this section

Basics on Working Without Behavior Models.....	263
How to Create Application Processes that Provide Default Tasks.....	264
How to Manually Assign Bus Configurations to Application Processes.....	265

## Basics on Working Without Behavior Models

### Overview

To implement bus communication in bus simulation containers, you can work without behavior models. If you do this, you must manually create application processes and tasks, and assign the bus configurations to the application processes.

### Introduction to application processes and tasks

Application processes are structuring elements that contain tasks. Tasks are pieces of code whose execution is controlled by the simulation platform, e.g., by the real-time operating system (RTOS). Tasks are required during the execution of [executable applications](#).

For more information, refer to [Modeling Executable Applications and Tasks \(ConfigurationDesk Real-Time Implementation Guide\)](#).

### Creating application processes when working without a behavior model

When you work without a behavior model, application processes are not created automatically. Instead, you must manually create application processes that provide default tasks. The default tasks are required in addition to the tasks that are created when you configure bus communication with the Bus Manager.

For each ConfigurationDesk application, you must create at least one application process. If you create more application processes, you can assign bus configurations to different application processes. When you generate bus simulation containers, one BSC file is generated for each of these application processes.

Refer to [How to Create Application Processes that Provide Default Tasks](#) on page 264.

#### Assigning bus configurations to application processes

Each bus configuration must be assigned to one application process. When you work without a behavior model, bus configurations are not assigned to application processes automatically. Instead, you must manually assign each bus configuration to an available application process. Refer to [How to Manually Assign Bus Configurations to Application Processes](#) on page 265.

## How to Create Application Processes that Provide Default Tasks

#### Objective

To work without a behavior model, you must manually create application processes that provide default tasks.

#### Basics

For basic information, refer to [Basics on Working Without Behavior Models](#) on page 263.

#### Method

##### To create application processes that provide default tasks

- 1 Open the Task Configuration table, for example, via Switch Controlbars on the View ribbon.
  - 2 In the Task Configuration table, right-click an empty area and select New - Application Process (Providing Default Task) from the context menu.
- An application process and a default task are created and displayed in the Task Configuration table.

Name	Priority	DAQ Raster ...	Real...	Period	Offset	Numb...	Jitter and L...
ApplicationProcess_1	0	Periodic Task 1	<input checked="" type="checkbox"/>	0.001	0	0	Standard (f...
Periodic Task 1							

#### Tip

You can rename application processes, e.g., in the Name column of the Task Configuration table. This is especially useful if you generate bus simulation containers because the generated BSC files are named after the related application processes.

- 3 Repeat the step above to create as many application processes as required.

For information on the required number, refer to [Basics on Working Without Behavior Models](#) on page 263.

**Result**

You created application processes that provide default tasks.

**Next step**

You can now manually assign bus configurations to the created application processes. Refer to [How to Manually Assign Bus Configurations to Application Processes](#) on page 265.

## How to Manually Assign Bus Configurations to Application Processes

**Objective**

To manually assign bus configurations to application processes.

**Basics**

For basic information, refer to [Basics on Working Without Behavior Models](#) on page 263.

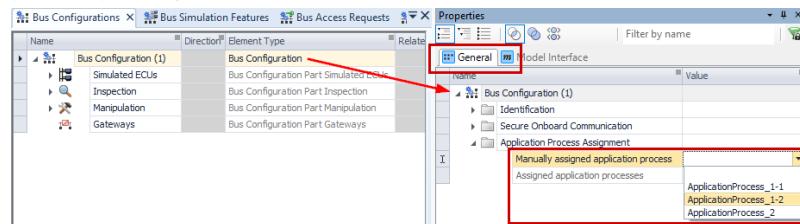
**Preconditions**

An application process is available in the active ConfigurationDesk application.

**Method**

### To manually assign bus configurations to application processes

- 1 Select a bus configuration, for example, in the Bus Configurations table.
- 2 On the General page of the Properties Browser, select an available application process from the list of the Manually assigned application process property.



The bus configuration is assigned to the selected application process.

- 3 Repeat the steps above to manually assign further bus configurations to application processes.

**Result**

You manually assigned bus configurations to the selected application processes. The Assigned application processes property displays all the application processes the related bus configuration is assigned to.

Name	Value
Bus Configuration (1)	
Identification	
Secure Onboard Communication	
Application Process Assignment	
Manually assigned application process	ApplicationProcess_1-2
Assigned application processes	ApplicationProcess_1-2

If the bus configuration is assigned to more than one application process, conflicts occur. For example, this can happen if the bus configuration is mapped to a behavior model and the behavior model is assigned to a different application process. Refer to [Basics on Bus Manager-Related Conflicts](#) on page 99.

# Modifying Communication Matrices

## Where to go from here

## Information in this section

Basics on Modifying Communication Matrices.....	267
Adding ISignal IPDUs to CAN Channels.....	271
Adding ISignals to CAN PDUs.....	272
Adding Cyclic Timing Elements to Basic PDUs.....	274
Configurable Settings of Communication Clusters.....	276
Configurable Settings of Frames.....	277
Configurable Settings of PDUs.....	281
Configurable Settings of ISignals.....	284

## Basics on Modifying Communication Matrices

### Introduction

You can modify [communication matrices](#) within the active ConfigurationDesk application. You can add user-defined elements and specify user-defined settings for configurable elements. For example, this allows you to do the following:

- Resolve conflicts in the active ConfigurationDesk application that result from conflicting communication matrix settings.
- Work with a preliminary version of a communication matrix.
- Work with the `Basic_ComMatrix.arxml` communication matrix.

If you do not have a communication matrix, you can use the `Basic_ComMatrix.arxml` communication matrix as a starting point: You can add user-defined elements to the communication matrix and specify user-defined settings for elements. Then, you can assign these elements to bus

configurations to implement and configure basic CAN communication in the signal chain.

The **Basic\_ComMatrix.arxml** file is available in the **BusManagerDemo** subfolder of the [Documents folder](#).

## Overview

**Adding user-defined elements** You can add the following user-defined elements to a communication matrix:

- [ISignal IPDUs](#)

Refer to [Adding ISignal IPDUs to CAN Channels](#) on page 271.

- [ISignals](#)

Refer to [Adding ISignals to CAN PDUs](#) on page 272.

- Cyclic timings

Refer to [Adding Cyclic Timing Elements to Basic PDUs](#) on page 274.

**Configurable settings of communication matrix elements** The following table provides an overview of the communication matrix elements that provide configurable settings:

Element	Configurable Settings	Further Information
Communication cluster	<ul style="list-style-type: none"> <li>▪ Baud rate</li> <li>▪ Data phase baud rate (only for CAN clusters)</li> </ul>	<a href="#">Configurable Settings of Communication Clusters</a> on page 276
Frame	<ul style="list-style-type: none"> <li>▪ Payload length</li> <li>▪ CAN frame triggering: <ul style="list-style-type: none"> <li>▪ Identifier</li> <li>▪ Extended addressing</li> <li>▪ CAN FD frame support</li> <li>▪ Bit rate switch</li> </ul> </li> <li>▪ LIN frame triggering: <ul style="list-style-type: none"> <li>▪ Identifier</li> <li>▪ Checksum type</li> </ul> </li> </ul>	<a href="#">Configurable Settings of Frames</a> on page 277
PDU	<ul style="list-style-type: none"> <li>▪ Name (only for user-defined ISignal IPDUs)</li> <li>▪ Payload length</li> <li>▪ Bit pattern for unused bits</li> <li>▪ Cyclic transmission: <ul style="list-style-type: none"> <li>▪ Time offset</li> <li>▪ Time period</li> </ul> </li> </ul>	<a href="#">Configurable Settings of PDUs</a> on page 281
ISignal	<ul style="list-style-type: none"> <li>▪ Name (only for user-defined ISignals)</li> <li>▪ Length</li> <li>▪ Initial value</li> <li>▪ Base data type for the coded and physical signal type</li> <li>▪ Category</li> <li>▪ Computation scale: Numerators and denominators (only for computation methods that have the Interpreted category set to LINEAR)</li> <li>▪ ISignal-to-IPDU mapping: Start bit position and endianness</li> </ul>	<a href="#">Configurable Settings of ISignals</a> on page 284

## Effects of modifying communication matrices

**General effects** Modifying a communication matrix has the following effects:

- When you add elements to a communication matrix, required related elements are added automatically. For example, if you add an ISignal IPDU, related elements such as a frame and a frame triggering are added as well.
- When you specify user-defined settings for configurable elements, the changes apply to the communication matrix element and all its instances. For example, if you modify the length of a TX ISignal, the changes apply to both directions of the ISignal (TX and RX), and all the ISignal instances in all bus configurations of the active ConfigurationDesk application.
- The modifications apply only to the communication matrix in the active ConfigurationDesk application. The original communication matrix and inactive applications remain unchanged.

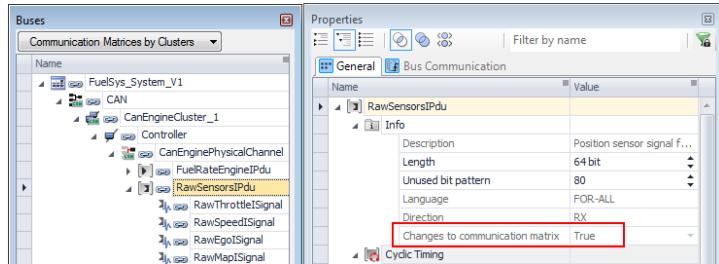
Depending on the specifications in the communication matrix and/or the use of modified elements in bus configurations, the changes might result in conflicting settings. For more information, refer to [Basics on Bus Manager-Related Conflicts](#) on page 99.

**Effects when deleting or replacing communication matrices** The modifications are lost if you delete a communication matrix from the application. Replacing an assigned communication matrix in a bus configuration has the following effects on the modified elements that are assigned to the bus configuration:

- Added ISignal IPDUs and ISignals are transferred to the newly assigned communication matrix only if they fulfill all replacing conditions. Otherwise, they are lost. For information on the replacing conditions, refer to [Basics on Replacing Assigned Communication Matrices](#) on page 291.
- Added cyclic timings and user-defined settings of configurable elements are lost, i.e., the changes are not transferred to the newly assigned communication matrix even if it contains identical elements.

## Identifying modified communication matrix elements

You can identify modified communication matrix elements via the Properties Browser: Elements that were added or modified by the user have the Changes to communication matrix property set to True.



You can also access modified communication matrix elements via ConfigurationDesk's automation interface and generate an overview of all the modified elements of the active ConfigurationDesk application. For an example, refer to [Examples of Automating Bus Manager Features \(ConfigurationDesk Automating Tool Handling\)](#).

**Filter for modified communication matrix elements**

You can filter the communication matrices of the active application for modified elements. The Filter for Changes to Communication Matrices is available via the context menu of the Buses Browser and on the Home ribbon. When the filter is active, the Buses Browser displays only the modified communication matrix elements and their related higher-level elements.

For an overview of further filters for communication matrices, refer to [Filtering communication matrix elements](#) on page 62. For details on using the filters, refer to [Using Display Filters \(ConfigurationDesk Real-Time Implementation Guide\)](#).

**Undo modifications**

You can undo the modifications of communication matrices. The Undo All Changes command of the Home ribbon lets you undo all the modifications of all the communication matrices of the active application. The Undo Changes to Communication Matrix command of a selected element in the Properties Browser lets you undo the modifications of only this element as follows:

- If the element was added to the communication matrix, it is deleted from the communication matrix, including the elements that were added automatically (e.g., the frame of an added user-defined ISignal IPDU).
- If the selected element was not added but has user-defined settings, the settings are reset to the default values, i.e., the original values specified in the communication matrix or the system default values.

**Restrictions for working with modified communication matrices**

You can work with a modified communication matrix only in the active ConfigurationDesk application. You cannot reuse the modifications in other ConfigurationDesk applications or transfer the changes to the original communication matrix.

**Tip**

If you need the modifications of a communication matrix in more than one ConfigurationDesk application, use automation scripts for modifying a communication matrix. This reduces the effort for modifying the communication matrix in each required ConfigurationDesk application significantly and reduces the risk of failures. For an example of modifying communication matrices via automation scripts, refer to [Examples of Automating Bus Manager Features \(ConfigurationDesk Automating Tool Handling\)](#).

If you assign user-defined ISignal IPDUs or ISignals to bus configurations, some limitations apply for importing working views. Refer to [Limitations for importing working views that contain Bus Configuration function blocks](#) on page 330.

**Related topics****Basics**

[Working with Communication Matrices](#).....58

**References**

Filter for Changes / Filter for Changes to Communication Matrices  
 (ConfigurationDesk User Interface Reference )  
 Undo All Changes (ConfigurationDesk User Interface Reference )  
 Undo Changes to Communication Matrix (ConfigurationDesk User Interface Reference )

## Adding ISignal IPDUs to CAN Channels

**Introduction**

You can add [ISignal IPDUs](#) to CAN channels that are specified in a communication matrix. For example, you can use these user-defined ISignal IPDUs to configure ISignal IPDUs in bus configurations that are not available yet in the current version of the communication matrix.

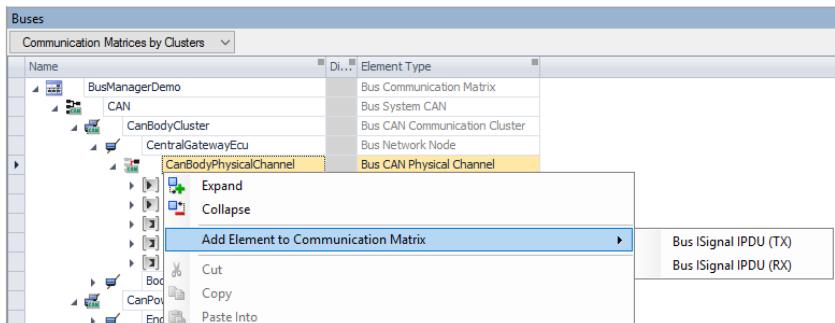
**Conditions for adding ISignal IPDUs**

For adding ISignal IPDUs to a communication matrix, the following conditions apply:

- You can add ISignal IPDUs only to CAN channels.
- You can add an ISignal IPDU to one CAN channel only, i.e., you cannot use an added ISignal IPDU on multiple channels.

**Adding ISignal IPDUs**

You can add [TX](#) and [RX](#) ISignal IPDUs via context menu when you select a CAN channel in the **Communication Matrices by Clusters** view of the **Buses Browser**, as shown in the following example.

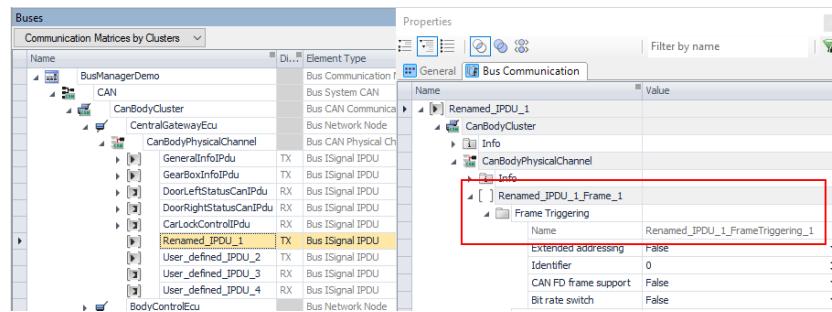


### Effects of adding ISignal PDUs

When you add an ISignal IPDU, the following elements are added as well:

- A PDU triggering
- A cyclic timing including a time period and a time offset
- A frame
- A frame triggering

The ISignal IPDU is added with a unique default name, i.e., `User_defined_IPDU_<number>`. The names of the frame and the frame triggering are derived from the PDU name. You can rename the PDU, for example, in the Buses Browser or the Properties Browser. If you do this, the frame and the frame triggering are renamed accordingly, as shown in the following example.



The PDU and the related elements are added with default settings. You can change some of these settings. For more information, refer to:

- [Configurable Settings of Frames](#) on page 277
- [Configurable Settings of PDUs](#) on page 281

### Restrictions for adding ISignal PDUs

You cannot add instances of an ISignal IPDU to a communication matrix. For example, you cannot add a TX instance and an RX instance of an ISignal IPDU to a communication matrix. Instead, you must add two separate ISignal PDUs, one TX ISignal IPDU and one RX ISignal IPDU.

## Adding ISignals to CAN PDUs

### Introduction

You can add [ISignals](#) to CAN PDUs. For example, you can use these user-defined ISignals to configure ISignals in bus configurations that are not available yet in the current version of the communication matrix.

### Conditions for adding ISignals

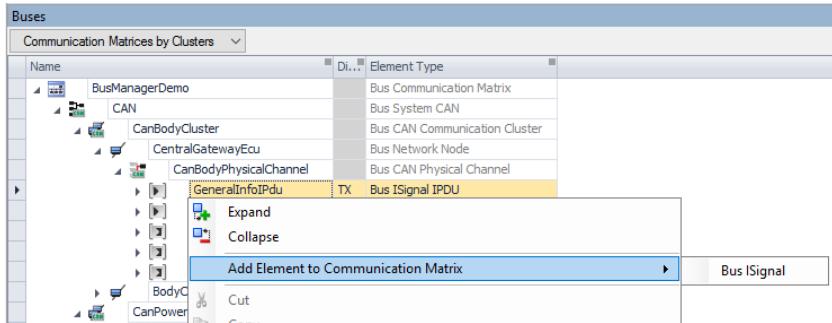
For adding ISignals to a communication matrix, the following conditions apply:

- You can add ISignals to a PDU only if the PDU is exchanged via exactly one communication cluster, which is a CAN cluster.

- You can add ISignals only to PDUs that can contain ISignals. For example, you can add ISignals to basic PDUs such as ISignal IPDUs or NMPDUs but not to container PDUs, multiplexed PDUs, or secured PDUs.

## Adding ISignals

You can add ISignals via context menu when you select a supported CAN PDU in the Communication Matrices by Clusters view of the Buses Browser, as shown in the following example.



The direction of the ISignal (TX or RX) is set automatically according to the direction of the selected PDU.

## Effects of adding ISignals

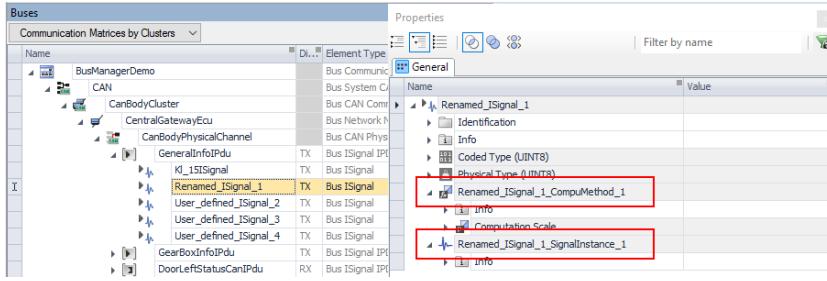
If multiple instances of a PDU are available in the communication matrix, ISignals are added to all instances of the related PDU, regardless of the direction (TX or RX). For example, if you add an ISignal to a TX PDU while an RX instance of the PDU is available in the communication matrix as well, the ISignal is added to both instances. The direction of the ISignal is adjusted automatically.

However, the ISignal is not assigned to bus configurations automatically, even if an instance of the PDU is assigned. If you want to use the ISignal in a bus configuration, you must assign it manually for each required PDU instance.

When you add an ISignal, the following elements are added as well:

- The base data types for the coded and physical signal types
- A linear computation method including a computation scale
- An ISignal-to-IPDU mapping

The ISignal is added with a unique default name, i.e., User\_defined\_ISignal\_<number>. The names of the computation method and the ISignal-to-IPDU mapping are derived from the ISignal name. You can rename the ISignal, for example, in the Buses Browser or the Properties Browser. If you do this, the computation method and the ISignal-to-IPDU mapping are renamed accordingly, as shown in the following example.



The ISignal and the related elements are added with default settings. You can change some of these settings. For more information, refer to [Configurable Settings of ISignals](#) on page 284.

## Adding Cyclic Timing Elements to Basic PDUs

### Introduction

You can add cyclic timing elements to [basic PDUs](#). For example, you can use these user-defined cyclic timing elements to specify a cyclic transmission of the related PDU if it is not specified yet in the current version of the communication matrix.

### Conditions for adding cyclic timing elements

For adding cyclic timing elements to a PDU, the following conditions apply:

- You can add cyclic timing elements only to basic PDUs.
- You can add a cyclic timing element to a basic PDU only if it is not available yet for the PDU.
- You can add a missing cyclic timing element only if the available cyclic timing elements are not grouped in the Properties Browser.

If the communication matrix specifies multiple cyclic timings for a PDU, the cyclic timing elements are grouped by default. To add missing elements in this case, you must ungroup the elements by selecting the List all elements intersection mode ( of the Properties Browser.

### Overview of the cyclic timing elements

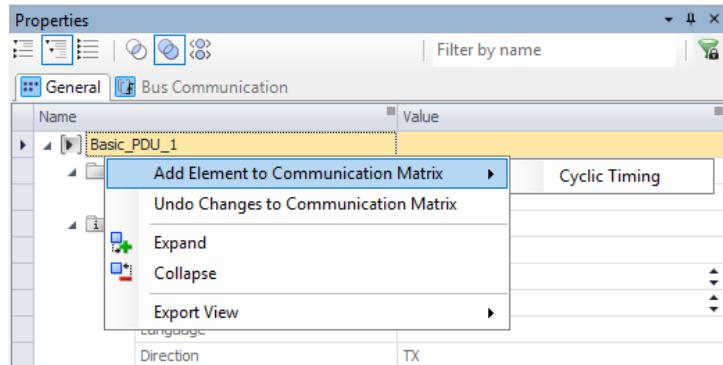
You can add the following cyclic timing elements to a basic PDU:

- **Cyclic Timing**  
If no cyclic transmission is specified yet for a PDU, you can add the Cyclic Timing element.
- **Time Offset**  
If a cyclic transmission is specified for a PDU but no time offset (delay time), you can add the Time Offset element.
- **Time Period**  
If a cyclic transmission is specified for a PDU but no time period (cycle time), you can add the Time Period element.

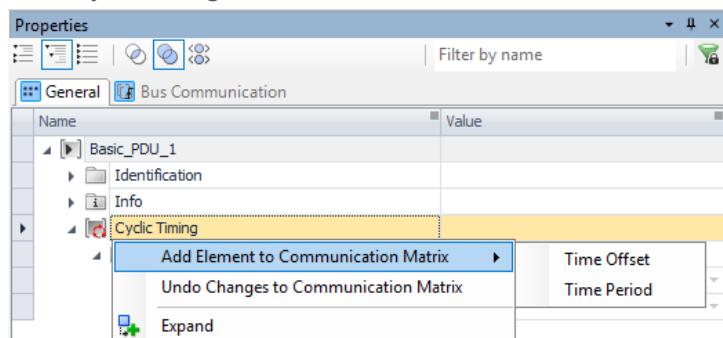
## Adding cyclic timing elements

When you select a basic PDU, for example, in the Buses Browser, you can add a cyclic timing element on the General page of the Properties Browser as follows:

- You can add a missing Cyclic Timing element via context menu of the basic PDU node.



- You can add a missing Time Offset or Time Period element via context menu of the Cyclic Timing node.



## Effects of adding cyclic timing elements

If multiple instances of a PDU are available, cyclic timing elements are added to all instances of the related basic PDU, regardless of the direction (TX or RX) and the location (communication matrix or bus configuration). For example, if you add cyclic timing elements to a TX basic PDU while TX and RX instances of the PDU are available in the communication matrix and/or in bus configurations, the elements are added to all the instances of the PDU, including the RX instances and the instances in the bus configurations.

When you add a Cyclic Timing element, a Time Offset and Time Period element is added automatically. When you add a Time Offset or Time Period element, only the related element is added.

The elements are added with default settings. You can change some of these settings. For more information, refer to [Configurable Settings of PDUs](#) on page 281.

**Restrictions for cyclic timing elements**

For the Cyclic Timing element, the Time Period element is mandatory while the Time Offset element is optional. If required, you can delete the Time Offset element but you must not delete the Time Period element.

## Configurable Settings of Communication Clusters

### Overview

Communication clusters  provide the following configurable settings:

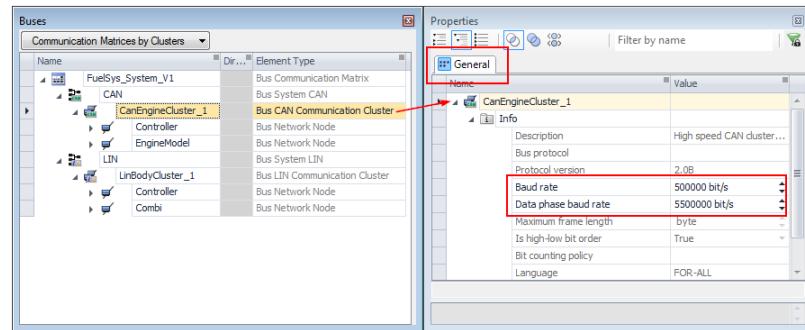
- Baud rate
- Data phase baud rate (only for CAN clusters)

### Accessing the configurable settings

In the ConfigurationDesk application, you can access the configurable settings in the following ways:

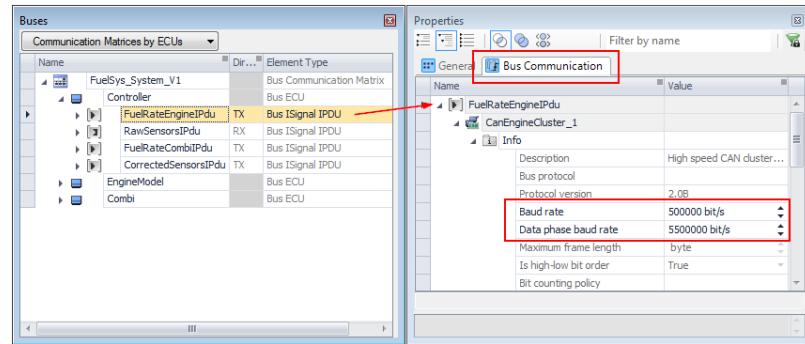
- Select a communication cluster node in the **Communication Matrices by Clusters** view of the Buses Browser or in the Bus Configurations table.

You can access the available baud rates on the General page of the Properties Browser.



- Select a PDU node in the Buses Browser or in one of the following tables:
  - Bus Configurations table
  - Bus Simulation Features table
  - Bus Inspection Features table
  - Bus Manipulation Features table

You can access the available baud rates of the related cluster on the Bus Communication page of the Properties Browser. The specified baud rates apply not only to the selected PDU, but to all PDUs of the cluster.



### Specifying the baud rates

The Properties Browser provides the following properties for specifying the baud rates of communication clusters.

**Baud rate** Lets you specify the baud rate (in bit/s) that applies to all PDUs of the cluster. The valid range depends on the cluster's bus protocol and the hardware that is used for the bus communication.

**Data phase baud rate** Available only for CAN clusters. Lets you specify the baud rate (in bit/s) that applies to the data phase of CAN FD frames. For more information, refer to [CAN FD protocol](#) on page 47. The valid range depends on the hardware that is used for the bus communication.

## Configurable Settings of Frames

### Overview

Frames provide the following configurable settings:

- Payload length
- CAN frame triggering: Extended addressing, identifier, CAN FD support, and bit rate switch
- LIN frame triggering: Identifier and checksum type

Configuring these settings affects the related frame independently of its direction, i.e., changes you make for TX instances of a frame affect its RX instances as well and vice versa.

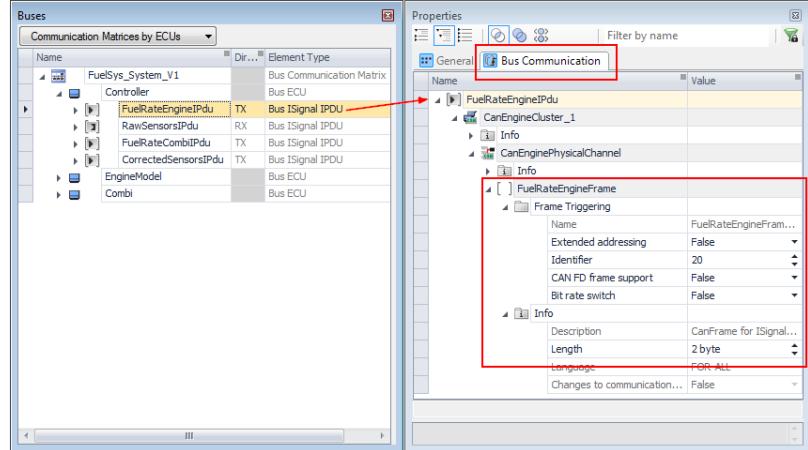
### Accessing the configurable settings

In the ConfigurationDesk application, you can access the configurable settings of a frame only when you select a PDU that is included in the frame. To access the configurable settings, you can select a PDU in the Buses Browser or in one of the following tables:

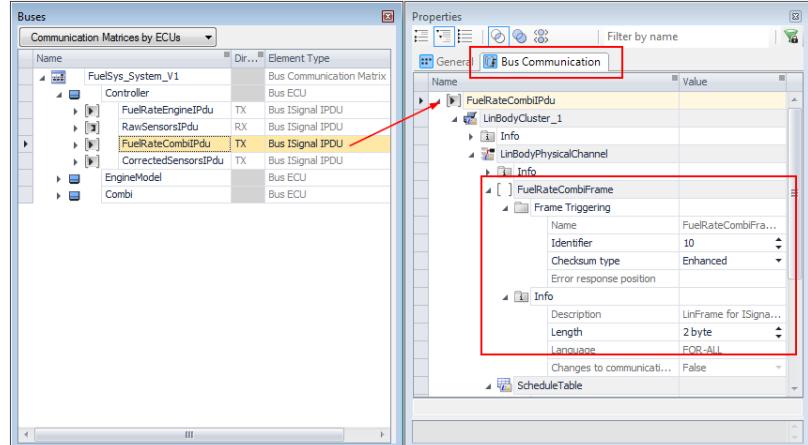
- Bus Configurations table
- Bus Simulation Features table
- Bus Inspection Features table
- Bus Manipulation Features table

For the selected PDU, the Bus Communication page of the Properties Browser provides properties for configuring the payload length and the frame triggering of the related frame. The configurable frame triggering properties differ for CAN and LIN frames, as shown in the following illustrations:

- Configurable properties available for CAN frames:



- Configurable properties available for LIN frames:



### Tip

When a frame is exchanged via multiple communication clusters, multiple frame triggerings apply to the frame. However, in most cases a PDU is selected in the context of only one communication cluster. In this case, the Properties Browser displays only one frame triggering of the frame. To identify the other frame triggerings that also apply to the frame, filter the Buses Browser for the PDU name, select all displayed PDU instances, and select the intersection mode in the Properties Browser, for example.

Even though you access the frame properties via a PDU, the specified settings apply to the frame itself, i.e., if several PDUs are included in one frame (e.g., via a container IPDU [?](#)), you must configure the frame only for one of these PDUs.

The specified settings are automatically available for all the other PDUs that are included in this frame.

#### Specifying the payload length of CAN and LIN frames

When you select a PDU, the Bus Communication page of the Properties Browser provides the following property for specifying the payload length of the related frame.

**Length** Lets you specify the payload length of CAN and LIN frames in bytes. The valid range depends on the bus system and bus protocol:

- Classic CAN frames: 0 byte ... 8 bytes.
- CAN FD frames: 0 byte ... 64 bytes.
- LIN frames: 1 byte ... 8 bytes.

Points to note regarding CAN FD:

- The CAN FD value range is available only if all the frame triggerings that apply to the frame support CAN FD.
- Not all values in the value range are valid. If you specify an invalid payload length according to CAN FD, the Bus Manager extends the payload length to the next valid CAN FD length by adding padding bytes. Refer to [CAN FD protocol](#) on page 47.

#### Tip

If you want to specify the frame length for individual CAN frame instances, you can use the Frame Access or Frame Length bus configuration feature. Refer to [Accessing CAN Frame Settings](#) on page 190 or [Manipulating the Payload Length of CAN Frames](#) on page 197, respectively.

#### Specifying the frame triggering of CAN frames

When you select a CAN PDU, the Bus Communication page of the Properties Browser provides the following properties for specifying the frame triggering of the related CAN frame.

**Extended addressing** Lets you specify whether the CAN frame uses a standard identifier format (11-bit) or extended identifier format (29-bit):

- True: Extended identifier format
- False: Standard identifier format

**Identifier** Lets you specify the identifier of the CAN frame. The identifier of a CAN frame arbitrates and prioritizes the frame. The valid value range depends on the identifier format (specified via the Extended addressing property):

- Extended identifier format: valid Identifier range 0 ...  $2^{29} - 1$
- Standard identifier format: valid Identifier range 0 ...  $2^{11} - 1$

**CAN FD frame support** Lets you specify whether CAN FD is supported for the frame:

- True: CAN FD is supported for the frame, i.e., the payload length of the frame can be up to 64 bytes and the data phase of the frame can be transmitted with a higher bit rate. For more information, refer to [CAN FD protocol](#) on page 47.
- False: CAN FD is not supported for the frame, i.e., the frame is treated as a classic CAN frame.

**Bit rate switch** Lets you specify the bit rate used to transmit the data phase of the frame. The specified setting applies only to CAN FD frames. It has no effects on classic CAN frames. You can select one of the following values:

- True: The data phase of the frame is transmitted with the bit rate that is specified for the Data phase baud rate property of the related CAN communication cluster.
- False: The data phase of the frame is transmitted with the bit rate that is specified for the Baud rate property of the related CAN communication cluster.

The baud rate properties of communication clusters can be configured. For more information, refer to [Configurable Settings of Communication Clusters](#) on page 276.

#### Tip

If you want to specify the frame triggering for individual CAN frame instances, you can use the Frame Access bus configuration feature. Refer to [Accessing CAN Frame Settings](#) on page 190.

### Specifying the frame triggering of LIN frames

When you select a LIN PDU, the Bus Communication page of the Properties Browser provides the following properties for specifying the frame triggering of the related LIN frame.

**Identifier** Lets you specify the identifier value (ID) of the LIN frame. The identifier is part of the protected identifier (PID) of a LIN frame header and assigns the header to a frame response. You can specify an ID in the range 0 ... 63. The specified ID must be unique for each LIN frame that the communication matrix assigns to the same LIN channel.

According to the LIN specifications, the specified ID determines the frame type as follows:

- ID 0 ... 59: Data-carrying frame (e.g., unconditional frame or event-triggered frame)
- ID 60: Diagnostic frame (master request frame)
- ID 61: Diagnostic frame (slave response frame)
- ID 62: Reserved for OEM-specific LIN communication
- ID 63: Reserved for future use

**Checksum type** Lets you specify the checksum type of the LIN frame. The checksum is part of the frame response and validates the frame response. You can select one of the following values:

- Classic
- Enhanced

For more information on the checksum types, refer to [Aspects of Supported LIN Bus Features](#) on page 52.

## Configurable Settings of PDUs

### Overview

[PDUs](#) provide the following configurable settings:

- Name (only for user-defined [ISignal IPDUs](#))
- Payload length
- Bit pattern for unused bits (not for [Container IPDUs](#) with dynamic container layout)
- Cyclic transmission: Cyclic timing, time period, and time offset (only for [basic PDUs](#))

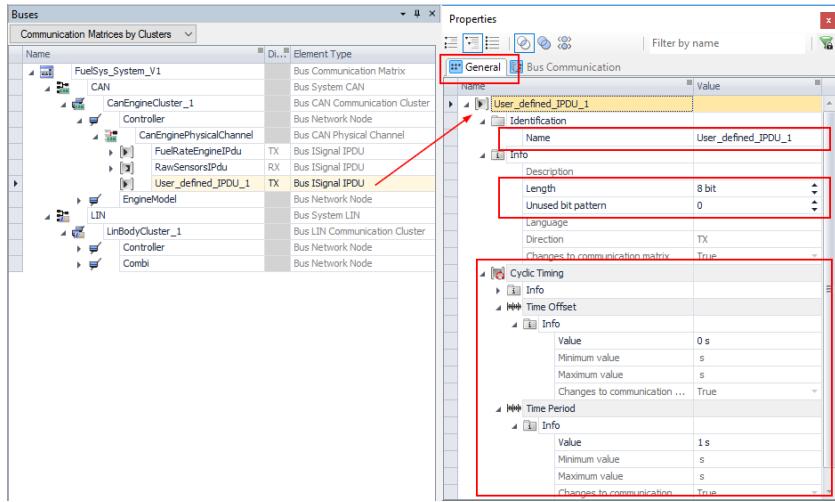
Configuring these settings affects the related PDU independently from its direction, i.e., changes you make for TX instances of a PDU affect its RX instances as well and vice versa.

### Accessing the configurable settings

In the ConfigurationDesk application, you can access the configurable settings when you select a PDU in the Buses Browser or in one of the following tables:

- Bus Configurations table
- Bus Simulation Features table
- Bus Inspection Features table
- Bus Manipulation Features table

The General page of the Properties Browser provides properties for configuring the PDU's name, the payload length, the unused bit pattern, and the cyclic transmission.



#### Specifying the name of user-defined ISignal IPDUs

When you select a user-defined ISignal IPDU, the General page of the Properties Browser provides the following property for specifying the PDU's name.

**Name** Lets you specify the name of a user-defined ISignal IPDU. The name can have a maximum length of 128 characters and must be unique for all PDUs of the same ECU and direction (TX or RX). It is recommended to use only alphabetic and numeric ASCII characters and underscores for the name. For more information, refer to [Limitations for element names](#) on page 329.

When you specify the name, the names of the related frame and frame triggering are adjusted accordingly. For more information, refer to [Adding ISignal IPDUs to CAN Channels](#) on page 271.

#### Specifying the payload length of PDUs

When you select a PDU, the General page of the Properties Browser provides the following property for specifying the PDU's payload length.

**Length** Lets you specify the payload length of the PDU in bytes. The valid range depends on the bus system and bus protocol as follows:

- Classic CAN and LIN: 0 ... 8 bytes
- CAN FD: 0 ... 64 bytes
- CAN J1939: 0 ... 1785 bytes

However, the CAN FD value range is available only if CAN FD is supported by all the frames in which the PDU is included, i.e.:

- If the PDU is included in CAN and LIN frames, the CAN FD value range is not available. This applies even if all the CAN frames support CAN FD.
- If the PDU is included only in CAN frames, all the frame triggerings of all the frames must support CAN FD. You can specify the CAN FD support. Refer to [Configurable Settings of Frames](#) on page 277.

Moreover, the valid maximum payload length depends on various factors, for example:

- The usable payload length of the related frame.
- The usable payload length of a related multiplexed IPDU.
- The usable payload length of a related container IPDU.

#### Tip

If you specify a payload length that is invalid in the context in which the PDU is used, conflicts occur. The conflicts can help you specify a valid payload length for the specific context. Refer to [Basics on Bus Manager-Related Conflicts on page 99](#).

If you modify the PDU length while a related PDU Raw Data function port exists, the function port is affected in the following ways:

- The data width of the function port depends on the PDU length. Therefore, the function port's data width adjusts to the modified PDU length.
- The data width determines the vector size of the function port's Initial value and Initial substitute value properties. This has the following effects on the specified values:
  - If the data width is reduced, the vector size of both properties reduces accordingly. The vector elements are deleted from right to left and the values specified for these elements are lost, as shown in the following example.

FuelRateEngineIPdu_RawData	
Initialization and Termin...	
Initial value	10; 15; 20; 25; 30; 35
Stop value	0; 0; 0; 0; 0; 0
Behavior Model	
Test Automation Support	
Activate test autom...	Enabled
Initial switch setting	Model signal
Initial substitute value	10; 15; 20; 25; 30; 35
Info	
Data width	6

→

FuelRateEngineIPdu_RawData	
Initialization and Termin...	
Initial value	10; 15
Stop value	0; 0
Behavior Model	
Test Automation Support	
Activate test autom...	Enabled
Initial switch setting	Model signal
Initial substitute value	10; 15
Info	
Data width	2

- If the data width is increased, the vector size of both properties increases accordingly. The new vector elements are added to the right of the existing vector elements. The default value of the new vector elements is 0.

For more information on PDU Raw Data function ports, refer to [Accessing the Payload of PDUs in Raw Data Format on page 157](#).

#### Specifying bit patterns for unused PDU bits

To prevent unintended behavior, unused PDU bits are filled with a specific bit pattern. AUTOSAR-compliant communication matrices can specify a bit pattern for each multiplexed IPDU, NMPDU, and ISignal IPDU for this purpose. If a communication matrix does not specify a bit pattern for a PDU (e.g., because it is a container IPDU with dynamic container layout or the communication matrix does not comply with AUTOSAR), the Bus Manager uses a default bit pattern: For J1939-compliant PDUs, the default bit pattern is 255. For all other PDUs, the default bit pattern is 0.

For [multiplexed IPDUs](#), [basic PDUs](#), and container IPDUs with static container layout, you can specify a user-defined bit pattern on the General page of the Properties Browser via the following property.

**Unused bit pattern** Lets you specify a bit pattern for the selected PDU in the range 0 ... 255.

#### Specifying cyclic transmission of basic PDUs

[Basic PDUs](#) provide Cyclic Timing, Time Period, and Time Offset elements that let you specify cyclic transmission: You can add missing elements and specify user-defined settings for configurable properties.

For information on adding missing elements, refer to [Adding Cyclic Timing Elements to Basic PDUs](#) on page 274.

When you select a basic PDU, the General page of the Properties Browser can provide the following properties for specifying the cyclic transmission of the PDU.

**Value (Time Offset)** If the Time Offset element is available, its Value property lets you specify a time offset (delay time) for the cyclic transmission of the selected PDU. You can specify the delay time in seconds. The specified delay time must be in the range 0 ... 3600 seconds.

**Value (Time Period)** If the Time Period element is available, its Value property lets you specify a time period (cycle time) for the cyclic transmission of the selected PDU. You can specify the cycle time in seconds. The specified cycle time must be in the range 0 ... 3600 seconds.

#### Tip

If you want to specify cyclic transmission for individual PDU instances, you can use the PDU Cyclic Timing Control bus configuration feature. Refer to [Controlling the Cyclic Timing of CAN PDUs](#) on page 159.

## Configurable Settings of ISignals

### Overview

[ISignals](#) provide the following configurable settings:

- Name (only for user-defined ISignals)
- Signal length
- Initial value
- Base data type for the coded and physical signal type
- Coded signal type: Category
- Computation scale: Numerators and denominators (only for computation methods that have the Interpreted category set to LINEAR)
- ISignal-to-IPDU mapping: Start bit position and endianness

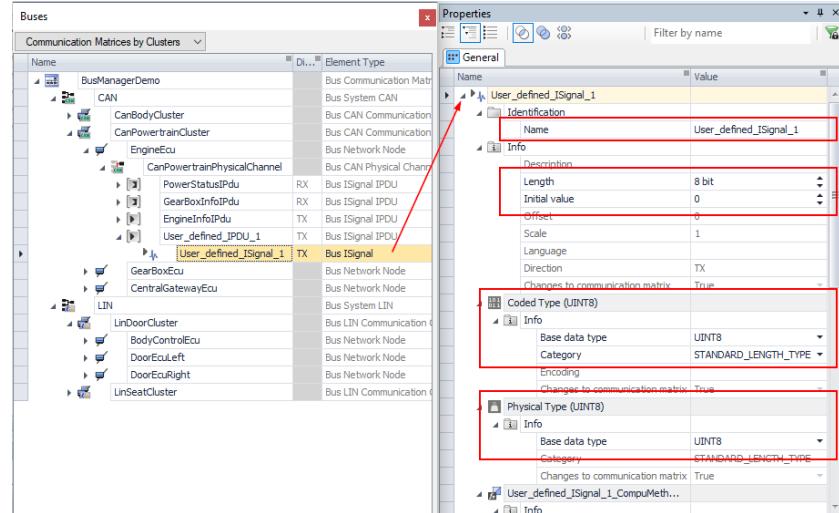
Configuring these settings affects the related ISignal independently from its direction, i.e., changes you make for TX instances of an ISignal affect its RX instances as well and vice versa.

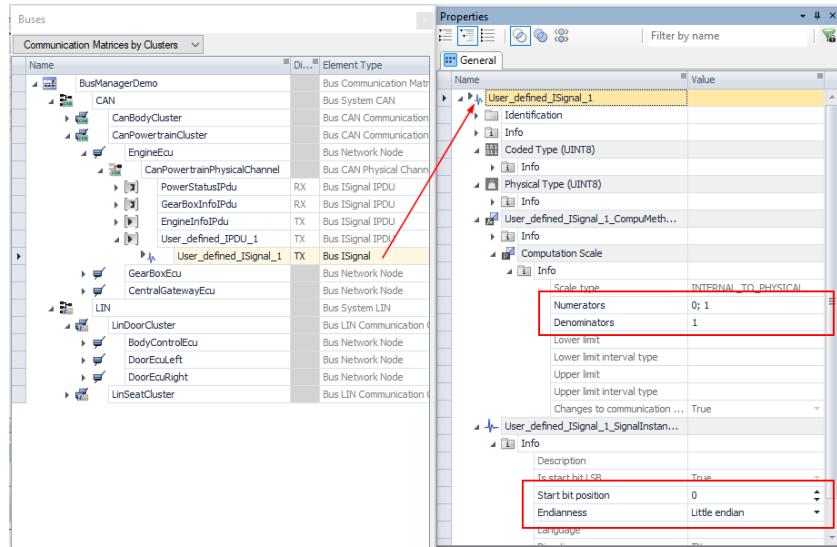
## Accessing the configurable settings

In the ConfigurationDesk application, you can access the configurable settings when you select an ISignal in the Buses Browser or in one of the following tables:

- Bus Configurations table
- Bus Simulation Features table
- Bus Inspection Features table
- Bus Manipulation Features table

The General page of the Properties Browser provides properties for accessing the configurable settings, as shown in the following examples. Depending on the specifications in the communication matrix, not all of the displayed properties are available for a selected ISignal.





### Specifying user-defined settings for ISignals

The Properties Browser provides the following properties for specifying user-defined settings for ISignals.

**Name** Available only for user-defined ISignals. Lets you specify the name of a user-defined ISignal. The name can have a maximum length of 128 characters and must be unique for all ISignals and ISignal groups of the same PDU. It is recommended to use only alphabetic and numeric ASCII characters and underscores for the name. For more information, refer to [Limitations for element names](#) on page 329.

When you specify the name, the names of the related computation method and the ISignal-to-IPDU mapping are adjusted accordingly. For more information, refer to [Adding ISignal IPDUs to CAN Channels](#) on page 271.

**Length** Lets you specify the signal length:

Signal Type	Range	Description
Standard ISignals (i.e., ISignals with Category set to STANDARD_LENGTH_TYPE)	1 bit ... 64 bits	<p>The valid signal length depends on the coded base data type of the ISignal:</p> <ul style="list-style-type: none"> <li>▪ If the coded base data type is an integer type (Int, UInt), the number of bits specified for the signal length must be equal or smaller than the number of bits specified for the coded base data type. Example: For a coded base data type of Int8, the valid signal length is 1 bit ... 8 bits.</li> <li>▪ If the number of bits specified for the signal length is smaller than the number of bits specified for the coded base data type and a signal value exceeds the value range of the signal length, the signal value is automatically saturated to the next valid value.</li> <li>▪ If the coded base data type is a float or double type, the number of bits specified for the signal length must be equal to the number of bits specified for the coded base data type. Example: For a coded base data type of Float64, the valid signal length is 64 bit.</li> </ul>
Array signals (i.e., ISignals with Category set to ARRAY_TYPE)	<ul style="list-style-type: none"> <li>▪ Classic CAN and LIN: 1 bit ... 64 bits</li> <li>▪ CAN FD: 1 bit ... 512 bits</li> </ul>	The specified length determines the vector size of the initial value and the data width of the ISignal Value function port. If required, the vector size and the data width are automatically adjusted.

Signal Type	Range	Description
	▪ J1939: 1 bit ... 1480 bits	However, the CAN FD value range is available only if CAN FD is supported by all the frames in which the related ISignal IPDU is included, i.e.: <ul style="list-style-type: none"> <li>▪ If the related ISignal IPDU is included in CAN and LIN frames, the CAN FD value range is not available. This applies even if all the CAN frames support CAN FD.</li> <li>▪ If the related ISignal IPDU is included only in CAN frames, all the frame triggerings of all the frames must support CAN FD. You can specify the CAN FD support. Refer to <a href="#">Configurable Settings of Frames</a> on page 277.</li> </ul>

**Initial value** Lets you specify the coded initial signal value:

Signal Type	Description
Standard ISignals (i.e., ISignals with Category set to STANDARD_LENGTH_TYPE)	The valid range depends on the specified coded base data type. If you specify an initial value that exceeds the valid range, the value is automatically saturated to the next valid value.
Array signals (i.e., ISignals with Category set to ARRAY_TYPE)	The initial value is a vector. The vector size is the signal length in bytes (e.g., for a signal length of 16 bits, the vector size is 2). The value range of each vector element is 0 ... 255.

The ISignal's initial value and the initial value of ISignal Value function ports influence each other:

- When you add an ISignal Value function port to an instance of the ISignal, the ISignal's initial value is used as the function port's initial value. If the ISignal's initial value exceeds the valid range of the function port's initial value, the function port's initial value is automatically saturated to the next valid value.

#### Tip

The valid range of the function port's initial value depends on the specified physical base data type of the ISignal.

- If you change the ISignal's initial value while a related ISignal Value function port already exists, the function port's initial value is not updated.
- If the ISignal's and function port's initial values differ, the function port's initial value is used at run time.

For more information on ISignal Value function ports, refer to [Working with ISignal Values](#) on page 139.

**Base data type (coded type)** Lets you specify the base data type for the coded signal value. For an introduction to coded and physical signal values, refer to [Signal Conversion by the Bus Manager](#) on page 54.

The valid values depend on the signal type:

- Standard ISignals (i.e., ISignals with Category set to STANDARD\_LENGTH\_TYPE): For an overview of the valid values, refer to [Supported signal data types](#) on page 38.
- Array signals (i.e., ISignals with Category set to ARRAY\_TYPE): The coded base data type must be UInt8.

The specified coded base data type determines the valid initial value and length range of the ISignal:

- If required, the specified initial value is automatically saturated to the next valid value when you change the data type.
- If required, you must adapt the signal length manually when you change the data type. Refer to [Length](#) on page 286.

Additionally, the coded base data type affects settings of the Counter Signal feature. If you change the coded base data type while the Counter Signal feature is already added to instances of the ISignal, you might have to adapt counter settings. Refer to [Working with Counter Signals](#) on page 141.

**Category (coded type)** Lets you specify the ISignal category, i.e., the signal type:

Value	Description
STANDARD_LENGTH_TYPE	The ISignal is a standard ISignal, i.e., a scalar.
ARRAY_TYPE	The ISignal is an array signal, i.e., a vector. The number of vector elements is derived from the specified signal length. However, this type is available only if the coded base data type is UInt8.

The specified category determines the valid length range of the ISignal. If required, you must adapt the signal length manually when you change the category. Refer to [Length](#) on page 286.

**Base data type (physical type)** Lets you specify the base data type for the physical signal value. For an overview of the valid values, refer to [Supported signal data types](#) on page 38. For an introduction to coded and physical signal values, refer to [Signal Conversion by the Bus Manager](#) on page 54.

The physical base data type influences the ISignal as follows:

- It determines if you can add the Counter Signal and ISignal Offset Value features to instances of the ISignal. For the Counter Signal feature, the physical base data type additionally influences some counter settings.
- For the function ports of the ISignal Value, ISignal Overwrite Value, and ISignal Offset Value features, the physical base data type determines:
  - The data type and the initial value range of the function ports.
  - The saturation range (if available)

If you change the physical base data type while the bus configuration features are already added to instances of the ISignal, the relevant properties are automatically updated and saturated, if required. However, conflicts might occur.

For more information on the bus configuration features, refer to:

- [Working with ISignal Values](#) on page 139
- [Working with Counter Signals](#) on page 141
- [Overwriting ISignal Values](#) on page 146
- [Adding Offset Values to ISignal Values](#) on page 149

## Specifying user-defined settings for computation scales

Depending on the specifications in the communication matrix, the Properties Browser provides access to the computation method and the computation scale of ISignals. For ISignals with a computation method whose **Interpreted** category is set to **LINEAR**, the Properties Browser provides the following properties for specifying user-defined settings for the computation scale.

**Numerators** Lets you specify the numerator array of a linear computation scale. The specified array must comply with the following rules:

- The array must contain at least two elements and the second element must not be 0.
- If the array contains a third element, it and all following elements must be 0.

**Denominators** Lets you specify the denominator array of a linear computation scale. The specified array must comply with the following rules:

- The first element of the array must not be 0.
- If the array contains a second element, it and all following elements must be 0.

For more information on computation methods and computation scales, refer to [Signal Conversion by the Bus Manager](#) on page 54.

## Specifying user-defined settings for ISignal-to-IPDU mappings

The Properties Browser provides the following properties for specifying user-defined settings for the ISignal-to-IPDU mapping of ISignals.

**Start bit position** Lets you specify the position of the ISignal's start bit within the related ISignal IPDU. You must specify a start bit position  $\geq 0$ . The valid maximum start bit position depends on various factors, for example:

- The usable payload length of the related ISignal IPDU.
- The ISignal length.
- The endianness of the ISignal.

### Tip

If you specify a start bit position that is invalid in the context in which the ISignal is used, conflicts occur. The conflicts can help you specify a valid start bit position for the specific context. Refer to [Basics on Bus Manager-Related Conflicts](#) on page 99.

**Endianness** Lets you specify the endianness, i.e., the byte order of the ISignal within the related ISignal IPDU and the ISignal's start bit:

Value	Description
Little endian	The least significant byte of the ISignal is stored first and the least significant bit is the start bit.
Big endian	The most significant byte of the ISignal is stored first and the most significant bit is the start bit.
Opaque	The byte order is similar to the Little endian format.

The following illustration shows the different layouts resulting from a Little endian and Big endian endianness for an ISignal with 10 bits and a start bit position of 3.

ISignal IPDU: Byte	0	1	
ISignal IPDU: Bit	7   6   5   4   3   2   1   0   15   14   13   12   11   10   9   8		
Start bit position			
ISignal with little endian	[2 <sup>4</sup>   2 <sup>3</sup>   2 <sup>2</sup>   2 <sup>1</sup>   2 <sup>0</sup>   -   -   -   -   -   -   2 <sup>9</sup>   2 <sup>8</sup>   2 <sup>7</sup>   2 <sup>6</sup>   2 <sup>5</sup> ]		
ISignal with big endian	[-   -   -   -   2 <sup>9</sup>   2 <sup>8</sup>   2 <sup>7</sup>   2 <sup>6</sup>   2 <sup>5</sup>   2 <sup>4</sup>   2 <sup>3</sup>   2 <sup>2</sup>   2 <sup>1</sup>   2 <sup>0</sup>   -   -]		

For array signals (i.e., ISignals with Category set to ARRAY\_TYPE), only Little endian and Opaque are supported as the endianness.

# Replacing Assigned Communication Matrices

---

## Where to go from here

## Information in this section

Basics on Replacing Assigned Communication Matrices.....	291
How to Replace Assigned Communication Matrices.....	294

## Basics on Replacing Assigned Communication Matrices

---

### Use scenario

Communication matrices are often subject to change. To adjust a bus configuration to a communication matrix that was modified externally from ConfigurationDesk, you can replace an assigned communication matrix with the modified communication matrix.

---

### General replacing rules

Replacing a communication matrix affects:

- Only the communication matrix elements that are assigned to the selected bus configuration.  
Elements of other communication matrices that are assigned to the bus configuration and other bus configurations remain unchanged.
- All bus configuration parts (Bus Access Requests, Simulated ECUs, Inspection, Manipulation) of the selected bus configuration that contain elements of the communication matrix, regardless for which part you replace the matrix.

When you replace a currently assigned communication matrix, the Bus Manager compares the assigned elements of this communication matrix with the elements specified in the newly assigned communication matrix. Elements that can unambiguously be identified are replaced, all the other elements are either left unchanged or ignored.

Due to the changes between the replaced and newly assigned communication matrix, the replacement might result in conflicting bus configurations. The conflicts are displayed in the Conflicts Viewer. For more information, refer to [Basics on Bus Manager-Related Conflicts](#) on page 99.

#### Replacing elements in a bus configuration

**Conditions for replacing elements** The Bus Manager compares the names and the position in the bus hierarchy of the currently assigned elements with the newly assigned communication matrix. The currently assigned elements are replaced if:

- The elements are found in both the bus configuration and the newly assigned communication matrix.
- The following definitions match:
  - Communication cluster (not compared for elements of DBC and LDF communication matrices)
  - ECU (only compared for communication matrix elements that are assigned to the Simulated ECUs part)
  - Frame
  - Channel
  - Direction (only compared for TX and RX elements, such as PDUs or ISignals)
  - Element name (e.g., PDU name, ISignal name)

**Effects on element settings** In general, settings of elements that are replaced by the newly assigned communication matrix are derived from the new communication matrix (e.g., communication cluster baud rates, PDU lengths, or initial ISignal values). If you specified user-defined settings, the effects depend on the setting and/or the element you specified it for. The following table provides an overview of the effects.

User-Defined Settings	Effects	Further Information
User-defined initial values of function ports	The user-defined initial values remain unchanged or are automatically saturated if necessary (e.g., due to a changed function port data type).	<a href="#">Specifying initial values</a> on page 120
User-defined saturation values of function ports	The user-defined saturation values remain unchanged or are automatically saturated if necessary (e.g., due to a changed function port data type).	<a href="#">Specifying saturation values for function imports</a> on page 123
User-defined settings of communication matrix elements	The user-defined settings are not transferred to the newly assigned communication matrix. Thus, the user-defined settings are lost for the replaced elements.	<a href="#">Basics on Modifying Communication Matrices</a> on page 267

#### Leaving elements in a bus configuration unchanged

Elements of the replaced communication matrix that are assigned to the bus configuration but cannot unambiguously be identified in the newly assigned communication matrix remain unchanged. They are moved to a new communication matrix node named <communication matrix node name> (deprecated <number>).

The elements of the deprecated node are still part of the bus configuration and are used in the signal chain with their previously configured settings (e.g., user-

defined settings and mappings to model port blocks remain unchanged). You can, for example:

- Explore the deprecated node to check which elements can no longer be used with the newly assigned communication matrix.
- Delete the deprecated node from the bus configuration if you do not need its elements anymore.
- Configure the deprecated node if you still need some or all of its elements in the bus configuration (e.g., rename the node, configure its elements etc.).

---

#### **Ignoring elements of the newly assigned communication matrix**

Elements of the newly assigned communication matrix that are not found in the bus configuration are ignored and not assigned to the bus configuration. For example, if an ISignal IPDU of the replaced communication matrix is assigned to the bus configuration and the newly assigned communication matrix defines new ISignals or frame triggerings for this IPDU, the new elements are not assigned to the bus configuration. If you want to use the new elements, you must assign them manually.

##### **Note**

You can access some communication matrix elements, such as frame triggerings, only via the Properties Browser when you select the related higher-level element (e.g., in the Buses Browser). To assign such elements to a bus configuration, you must drag the related higher-level element to the bus configuration.

---

#### **Effects on the name of communication matrix nodes**

When you replace an assigned communication matrix in a bus configuration, the name of the communication matrix node remains unchanged. Therefore, the bus configuration structure remains unchanged regarding the communication matrix nodes. If you work with test automation scripts later on, for example, you do not have to adjust the scripts.

##### **Tip**

If you work with DBC or LDF communication matrices, the same applies to the name of the communication cluster node, which is available for elements that are assigned to the Inspection or Manipulation part of the bus configuration.

For details on node names, refer to [Element identification via node names](#) on page 65.

---

#### **Effects on the model interface**

Model port blocks that are mapped to the bus configuration are not updated automatically. You must update the model interface (e.g., delete obsolete model port blocks or add new model port blocks).

**Tip**

If you work with a MATLAB/Simulink behavior model and MATLAB/Simulink and [Model Interface Package for Simulink](#) are installed, ConfigurationDesk can update the model port blocks in ConfigurationDesk and the Simulink model. However, when you do this, it is recommended that you protect parts of your Simulink model from being accidentally changed by setting subsystems to read-only in Simulink. For more information, refer to [Handling the Model Interfaces of ConfigurationDesk and Simulink Behavior Models \(ConfigurationDesk Real-Time Implementation Guide\)](#).

---

**Limitations**

When you replace an assigned communication matrix, some limitations apply. Refer to [Limitations for Bus Configuration Handling](#) on page 328.

## How to Replace Assigned Communication Matrices

---

**Objective**

To update a bus configuration to a new version of an assigned communication matrix, you can replace the assigned communication matrix.

---

**Preconditions**

- Two or more communication matrices must be available in the ConfigurationDesk application.
- Communication matrix elements must be assigned to a bus configuration.

---

**Method**

**To replace an assigned communication matrix**

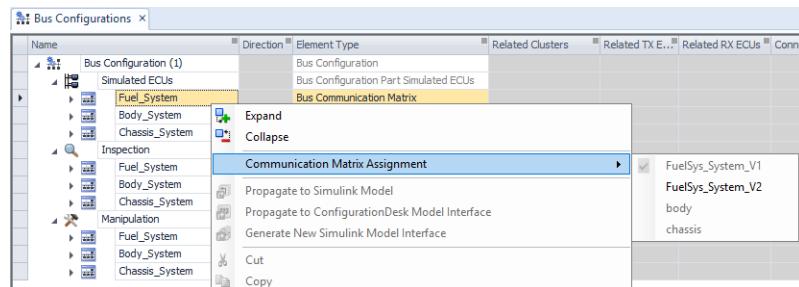
- 1 Switch to the Buses view set if necessary.
- 2 Select the Bus Configurations table.

**Tip**

You can also select the Bus Access Requests table.

- 3 Expand the Simulated ECUs, Inspection, Manipulation, or Bus Access Requests node down to the level of the communication matrix nodes.
- 4 Select the communication matrix node for which you want to replace the assigned communication matrix.

- 5 Right-click the selected node and select **Communication Matrix Assignment - <communication matrix name>** from the context menu.



### Tip

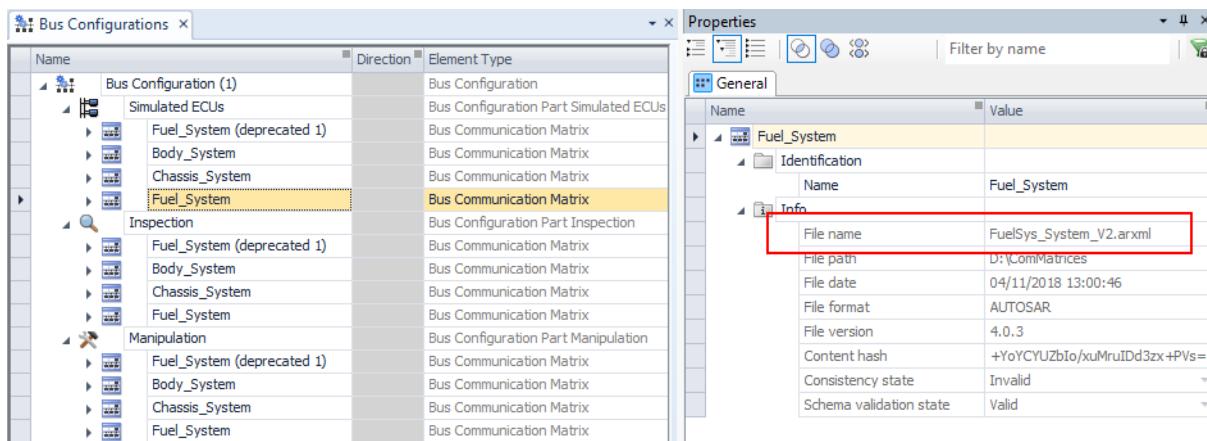
In the context menu, the currently assigned communication matrix is unavailable and marked by a  symbol.

### Result

You replaced the previously assigned communication matrix with the selected communication matrix. The replacement affects all bus configuration parts of the selected bus configuration that have at least one element of the communication matrix assigned.

Elements that are assigned to the bus configuration and that can be unambiguously identified in the newly assigned communication matrix are replaced. Elements that cannot be unambiguously identified in the newly assigned communication matrix are moved to a new communication matrix node named <communication matrix node name> (deprecated <number>). Elements that are part of only the newly assigned communication matrix but not of the bus configuration are not assigned.

For the communication matrix node, the Properties Browser displays the name of the newly assigned communication matrix.



**Tip**

If more than one communication matrix is assigned to the bus configuration, the assigned elements of the other communication matrices are not affected by the replacement.

# Working with the Bus Manager Demo

## Where to go from here

## Information in this section

<a href="#">Introduction to the Bus Manager Demo</a> .....	297
The Bus Manager demo illustrates the basic steps of using a Python automation script to configure bus communication with the Bus Manager and build a real-time application.	
<a href="#">Overview of the Example Used in the Bus Manager Demo</a> .....	298
The Bus Manager demo lets you build an application to simulate locking and unlocking the front doors of a car as well as opening and closing the side windows.	
<a href="#">How to Run the Bus Manager Demo Script via the Bus Manager</a> .....	301
To run the Bus Manager Demo script via the Bus Manager.	
<a href="#">Running the Bus Manager Demo Script via an External Interpreter</a> .....	304
You can run the Bus Manager demo script from outside the Bus Manager via an external interpreter.	
<a href="#">Steps of the Bus Manager Demo Script</a> .....	306
<a href="#">Example of Experimenting in ControlDesk</a> .....	312
This example illustrates how you can experiment with the results of the Bus Manager demo script execution in ControlDesk. It is based on the automatic central locking scenario.	

## Introduction to the Bus Manager Demo

### Overview

The Bus Manager demo illustrates the basic steps of using a Python automation script to configure bus communication with the Bus Manager and build a real-time application.

The demo is based on a simple example of door and window mechanisms of a car.

---

#### Demonstrated features

The demo covers the following features of the Bus Manager:

- Adding bus configurations and communication matrices to a ConfigurationDesk application
- Assigning communication matrix elements to the bus configurations
- Configuring the bus configurations

The demo also covers several ConfigurationDesk features not specific to the Bus Manager, such as creating projects and ConfigurationDesk applications, adding Simulink implementation containers, and building a real-time application.

---

#### Demo files

The Bus Manager demo files are located in the `BusManagerDemo` subfolder of the Bus Manager [Documents folder](#). Specifically, these are:

File	Description
<code>BusManagerDemo.py</code>	The Python script that sets up the Bus Manager demo in the Bus Manager.
<code>BusManagerDemo.arxml</code>	The <a href="#">communication matrix</a> of the simulated bus network.
<code>CentralGatewayECU.sic</code> <code>EngineAndBodyECUs.sic</code>	The <a href="#">Simulink implementation containers</a> , which are added to the ConfigurationDesk application by the demo script.
<code>CentralGatewayECU.slx</code> <code>EngineAndBodyECUs.slx</code>	The Simulink models on which the SIC files are based.

---

#### Running the Bus Manager demo script

There are two methods for running the Bus Manager demo script:

- Running the Bus Manager demo script via the Bus Manager. Refer to [How to Run the Bus Manager Demo Script via the Bus Manager](#) on page 301.
- Running the Bus Manager demo script from outside the Bus Manager via an external interpreter, such as Python.exe. Refer to [Running the Bus Manager Demo Script via an External Interpreter](#) on page 304.

## Overview of the Example Used in the Bus Manager Demo

---

#### Use cases

The Bus Manager demo lets you build an application to simulate locking and unlocking the front doors of a car as well as opening and closing the side windows.

Specifically, this includes the following use cases:

- Centrally locking/unlocking the doors
- Automatic central locking

- Locking/unlocking the doors individually
- Opening/closing the side windows

To illustrate the functionality of the demo, the automatic central locking use case is explained in more detail below.

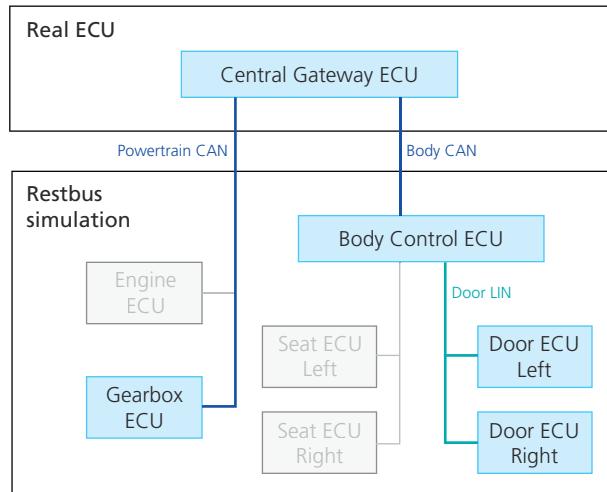
#### ECUs and communication clusters

The example used in the demo consists of five [ECUs](#) that are members of three [communication clusters](#):

Communication Cluster	ECUs
Powertrain	Central Gateway Gearbox
Body CAN	Central Gateway Body Control
Door LIN	Body Control Door Left Door Right

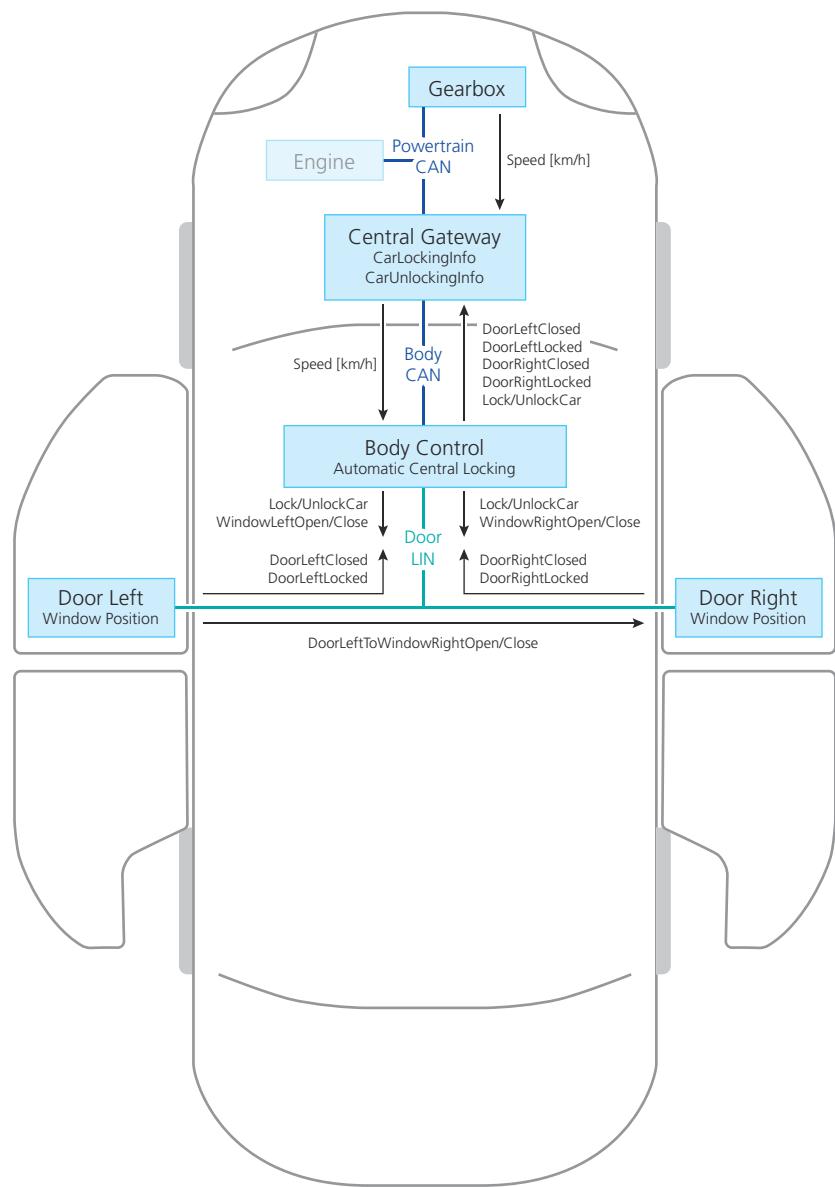
The Gearbox ECU, Body Control ECU, and the Door ECUs are part of a restbus simulation, while the Central Gateway ECU simulates a real ECU.

The following illustration shows the ECUs and how they are organized in communication clusters. The grayed-out elements are contained in the [communication matrix](#) of the Bus Manager demo, but are not used in the demo.

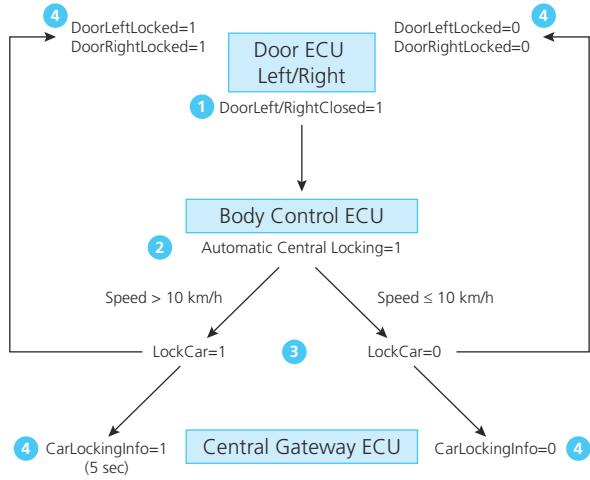


#### Functionality

The following illustration summarizes the functionality of the demo. It shows all relevant signals together with their directions and associated ECUs. The labels inside the Body Control, Central Gateway, and Door ECUs represent internal ports of the corresponding Simulink models, which are not accessible in the Bus Manager. Their values can be displayed as part of an experiment in ControlDesk, for example.



**Automatic central locking** The following schematic illustrates the signal sequencing of the automatic central locking use case.



1. When one of the doors is closed, the Body Control ECU receives the signal **DoorLeft/RightClosed** with a value of 1 from the Door ECU Left/Right.
2. The Body Control ECU sets the internal **Automatic Central Locking** flag to 1.
3. If the **Speed** signal exceeds 10 km/h, the Body Control ECU sends the signal **LockCar** with a value of 1 to the Door ECUs and to the Central Gateway ECU.
4. The Central Gateway ECU sets the internal port **CarLockingInfo** to 1 for five seconds to indicate the locking procedure in the cockpit. As soon as the doors are locked, the Door ECUs send the signal **DoorLocked** with a value of 1.

## How to Run the Bus Manager Demo Script via the Bus Manager

### Objective

To run the Bus Manager demo script via the Bus Manager.

### Preconditions

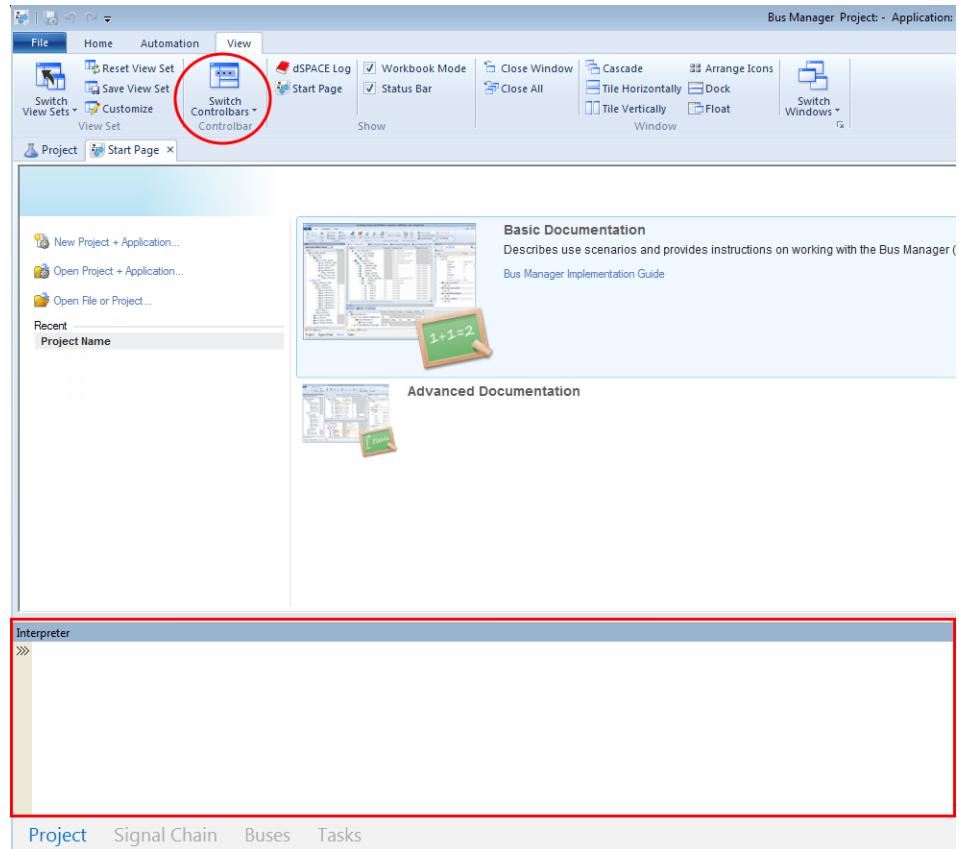
- The Bus Manager license, CAN module license, and LIN module license are available and accessible.  
Refer to [Basics on Licenses for Working with the Bus Manager \(Stand-Alone\)](#) on page 24.
- You started the Bus Manager and no unsaved project is open.

**Method**

**To run the Bus Manager demo script**

- 1 On the View ribbon, click Switch Controlbars - Interpreter.

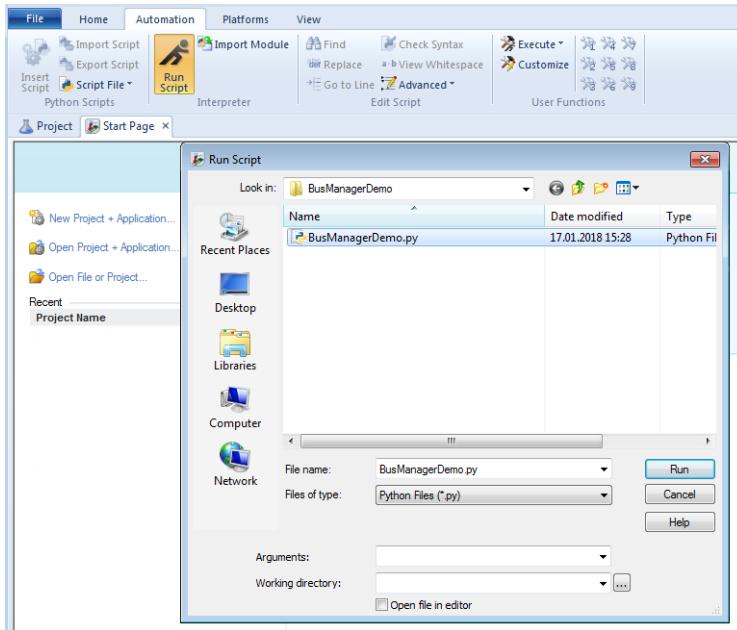
The Interpreter opens. It displays the script progress and potential error messages.



- 2 From the Automation ribbon, choose Interpreter - Run Script.

The Run Script dialog opens.

- 3 In the Run Script dialog, navigate to the **BusManagerDemo** folder in the Bus Manager **Documents folder** [?.](#)

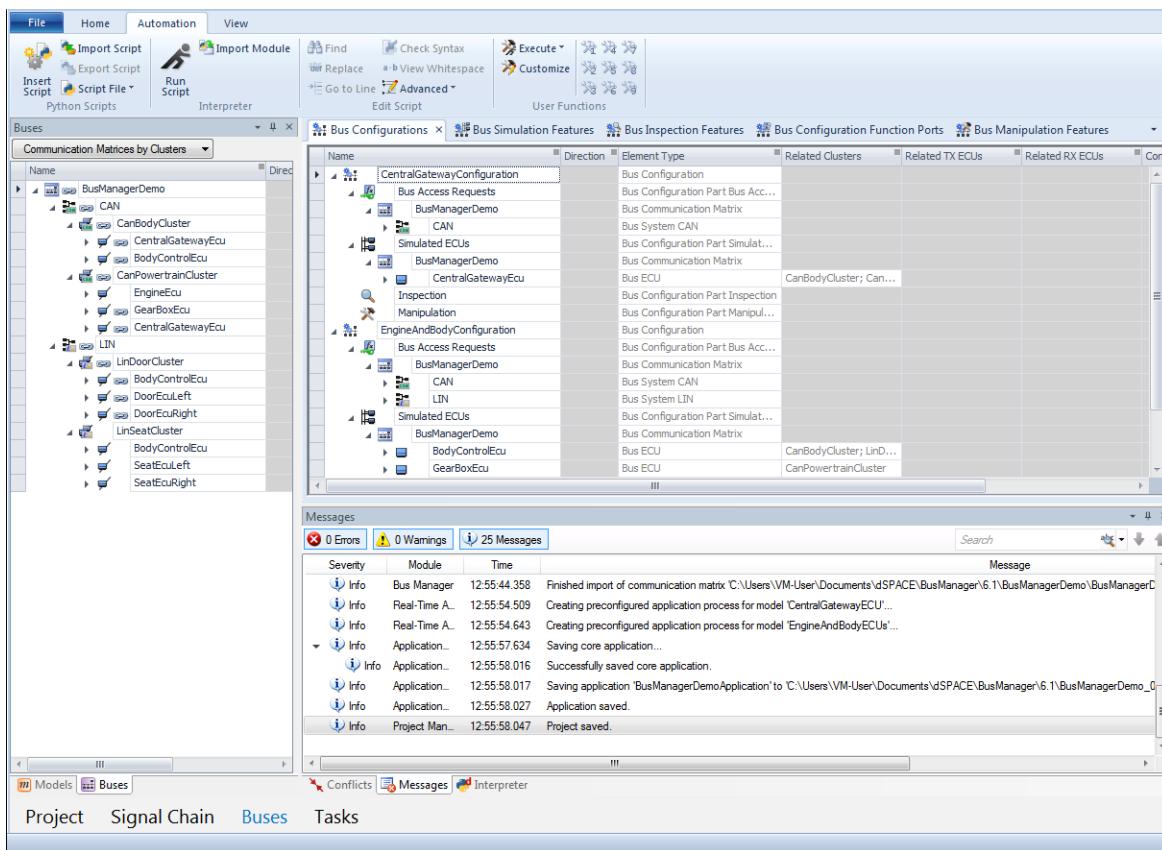


- 4 Select the **BusManagerDemo.py** file and click **Run**.

## Result

You run the Bus Manager demo script via the Bus Manager.

The Bus Manager displays the Buses view set, as shown in the following illustration. The Message Viewer displays all the steps initiated by the demo script.



## Running the Bus Manager Demo Script via an External Interpreter

### Introduction

You can run the Bus Manager demo script from outside the Bus Manager via an external interpreter, such as Python.exe, which is included in the dSPACE Python distribution. If you want to do this, you must adapt the demo script.

#### Note

dSPACE is not responsible for the correct functioning of the Bus Manager demo once you make changes to the script.

For an overview of the external interpreters that are included in the dSPACE Python distribution, refer to [Basics on External Interpreters \(ConfigurationDesk Automating Tool Handling\)](#).

### Preconditions

- To access the demo script, you must have started the Bus Manager at least once. By this, the [Documents folder](#) is set up. The script is located in the BusManagerDemo subfolder of the Documents folder.

- To run the demo script, the Bus Manager license, CAN module license, and LIN module license must be available and accessible. Refer to [Basics on Licenses for Working with the Bus Manager \(Stand-Alone\)](#) on page 24.

### Adapting the demo script

To run the demo script via an external interpreter, you must adapt the script as follows:

1. Open the `BusManagerDemo.py` file in an editor.
2. Go to the `def CreateApplication(self)` code line.
3. In the definition of the `CreateApplication` method, do the following:
  - Remove the `#`, including the whitespace before `globals()['Application'] = Dispatch('BusManager.Application')`.
  - If there is no `#` before `globals()['Application'] = Dispatch('ConfigurationDesk.Application')`, add it.

The resulting code looks as follows:

```
def CreateApplication(self):  
    ...  
    # globals()['Application'] = Dispatch('ConfigurationDesk.Application')  
    globals()['Application'] = Dispatch('BusManager.Application')
```

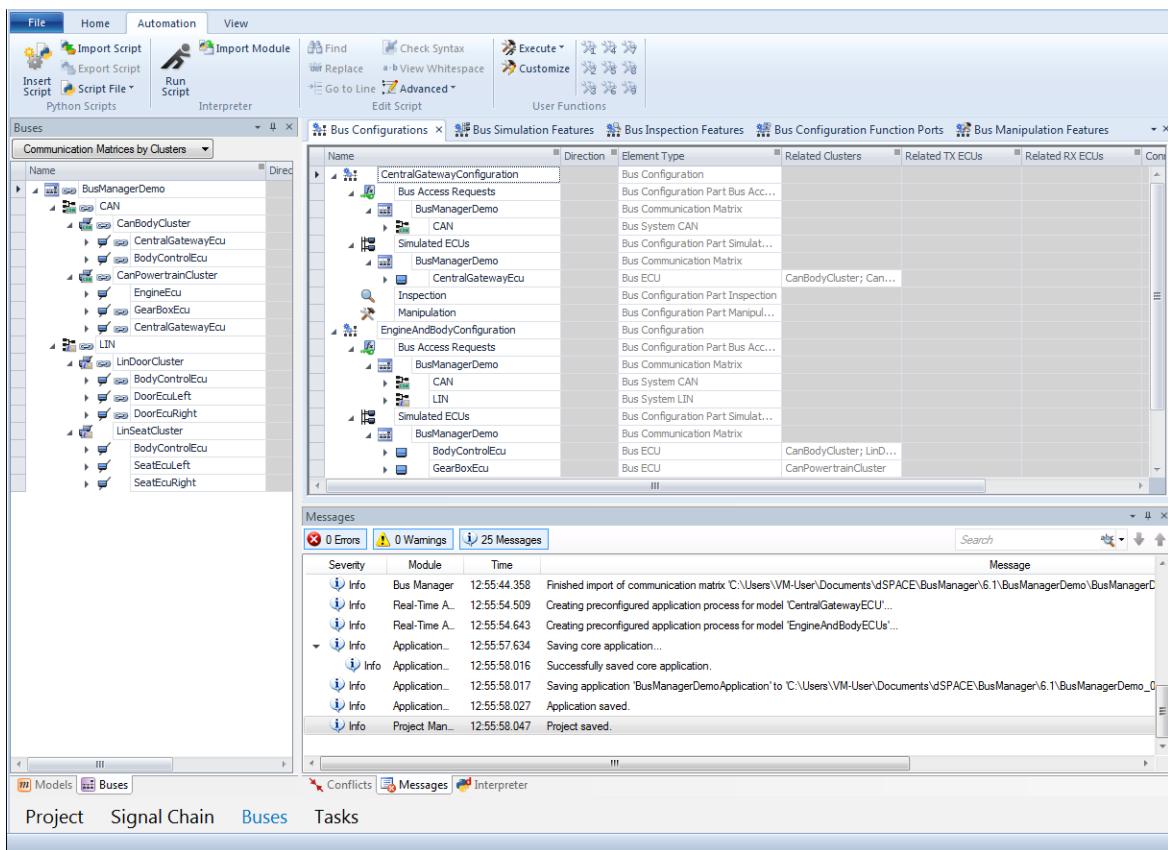
4. Save the `BusManagerDemo.py` file.

### Running the demo script

After you adapted the `BusManagerDemo.py` file, you can run the script via an external interpreter.

When you run the script, it starts the Bus Manager and sets up the Bus Manager demo.

After the script has finished, the Bus Manager displays the Buses view set, as shown in the following illustration. The Message Viewer displays all the steps initiated by the demo script.



## Steps of the Bus Manager Demo Script

### Overview

### Tip

The Bus Manager demo script can be used with both variants of the Bus Manager, the Bus Manager in ConfigurationDesk and the Bus Manager (stand-alone). However, with the Bus Manager (stand-alone), some steps of the Bus Manager demo script are skipped.

The Bus Manager demo script executes the following steps:

- A new project is created along with a ConfigurationDesk application. Refer to [Creating a new ConfigurationDesk project and application](#) on page 307.
- Two Simulink implementation containers are added to the ConfigurationDesk application. Refer to [Adding Simulink implementation containers](#) on page 308.
- A hardware topology is created. With the Bus Manager (stand-alone), this step is skipped.
- A communication matrix is added to the ConfigurationDesk application. Refer to [Adding a communication matrix](#) on page 308.

- Two bus configurations are added to the ConfigurationDesk application. Refer to [Adding bus configurations](#) on page 309.
- Communication matrix elements are assigned to the bus configurations. Refer to [Assigning communication matrix elements to bus configurations](#) on page 309.
- The bus configuration function ports are configured. Refer to [Configuring bus configuration function ports](#) on page 310.
- Function ports are mapped to model ports. Refer to [Mapping function ports to model ports](#) on page 311.
- Bus function blocks are assigned to bus access requests and configured. Refer to [Adding and configuring bus function blocks](#) on page 311.
- A 12 V power supply for the LIN controller is added as an external device. Refer to [Adding an external device](#) on page 311.
- A preconfigured application process is created for each model. Refer to [Creating application processes](#) on page 312.
- A real-time application is built. With the Bus Manager (stand-alone), this step is skipped.

After the execution of the script has finished, you can generate bus simulation containers, for example. Refer to [Next steps](#) on page 312.

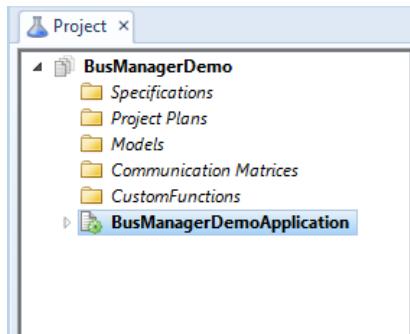
#### Note

dSPACE is not responsible for the correct functioning of the Bus Manager demo once you make changes to the script.

---

#### Creating a new ConfigurationDesk project and application

In the first step, the Bus Manager demo script creates a project named **BusManagerDemo** which contains the **BusManagerDemoApplication** application as shown in the following illustration.



If a project with a name containing the **BusManagerDemo** string already exists, the demo script names the new project **BusManagerDemo** and extends it by an increasing number, i.e., **BusManagerDemo\_001** and so on.

The corresponding function is named **CreateApplication**. It also defines variables in which the required API components and relations are stored, for example:

```

1 self.BusManager = ActiveApplication.Components.Item('BusManager')
2 self.BusConfigurationsRelation = ActiveApplication.Relations.Item('BusConfigurations')

```

For more information on the required API components and relations, refer to [ICaComponent <>Collection>> \(ConfigurationDesk Automating Tool Handling\)](#) and [ICaRelation <>Interface>> \(ConfigurationDesk Automating Tool Handling\)](#), respectively.

## Adding Simulink implementation containers

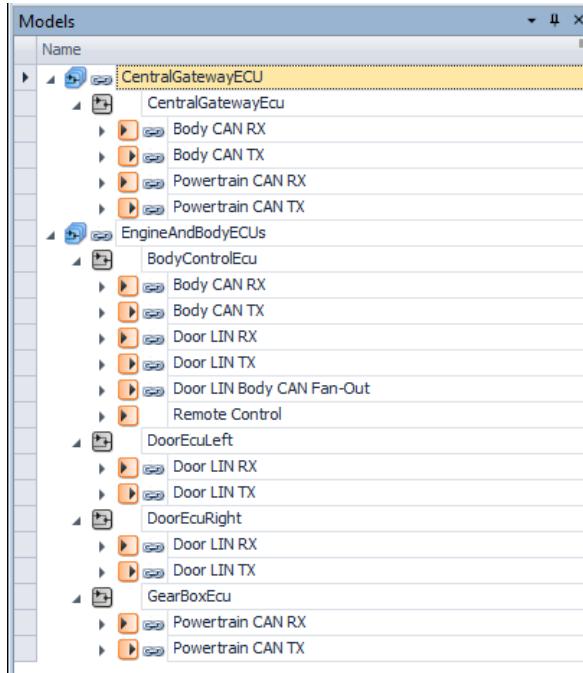
The `LoadSimulinkModel` function adds the `CentralGatewayECU.sic` and `EngineAndBodyECUs.sic` files to the application. It takes only the file name part as its argument, prefixes the current path, and extends the file name part with the extension `.sic`, as shown in the following excerpt from the function definition.

```

1 def LoadSimulinkModel(self, SimulinkModelFile):
2     ...
3     self.ModelTopology.Configure('AddModel', ['%s\\%s.sic' % (self.CurrentPath, SimulinkModelFile)])
4     ...

```

The following illustration shows the Model Browser displaying the model topologies.



## Adding a communication matrix

The `LoadCommunicationMatrix` function checks if a communication matrix has already been added to the ConfigurationDesk application. If this is not the case, it adds the communication matrix named after the script, i.e., `BusManagerDemo.arxml`, from the current directory, as shown in the following code listing.

```

1 def LoadCommunicationMatrix(self):
2     ComMatrixFile = '%s\\%s.arxml' % (self.CurrentPath, self.ScriptName)

```

```

3     try:
4         # Check whether a communication matrix has already been loaded.
5         if (self.CommunicationMatricesByEcusRelation.GetTopNodes().Count == 0):
6             self.BusManager.Configure('AddCommunicationMatrix', [ComMatrixFile])
7         except com_error as ComError:
8             self.SubmitErrorAndExit('Failed to load communication matrix file "%s' % (ComMatrixFile))

```

## Adding bus configurations

The bus configuration names are stored in the `BusConfigurationName` list. The list is passed to the `CreateBusConfiguration` function of the Bus Manager demo script which adds the bus configurations to the ConfigurationDesk application. The following code listing shows the relevant lines of the function definition.

```

1 def CreateBusConfiguration(self, BusCfgName):
2     ...
3     BusConfiguration =
4     self.BusConfigurationsRelation.CreateDataObject(self.BusConfigurationsRelation.GetCreatableTypes().Item(0))
5     BusConfiguration.Name = BusCfgName
6     ...
7     return BusConfiguration

```

## Assigning communication matrix elements to bus configurations

The `AssignEcuToBusCfg` function is called with two arguments, `BusConfiguration` and ECUs. It assigns all communication matrix elements that belong to the ECUs in the `ECUs` list to the bus configuration specified in `BusConfiguration`. The communication matrix elements contained in the `ExcludeListOfComMatrixElementNames` list are excluded. This list is defined at the start of the script.

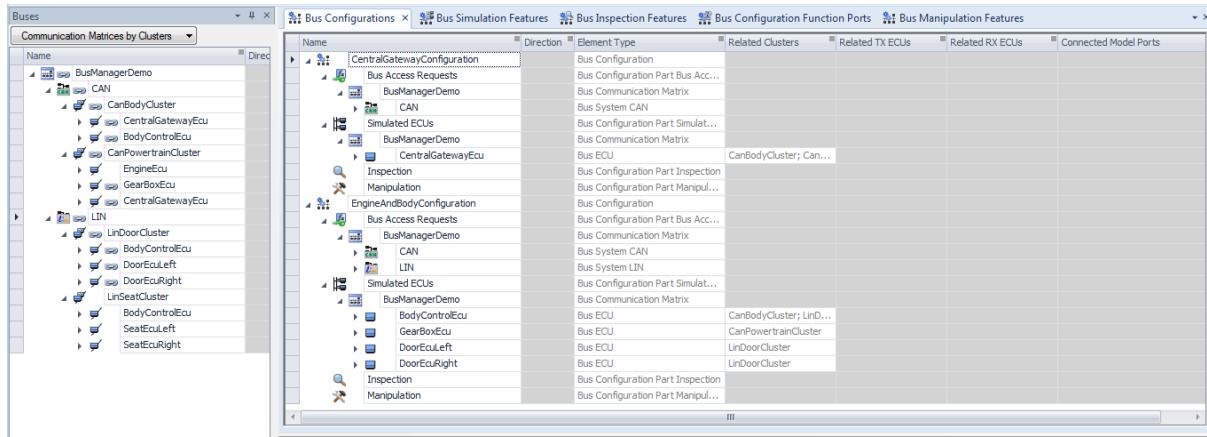
The following code listing shows the relevant part of the function definition.

```

1 def AssignEcuToBusCfg(self, BusConfiguration, ECUs):
2
3     CommunicationMatrix = self.CommunicationMatricesByEcusRelation.GetTopNodes().Item(0)
4
5     WriteTransaction = self.OpenTransaction('AssignElements')
6     try:
7         for Ecu in self.CommunicationMatricesByEcusRelation.GetElements(CommunicationMatrix):
8             if (Ecu.Name in ECUs):
9                 for Pdu in self.CommunicationMatricesByEcusRelation.GetElements(Ecu):
10                     # Check content of bus element exclude list.
11                     if (not Pdu.Name in ExcludeListOfComMatrixElementNames):
12                         for Signal in self.CommunicationMatricesByEcusRelation.GetElements(Ecu):
13                             if (not Signal.Name in ExcludeListOfComMatrixElementNames):
14                                 self.BusManager.Configure('AssignElements', [[Signal], BusConfiguration])
15 ...

```

The following illustration shows a part of the Buses view set that displays the resulting bus configurations in the Bus Configurations table and the corresponding communication matrix in the Buses Browser.



## Configuring bus configuration function ports

The `ActivateModelAccess` function performs the configuration of the bus configuration function ports. It activates model access for all bus configuration function ports of the given bus configuration except for those specified in the list `ExcludeListOfModelAccessFunctionPortNames`. It also enables test automation support for all bus configuration function ports, without exceptions.

The following code listing shows the relevant part of the function definition:

```

1 def ActivateModelAccess(self, BusConfiguration):
2
3     WriteTransaction = self.OpenTransaction('ActivateModelAccess')
4
5     ...
6     for FunctionPort in self.BusConfigurationsRelation.FindByXPath('/BusConfiguration[@Name = "' + 
7         BusConfiguration.Name + '"]/FunctionPort'):
8         if (not FunctionPort.Name in ExcludeListOfModelAccessFunctionPortNames):
9             FunctionPort.Properties['IsMappable'].TrySetValue(True)
10            else:
11                # Model access is not enabled. Set the initial switch setting to a substitute value.
12                FunctionPort.Properties['InitialSwitchSetting'].TrySetValue(1)
13                FunctionPort.Properties['IsTestAutomationSupportEnabled'].TrySetValue(True)
14
15    ...

```

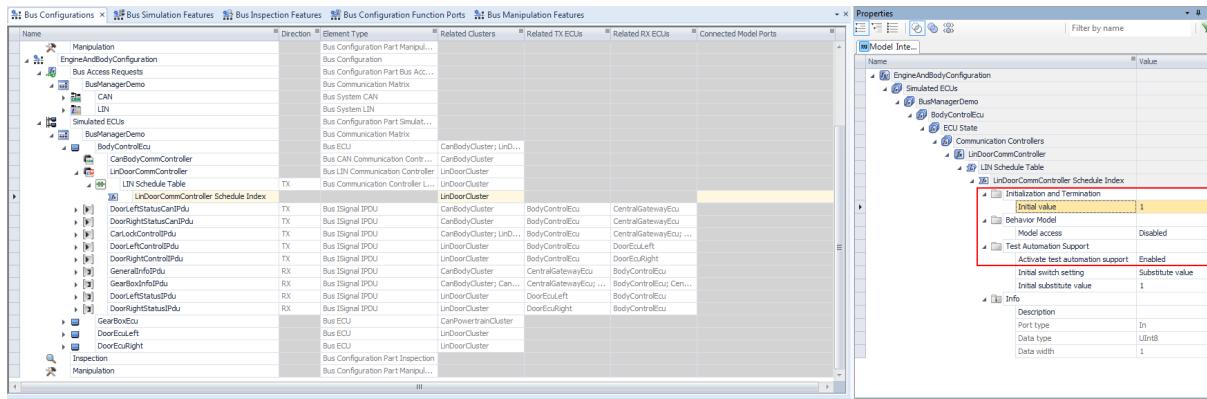
**Enabling LIN communication** The Bus Manager does not activate LIN communication by default. The Bus Manager demo script enables LIN communication by setting the `Initial` value property of the LIN Schedule Index function ports to 1:

```

1 ScheduleIndexes = DemoController.BusConfigurationsRelation.FindByXPath('/BusConfiguration//'
2     BusCommunicationControllerLinScheduleTableAccess/FunctionPort')
3 for SchedulIndex in ScheduleIndexes:
4     SchedulIndex.Properties['InitialValue'].Value = 1
5     SchedulIndex.Properties['InitialSubstituteValue'].Value = 1

```

The following illustration shows the LIN Schedule Index function port in the Bus Configurations table and the function port properties in the Properties Browser. Because the LIN Schedule Index function port is on the `ExcludeListOfModelAccessFunctionPortNames` list, model access is disabled.



## Mapping function ports to model ports

The `CreatePortLinks` function maps function ports to model ports as shown in the following excerpt of the function code.

```
1 def CreatePortLinks(self, BusConfiguration, Model):
2     ...
3     for ModelEcu in Model:
4         EcuName = ModelEcu.Name
5         BusEcus = self.BusConfigurationsRelation.FindByXPath('/BusConfiguration[@Name = "%s"]//BusEcu[@Name
6             "%s"]' % (BusConfiguration.Name, ModelEcu.Name))
7         if (BusEcus.Count > 0):
8             for ModelPortBlock in ModelEcu:
9                 for ModelFunctionPort in ModelPortBlock:
10                     # Make sure the model port is not mapped yet.
11                     if ModelFunctionPort.Links.Count == 0:
12                         xPath = '/BusConfiguration//BusEcu[@Name = "%s"]//FunctionPort[@Name = "%s"
13                             and @PortType != "%s"]' % (ModelEcu.Name, ModelFunctionPort.Name,
14                             str(ModelFunctionPort.Properties['PortType'].Value))
15                         BusFunctionPorts = self.BusConfigurationsRelation.FindByXPath(xPath)
16                         for BusFunctionPort in BusFunctionPorts:
17                             if BusFunctionPort.Links.Count == 0:
18                                 self.ActiveApplication.ConnectObjects(BusFunctionPort, ModelFunctionPort)
19
20     ...
```

## Adding and configuring bus function blocks

The `CreateBusAccess` function adds suitable CAN and LIN function blocks to the signal chain and assigns them to the [bus access requests](#). This step is required only if you build a real-time application later on. The CAN and LIN function blocks have no effect on generating bus simulation containers. With the Bus Manager (stand-alone), you can therefore ignore the CAN and LIN function blocks.

## Adding an external device

The `SetupDeviceWiring` function creates an external device representing a 12 V power supply and connects it to the LIN controller of the LIN function block. This is required only if you build a real-time application later on and download it to real-time hardware. The external device has no effect on generating bus simulation containers. With the Bus Manager (stand-alone), you can therefore ignore the external device.

## Creating application processes

The `CreateProcessAndTasks` function creates a processing unit application if none exists yet, and a corresponding [preconfigured application process](#) for each model.

The following illustration shows the result in the **Task Configuration** table. The periodic tasks are provided by the models.

Name	Source name	Priority	DAQ Raster Name	Real-Time Testing	Period	Offset	Number of Accepted Ov...	Jitter and Latency Optimiz...
CentralGatewayECU	CentralGatewayConfig...	0						
BusManagerDemo [CentralGatewayConfig...	CentralGatewayConfig...	40	Periodic Task 1	<input checked="" type="checkbox"/>	0.001	0	0	Standard (full functionality)
Periodic Task 1								
EngineAndBodyECUs	EngineAndBodyConfig...	0						
BusManagerDemo [EngineAndBodyConfig...	EngineAndBodyConfig...	40	Periodic Task 1	<input checked="" type="checkbox"/>	0.001	0	0	Standard (full functionality)
Periodic Task 1								

## Next steps

You can use the results of the Bus Manager demo script execution to simulate the functionality of the Bus Manager demo in ControlDesk. This involves the following steps:

**Generating bus simulation containers** You can generate bus simulation containers that contain the configured bus communication. To do this, click **BSC - Generate** on the Home ribbon in the Buses view set.

The Bus Manager generates one bus simulation container for each application process and saves the generated bus simulation containers under `BusManagerDemo/BusManagerDemoApplication/GeneratedContainers` in the Bus Manager [Documents folder](#).

You can then import the bus simulation containers to the [VEOS Player](#) for integration in an offline simulation. Refer to [How to Import Bus Simulation Containers \(VEOS Manual\)](#).

**Experimenting with ControlDesk** Once you have integrated a bus simulation container in an offline simulation in the VEOS Player, you can use ControlDesk's Bus Navigator to experiment with the bus communication. Refer to [Example of Experimenting in ControlDesk](#) on page 312.

## Example of Experimenting in ControlDesk

### Introduction

This example illustrates how you can experiment with the results of the Bus Manager demo script execution in ControlDesk.

The example is based on the automatic central locking scenario described in [Automatic central locking](#) on page 301.

### Preconditions

- You generated bus simulation containers and integrated them into an offline simulation application as described in [Next steps](#) on page 312.
- You have a decrypted installation of ControlDesk. For information on installing and decrypting dSPACE software, refer to [Installing dSPACE Software](#).

## Workflow

The example consists of the following steps:

- Preparing an experiment in ControlDesk. Refer to [Part 1](#) on page 313.
- Experimenting and measuring in ControlDesk. Refer to [Part 2](#) on page 316.

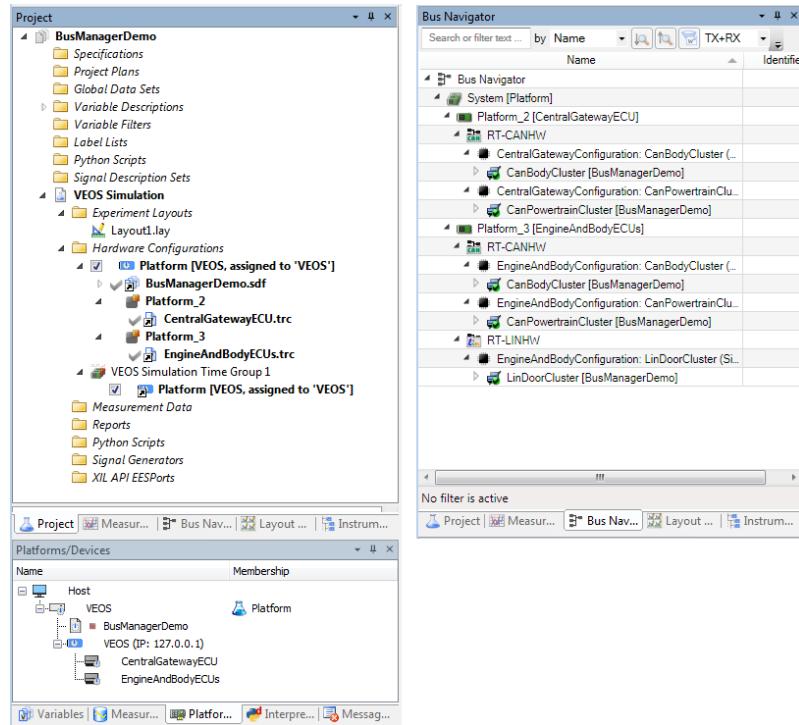
## Part 1

### To prepare the experiment in ControlDesk

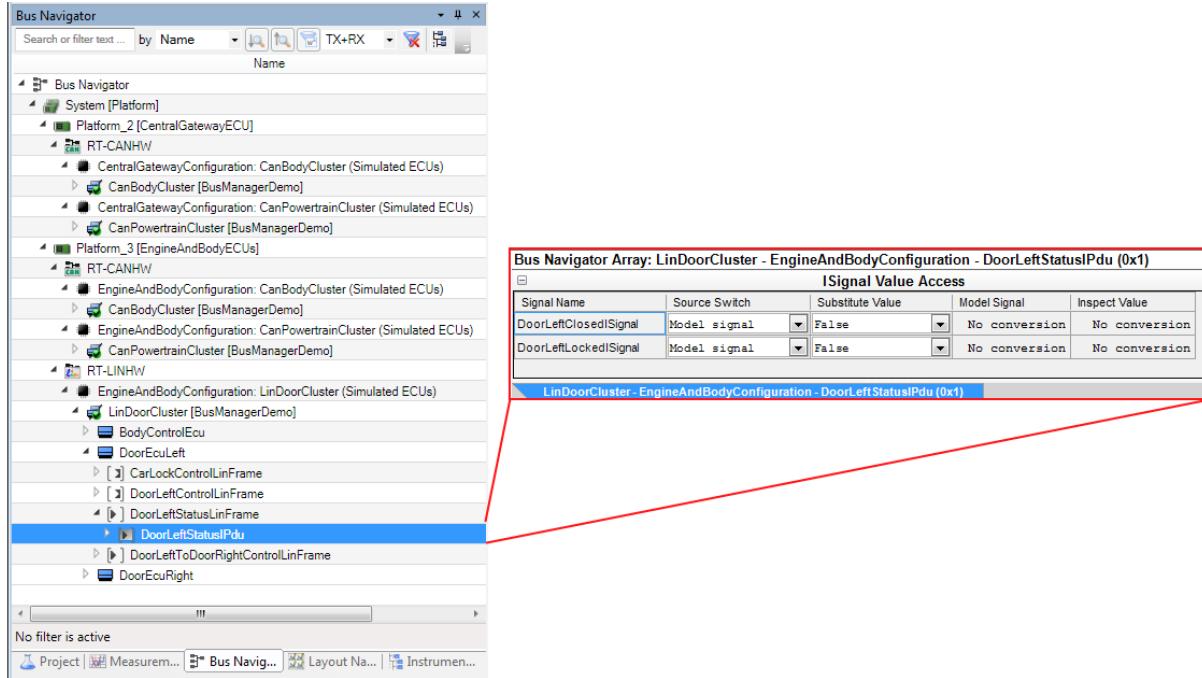
- 1 Start ControlDesk.
- 2 Register VEOS.
- 3 Create a new project and experiment with the following settings:

Project	Experiment	Platform	Variable Description
BusManagerDemo	VEOS Simulation	VEOS	<VEOS documents folder>/ BusManagerDemo.sdf

After you finish, the Project Manager and the Platforms/Devices controlbar look as shown in the following illustration. The left side shows the Project Manager and the Platforms/Devices controlbar. The Bus Navigator controlbar is shown on the right side.



- 4 In the Bus Navigator controlbar, expand the LinDoorCluster communication cluster and generate a Bus Instrument (TX type) for the DoorLeftStatusPdu in the DoorLeftStatusLinFrame of the DoorEcuLeft ECU, as shown in the following illustration.

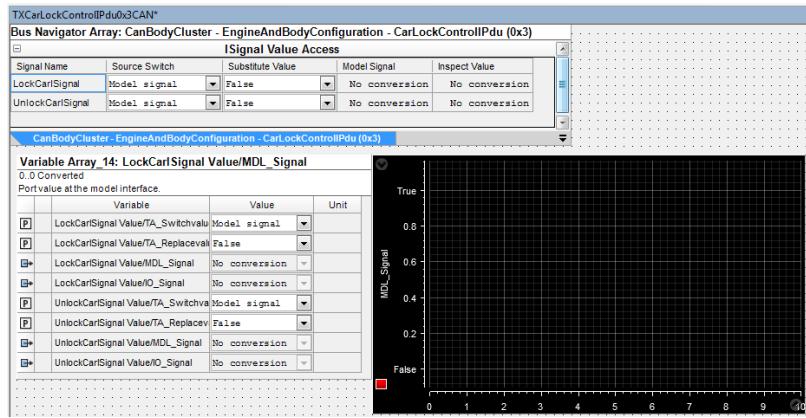


- 5 Proceed as in step 4 to generate the following TX layouts for the Platform\_3 [EngineAndBodyECUs] platform:
  - LinDoorCluster - DoorEcuRight - DoorRightStatusLinFrame - DoorRightStatusPdu
  - CanPowertrainCluster - GearBoxEcu - GearBoxInfoFrame - GearBoxInfoPdu
  - CanBodyCluster - BodyControlEcu - CarLockControlCanFrame - CarLockControlPdu
- 6 In the DoorLeftStatusPdu and DoorRightStatusPdu instruments, change Source Switch of DoorLeftClosedISignal and DoorRightClosedISignal to Substitute value. Leave Substitute Value at False.

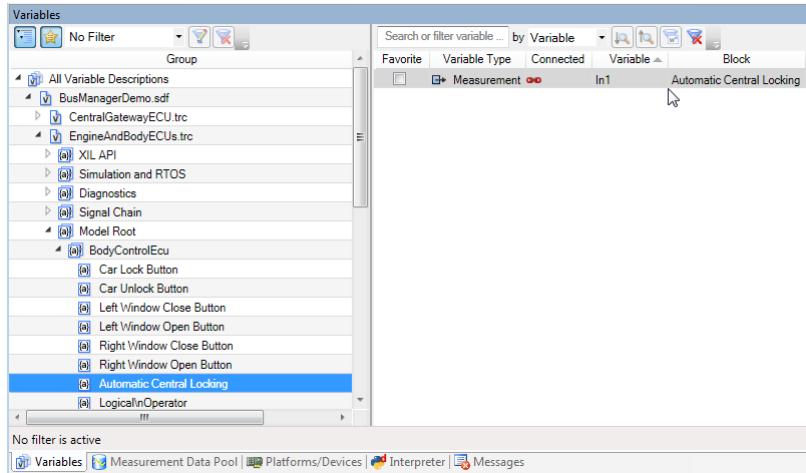
Bus Navigator Array: LinDoorCluster - EngineAndBodyConfiguration - DoorRightStatusPdu (0x2) - ISignal Value Access				
Signal Name	Source Switch	Substitute Value	Model Signal	Inspect Value
DoorRightClosedSignal	Substitute value	False	No conversion	No conversion
DoorRightLockedSignal	Model signal	False	No conversion	No conversion

- 7 In the CarLockControlPdu instrument, right-click the LockCarISignal field and select Variable(s) - Copy to Instrument - Variable Array. ControlDesk displays all the variables mapped to LockCarISignal.
- 8 In the Variable Array, open the context menu of LockCarISignal Value/MDL\_Signal and select Variable(s) - Copy to Instrument - Time Plotter. ControlDesk displays a Time Plotter for the model signal.

The following illustration shows the CarLockControlIPdu layout with the Variable Array and the Time Plotter.

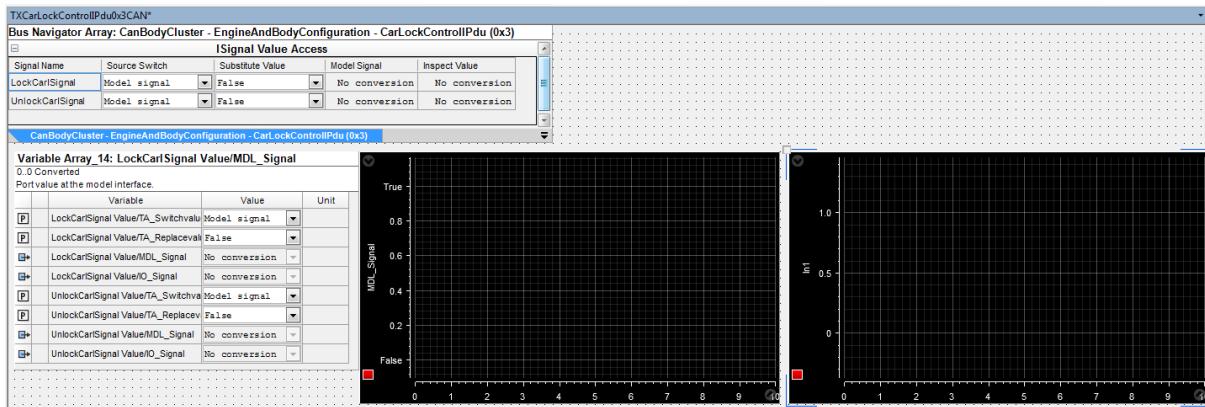


- In the Variables controlbar, expand the model tree as shown in the following illustration.



From the context menu of Automatic Central Locking in the variable list on the right, select Visualize Variables.

ControlDesk displays a Time Plotter for the Automatic Central Locking variable. In the following illustration the Time Plotter is arranged to the right of the model signal plot in the CarLockControlIPdu layout.



### Interim result

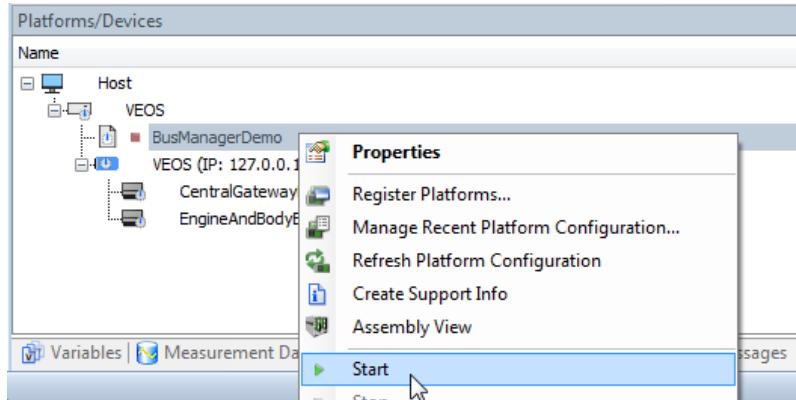
You created a project and experiment in ControlDesk and generated Bus Instruments to manipulate signals of the Bus Manager demo. You also created Time Plotters to visualize the results.

You can now start experimenting and measuring, i.e., you can access signals and parameters of the offline simulation application.

### Part 2

#### To experiment and measure in ControlDesk

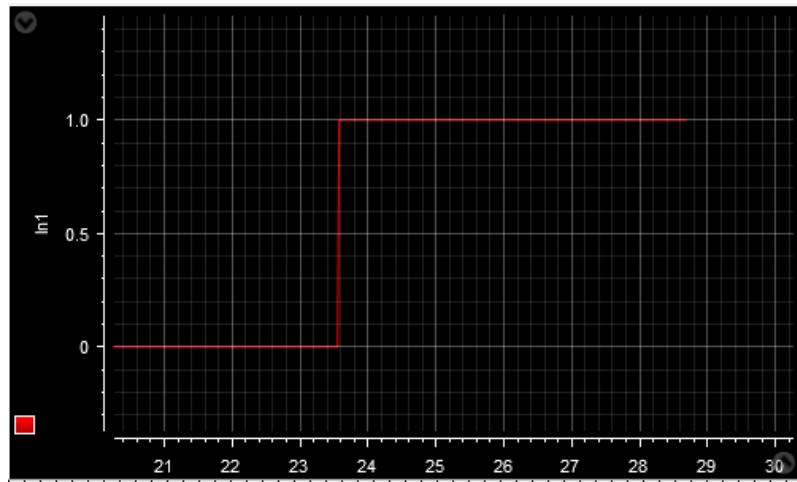
- In the Platforms/Devices controlbar, open the context menu of the BusManagerDemo simulation and click Start.



VEOS starts the simulation.

- On the Home ribbon, click Status Control - Start Measuring. ControlDesk starts the measurement. Because both door ECUs indicate that the doors are open, the Automatic Central Locking parameter is 0. The value of LockCarlSignal is False.
- In the DoorLeftStatusIPdu instrument, change the Substitute Value of DoorLeftClosedISignal to True.

The Automatic Central Locking parameter switches to 1 as shown in the following illustration.

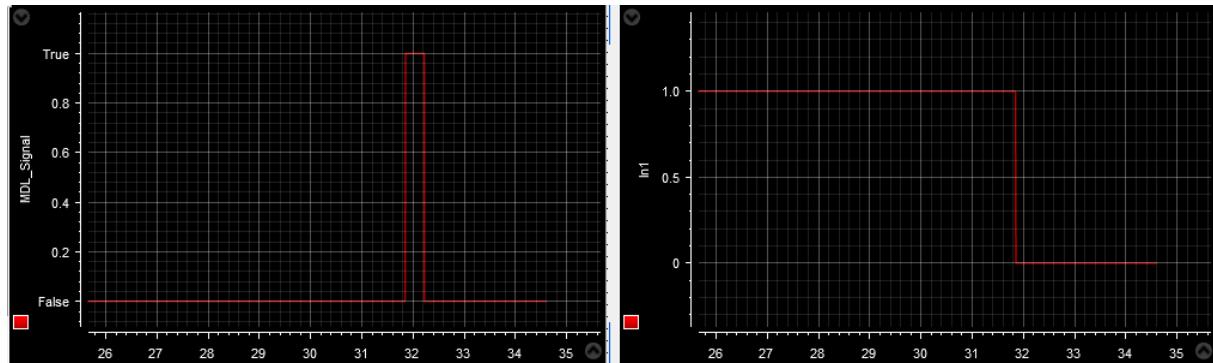


- 4 In the GearBoxInfoIPdu instrument, change the Source Switch of SpeedISignal to Substitute value, then change the Substitute Value to a value greater than 10.

The value of LockCarISignal briefly switches to 1. Consequently, the DoorLeftLockedISignal and DoorRightLockedISignal change to True.

The Automatic Central Locking parameter switches back to 0.

The following illustration shows the behavior of LockCarISignal on the left side and the Automatic Central Locking parameter on the right side.



## Result

You simulated automatic central locking in ControlDesk.



# Limitations for Using the Bus Manager

## Where to go from here

## Information in this section

Limitations for Communication Matrices and Communication Standards.....	319
Limitations for CAN Communication.....	323
Limitations for LIN Communication.....	325
Limitations for Bus Configuration Handling.....	328
Limitations for Bus Simulation Containers.....	331

## Limitations for Communication Matrices and Communication Standards

### Limitations for using transmission modes according to AUTOSAR and FIBEX

**Support of only one cyclic timing** The Bus Manager supports only one transmission mode with a cyclic timing. According to the AUTOSAR and FIBEX standards, a communication matrix can specify two transmission modes (True and False) for a PDU. In this case, the Bus Manager evaluates the transmission modes in the following order:

1. Transmission mode True
2. Transmission mode False

The Bus Manager uses the first found cyclic timing for transmitting a PDU. Other cyclic timings and transmission modes (e.g., event-controlled timings) are ignored.

PDUs for which the communication matrix does not specify a cyclic timing can be transmitted only if you specify a cyclic transmission via the Bus Manager, add the PDU Trigger feature to a PDU, or the Frame Access feature to a frame.

**No switching of transmission modes during run time** You cannot switch between the transmission modes (True and False) of a PDU during run time.

<b>Limitations for multiplexed IPDUs</b>	<p><b>No support of selector fields with more than one byte</b> The Bus Manager supports only selector fields of multiplexed IPDUs with a length of 1 byte. Longer selector fields are not supported.</p> <p><b>Only dynamicPartTrigger trigger mode supported</b> The Bus Manager supports only <code>dynamicPartTrigger</code> as the trigger mode. Other trigger modes such as <code>staticPartTrigger</code> or <code>staticOrDynamicPartTrigger</code> are not supported.</p>
<b>Limitations for container IPDUs</b>	<p><b>No support of nested container IPDUs</b> The Bus Manager does not support nested container IPDUs. This applies also to hierarchical nested container IPDUs, i.e., to container IPDUs that contain at least one secured IPDU which is a container IPDU itself.</p> <p><b>No support of Trigger Transmit mode</b> For the AUTOSAR parameter <code>IpduMContainerTxTriggerMode</code>, the Bus Manager does not support the Trigger Transmit mode, i.e., instances of a container IPDU cannot be queued (e.g., if adding a contained IPDU to the container IPDU exceeds the boundaries of the container IPDU).</p> <p><b>No interface for displaying received queued contained IPDU instances</b> When instances of a contained IPDU are queued within an RX container IPDU, the Bus Manager receives all the instances but cannot display them. Instead, only the last received contained IPDU instance is displayed.</p> <p><b>No support of IPDUs with dynamic length for container IPDUs with static container layout</b> For container IPDUs with a static container layout, the Bus Manager does not support contained IPDUs with dynamic length.</p> <p><b>Limitation for transmitting container IPDUs with static container layout</b> For container IPDUs with a static container layout, the Bus Manager ignores the <code>THRESHOLD-SIZE</code> attribute, i.e., the transmission of the container IPDUs cannot be triggered by a specified threshold size. Instead, the transmission of container IPDUs with a static container layout can be triggered by a container timeout or container IPDU triggering, for example.</p>
<b>Limitations for secure onboard communication</b>	<p><b>No automatic rejection of received secured IPDUs and authentic IPDUs</b> The Bus Manager processes received secured IPDUs and the related authentic IPDUs regardless of whether the received authentication information is verified and the verification indicates a communication error. To reject a secured IPDU and the related authentic IPDU in case of a communication error, you must verify the authentication information via the SecOC bus configuration feature and provide the verification result to a mapped behavior model that realizes the rejection.</p> <p><b>No support of multiple secured IPDUs per ECU for securing one authentic IPDU</b> The Bus Manager does not support multiple secured IPDUs per ECU that secure the payload of the same authentic IPDU. This applies regardless of whether the secured IPDU is configured as cryptographic IPDU. If two or more secured IPDUs of one ECU are assigned to the Simulated ECUs part of a bus</p>

configuration and these PDUs secure the payload of the same authentic IPDU, only the secured IPDU that was assigned first to the bus configuration is used to secure the payload of the authentic IPDU.

**No support of secured PDU headers for received secured IPDUs** The Bus Manager does not support secured PDU headers for received secured IPDUs. If the `useSecuredPduHeader` attribute is specified for a received secured IPDU and set to any value but `noHeader`, the related authentic IPDU cannot be decoded. Instead, only the raw data can be accessed, e.g., via experiment software.

**Limitations for using secure onboard communication on simulation platforms via bus simulation containers** When you configure secure onboard communication via the Bus Manager and generate bus simulation containers (BSC files), the configured secure onboard communication is included in the generated BSC files. BSC files are supported by various simulation platforms. It depends on the specific simulation platform whether the configured secure onboard communication is supported.

#### Limitations for end-to-end protection

The Bus Manager processes received IPDUs and the contained ISignal groups regardless of whether the received end-to-end protection information is evaluated and the evaluation indicates a communication error. To reject an IPDU and a contained ISignal group in case of a communication error, you must evaluate the end-to-end protection information via the **ISignal Group End-to-End Protection Status** bus configuration feature and provide the evaluation result to a mapped behavior model that realizes the rejection.

#### Limitations for global time synchronization

**Support as of AUTOSAR 4.3.0** The Bus Manager supports global time synchronization as of AUTOSAR 4.3.0. Older versions of global time synchronization are not supported.

**No support of offset time bases** The Bus Manager does not support offset time bases.

**No support of immediate time synchronization** The Bus Manager does not support the immediate time synchronization functionality.

**No support of debounce time** The Bus Manager does not support the debounce time functionality.

#### Limitations for coded and physical ISignal values

**No interface for coded ISignal values** The Bus Manager does not provide an interface for coded ISignal values. Therefore, you cannot use coded ISignal values from the behavior model or an experiment software in the Bus Manager. Instead, the Bus Manager accesses ISignals via their converted physical signal values.

**Only linear computation methods supported** To transmit or receive ISignals on the bus, physical signal values must be converted to coded values. The Bus Manager supports only linear computation methods for the conversion of physical signal values to coded values and vice versa. If no linear computation

method is defined, an ISignal value is not converted and no code can be generated.

**Limited precision for linear scalings** If a linear computation method that is specified for a signal is not an identical computation method, the scaling is performed with double precision arithmetic (IEEE 754). This results in at most 52 significant bits.

**No support of multiple computation scales** The Bus Manager does not support multiple computation scales for the conversion of physical signal values to coded values and vice versa. If the communication matrix defines multiple computation scales for an ISignal, the Bus Manager uses the first linear scale it finds and ignores all the following scales. However, in this case the conversion might not be deterministic.

**No code generation for ISignals without a data type for physical values** If the communication matrix does not define a data type for the physical value of an ISignal, the Bus Manager can determine a data type according to the computation scale specified for a coded ISignal. If the communication matrix specifies an inconsistent or unsupported computation method for a coded ISignal, the Bus Manager cannot determine a physical data type.

Because the data type of a function port cannot be undefined, the Bus Manager defines Double as the data type for the ISignal's function port in this case. However, no code can be generated for this ISignal and a conflict occurs for the affected bus configuration.

#### Limitations for array signals

**Limitations for data types and byte order formats** The Bus Manager only supports array signals (i.e., signals with a width > 1) of the following type:

- Data type: UInt8
- Byte order format: Opaque

Other data types and byte order formats are not supported for array signals.

**No support of dynamic length array signals** The Bus Manager does not support dynamic length array signals, i.e., array signals of `UINT8_DYN` data type.

**No support of computation methods** The Bus Manager does not support computation methods for array signals. If the communication matrix specifies computation methods for array signals, they are ignored by the Bus Manager.

#### No support for Scale Constraints and Invalid Values

The Bus Manager does not support Scale Constraints and Invalid Values.

#### No automatic usage of ISignal value limits specified in the communication matrix

The communication matrix can provide upper and lower signal value limits for TX ISignals. The Bus Manager cannot identify these values to use them as default minimum or maximum saturation values.

## Limitations for CAN Communication

<b>Limitations for transmitting CAN frames</b>	If more than one CAN frame is to be transmitted via one <a href="#">communication cluster</a> in one sampling step, the transmission sequence is undefined.
<b>Limitations for CAN frame variants</b>	<p>The Bus Manager does not support CAN frame variants within one communication cluster. CAN frame variants are frames with the same CAN identifier and addressing mode (standard or extended) but different frame-related settings (e.g., CAN FD support, bit rate switch, or frame length). If CAN frame variants are exchanged via a communication cluster at run time, their behavior is undefined.</p> <p>If you want to work with CAN frame variants for a communication cluster, you must configure the frames in one of the following ways:</p> <ul style="list-style-type: none"><li>▪ Assign each frame variant to a separate bus configuration and add the <a href="#">Bus Configuration Enable</a> feature to all the bus configurations. At run time, you must ensure that only one of the bus configurations is active at a time, i.e., the state of all other bus configurations must be set to 0: Disabled.</li><li>▪ Add the <a href="#">Frame Access</a> feature to the frame variants. At run time, you must specify unique frame settings for each frame variant.</li></ul> <p>Refer to <a href="#">Working with Bus Configuration Features</a> on page 115.</p>
<b>Limitations for decoding data bytes of received PDUs</b>	<p>For CAN PDUs, the number of data bytes that are transmitted on the bus might differ from the number of data bytes that is specified in the communication matrix. The Bus Manager can receive such PDUs. However, the following limitations apply in this case:</p> <ul style="list-style-type: none"><li>▪ If a received PDU has more data bytes than specified in the communication matrix, the Bus Manager decodes only the number of data bytes that is specified in the communication matrix. The other data bytes are ignored.</li><li>▪ The Bus Manager decodes signals only if they are received completely. Therefore, if a signal is only partially received because the related PDU is transmitted with fewer data bytes than specified in the communication matrix, the Bus Manager does not decode this signal at all.</li></ul> <p>However, in case of a J1939-compliant PDU that is transmitted using the Connection Mode Data Transfer protocol, the Bus Manager returns the number of data bytes that are actually received with the <i>End of Message Acknowledgment (EOM)</i>.</p>
<b>Limitations for CAN FD</b>	The Bus Manager does not support multiple PDUs per CAN FD frame. Each CAN FD frame must contain exactly one PDU.
<b>Limitations for J1939</b>	<b>No support of AUTOSAR system description files</b> The Bus Manager does not support J1939 information that is specified in AUTOSAR system description

files (ARXML files). If an ARXML file contains J1939 information, this information is ignored and not imported to the ConfigurationDesk application when you import the ARXML file. Instead, the Bus Manager supports only J1939 information that is specified in DBC files.

**No support of J1939 diagnostic communication manager**

**(J1939DCM)** The Bus Manager does not support the J1939 diagnostic communication manager according to J1939-73. If a DBC file specifies J1939 diagnostic messages, these messages are imported to the ConfigurationDesk application but treated as standard ISignal PDUs.

**No support of J1939 request manager** The Bus Manager does not support the J1939 request manager, i.e., the Bus Manager cannot automatically trigger the transmission of a TX PDU when a J1939 request message is received on the bus.

To trigger the transmission of a TX PDU with the receipt of a J1939 request message, you must configure this behavior manually. For example, you can do this as follows: Assign the RX PDU that contains the J1939 request message and the related TX PDU to the Simulated ECUs part of a bus configuration. Add the PDU RX Status feature to the RX PDU and the PDU Trigger feature to the TX PDU. In a mapped behavior model, you can model the triggering behavior by using the value of the State function port as the input source for the Trigger function port. For more information, refer to [Basics on Bus Configuration Features](#) on page 116.

**Limitations for the support of J1939 transport protocols** For J1939 transport protocol nodes (J1939 TP nodes) that are simulated by the Bus Manager, the following limitations apply:

- If a J1939 TP node uses the Broadcast Announce protocol to transmit J1939-compliant PDUs, only one PDU can be transmitted at a time.  
If two PDUs are to be transmitted by using the Broadcast Announce protocol, the transmission of the second PDU is delayed until the transmission of the first PDU has finished.
- If a J1939 TP node uses the Connection Mode Data Transfer protocol to transmit J1939-compliant PDUs to the same receiving J1939 TP node, only one PDU can be transmitted at a time.  
If two PDUs are to be transmitted to the same receiving J1939 TP node by using the Connection Mode Data Transfer protocol, the transmission of the second PDU is delayed until the transmission of the first PDU has finished.

**Restrictions for using J1939-compliant PDUs in bus**

**configurations** When you use J1939-compliant PDUs in bus configurations, the following restrictions apply:

- The Bus Manager does not support J1939-compliant PDUs for inspection. Therefore, you cannot assign J1939-compliant PDUs to the Inspection part.
- For manipulation, the Bus Manager supports only J1939-compliant PDUs with a payload length of up to 8 bytes. You can assign J1939-compliant PDUs with a payload length of more than 8 bytes to the Manipulation part but you cannot add bus manipulation features to the PDUs or their included ISignals.

---

<b>Limitations for extended signal multiplexing</b>	<p>For extended signal multiplexing with the Bus Manager, the following limitations apply:</p> <ul style="list-style-type: none"> <li>▪ Only scalar signals of integer type are supported as multiplexer signals. If required, you can modify a multiplexer signal to change its coded category and base data type. Refer to <a href="#">Basics on Modifying Communication Matrices</a> on page 267.</li> <li>▪ The Bus Manager does not check extended multiplexed IPDUs for overlapping signals.</li> <li>▪ Up to five nesting levels are supported. If extended multiplexed IPDUs with more nesting levels are assigned to a bus configuration, the run-time behavior is undefined.</li> <li>▪ Multiplexer values with up to 10 values or value ranges are supported. If multiplexed signals are assigned to a bus configuration to which more multiplexer values apply, the run-time behavior is undefined.</li> </ul>
---	--

## Limitations for LIN Communication

---

<b>Limitations for AUTOSAR files</b>	AUTOSAR Release 3.2.1 does not support collision resolver tables. Thus, resolving collisions of event-triggered frames by automatically executing a related collision resolver table is not possible with AUTOSAR files according to Release 3.2.1.
<b>Limitations for DBC files</b>	The DBC file format provides no definitions for master nodes or schedules. Because the Bus Manager supports only the transmission of LIN IPDUs according to a schedule defined by the LIN master, the Bus Manager does not support DBC files for describing a LIN bus system.
<b>Limitations for SAE J2602 support</b>	<p><b>Limitations for master parameters and node attributes of LDF files</b> J2602-compliant LDF files can contain optional master parameters and node attributes. The following parameters/attributes are not evaluated by the Bus Manager:</p> <ul style="list-style-type: none"> <li>▪ Master parameters <ul style="list-style-type: none"> <li>▪ max_header_length</li> <li>▪ response_tolerance</li> </ul> </li> <li>▪ Node attributes <ul style="list-style-type: none"> <li>▪ response_tolerance</li> <li>▪ wakeup_time</li> <li>▪ poweron_time</li> </ul> </li> </ul>

If an LDF file contains these parameters/attributes, they are ignored by the Bus Manager.

**Unused frame bits are dominant** In contrast to the SAE J2602 standard, unused bits of a frame are transmitted as dominant bits (0), not as recessive bits.

**ISignals included only once in a frame** The SAE J2602 standard allows to include ISignals more than once in a frame. The Bus Manager does not support such frames. Each ISignal of a frame must be included exactly once in that frame.

**Response tolerances limited by dSPACE hardware** The SAE J2602 standard allows response tolerances of less than 40% for LIN slaves. It cannot be ensured that response tolerances of less than 40% can be achieved for simulated LIN slaves on dSPACE hardware.

**Errors do not abort the transmission of frame headers and responses** In contrast to the SAE J2602 standard, the Bus Manager does not abort the transmission of frame headers or frame responses if an error occurs during the transmission.

**No automatic parameterization of J2602 status bytes** According to the SAE J2602 standard, the first data byte of each frame is a status byte. The Bus Manager does not parameterize the status bytes automatically (e.g., set a valid bit ID). The parameterization of the status bytes must be done via a mapped behavior model, for example.

**Limitations for J2602 error states** The Bus Manager does not support the J2602-compliant ID parity, framing, and bit error states. It is therefore not possible to detect these errors during run time by using the status bytes as intended by the SAE J2602 standard. To detect these errors at run time, you can use checksum algorithms instead.

**Limitations for J2602 functionalities** The Bus Manager does not automatically support the following J2602 functionalities:

- Targeted Reset
- Broadcast Reset
- Wake-up Timings

To use these functionalities, they must be modeled in a mapped behavior model.

**No automatic generation of errors due to sleep mode** According to the SAE J2602 standard, an error must be generated if sleep mode lasts for more than four seconds. The Bus Manager does not automatically generate an error in this case. The error generation must be modeled in a mapped behavior model.

---

#### Limitations for LIN schedule tables

**Maximum number of scheduled frames per LIN schedule table** A LIN schedule table can schedule the transmission of one or more frames. The Bus Manager supports only schedule tables that schedule the transmission of a maximum of 255 frames.

**Resume positions of interrupted LIN schedule tables specified in FIBEX files**

For interrupted LIN schedule tables, the FIBEX standard supports the following resume positions (e.g., after collision resolving):

- Start from the beginning.
- Start from the position where the schedule table was interrupted.
- Start from any specific position.

The Bus Manager does not support the start from any specific position. Instead, the Bus Manager treats the FIBEX-specific resume position like AUTOSAR-specific resume positions. This results in the following behavior:

- `ResumePosition = 0`: Start from the beginning.
- `ResumePosition ≥ 1`: Start from the position where the schedule table was interrupted.

**Resume positions of interrupted LIN schedule tables specified in LDF files**

LDF files do not support definitions for the resume position of interrupted LIN schedule tables. An interrupted schedule table always starts from the position where it was interrupted.

**Limitations for LIN transport protocol support**

The Bus Manager does not provide automatic support for the LIN transport protocol, which can be used for diagnostic purposes. You have to implement the related functions with basic features, such as standard master request and slave response frames.

**Limitations for slave node configuration****Missing parameters for slave node configuration result in sleep command**

The Bus Manager supports slave node configuration via master request frames. The required parameters for the slave node configuration must be specified in the communication matrix. If a required parameter is missing (e.g., `InitialNAD`), a sleep command is generated (i.e., the first data byte of the master request frame is set to 0). The next valid header that is sent wakes up the bus.

**No automatic slave node configuration** The Bus Manager does not support automatic slave node configuration. For example, if you work with off-the-shelf slave nodes, the LIN master uses special configuration schedule tables with configuration commands to set the network addresses and frame identifiers of these slave nodes. The Bus Manager does not support this way of slave node configuration. Instead, the Bus Manager uses the network addresses and frame identifiers that are defined in the communication matrix for these slave nodes.

**No transmission of slave node configuration commands according to an external LIN scheduling** The Bus Manager cannot transmit slave node configuration commands according to an external LIN scheduling. Instead, the transmission of slave node configuration commands must be scheduled by a LIN master that is assigned to a bus configuration.

<b>Limitations for frames and PDUs</b>	<p><b>Transmission of unconditional LIN frames</b> Unconditional LIN frames can be transmitted only according to a schedule defined by the LIN master. Timings defined for PDUs that are included in an unconditional LIN frame are ignored.</p> <p><b>Transmission of substituted frames of sporadic frames</b> The Bus Manager cannot transmit substituted frames of a sporadic frame according to an external LIN scheduling. Substituted frames of a sporadic frame are transmitted only if the transmission is scheduled by a LIN master that is assigned to a bus configuration.</p> <p><b>No support of multiplexed IPDUs</b> The Bus Manager does not support LIN multiplexed IPDUs.</p> <p><b>No support of LIN frame variants</b> The Bus Manager does not support LIN frame variants within one communication cluster. LIN frame variants are frames with identical LIN identifier but different frame-related settings (e.g., frame type, checksum type, or frame length). If LIN frame variants are exchanged via a communication cluster at run time, their behavior is undefined.</p> <p>If you want to work with LIN frame variants for a communication cluster, you must assign each frame variant to a separate bus configuration and add the Bus Configuration Enable feature to all the bus configurations. At run time, you must ensure that only one of the bus configurations is active at a time, i.e., the state of all other bus configurations must be set to 0: Disabled.</p> <p>Refer to <a href="#">Working with Bus Configuration Features</a> on page 115.</p>
<b>Limitations for simulating LIN slave responses</b>	<p>The Bus Manager does not support the simulation of more than one LIN slave response via one bus access at the same time.</p> <ul style="list-style-type: none"><li>▪ Effects on real-time applications: If two LIN slave responses are to be transmitted via one bus access at the same time, only the last response is sent at run time. If more than two LIN slave responses are to be transmitted, the real-time application stops and the web interface of the real-time hardware displays an error message in the message viewer.</li><li>▪ Effects on offline simulation applications: If two or more LIN slave responses are to be transmitted via one bus access at the same time, only the last response is sent at run time.</li></ul>

## Limitations for Bus Configuration Handling

---

<b>One bus configuration must run on one core</b>	<p>The Bus Manager supports multicore (MC) and multi-processing-unit (multi-PU) applications. However, a bus configuration must always be assigned to one core. You cannot split a bus configuration to run parts of it on different cores. Therefore, you must not connect one bus configuration to two or more behavior models that are assigned to different application processes, for example.</p>
---	---

<b>Limitations for replacing an assigned communication matrix</b>	When you replace an assigned communication matrix, existing ISignals can be identified only by their name. They cannot be identified by their start bits.
<b>Limitations for element names</b>	<p>Depending on the tool chain you use (operating system, experiment software, modeling tool, etc.), UTF-16 characters might not be supported. To avoid problems, use only alphabetic and numeric ASCII characters and underscores for names of communication matrix elements and bus configuration elements.</p> <p>Additionally, to avoid conflicts in the TRC file generation, do not use the following characters:</p> <ul style="list-style-type: none"> <li>▪ "</li> <li>▪ /</li> <li>▪ .</li> </ul> <p>For most bus configuration elements (e.g., ECUs, communication controllers, PDUs etc.), you cannot edit the names in the ConfigurationDesk application. For these elements, adhere to the restrictions above when specifying the names within the communication matrix.</p>
<b>No support for stop values</b>	ConfigurationDesk lets you use stop values for signals. Bus configurations do not support stop values.
<b>Limitation for event periods</b>	Depending on the configured bus communication, various events with preconfigured periods are generated automatically for each bus configuration. If you want to change a preconfigured period while you are working without a behavior model, the minimum supported period is 1 ms.
<b>Limitations for transmitting PDUs that are assigned to the Simulated ECUs part</b>	<p>The transmission of PDUs that are assigned to the Simulated ECUs part of a bus configuration is influenced by the related Bus Configuration task. Regarding the Bus Configuration task, the following limitations apply:</p> <ul style="list-style-type: none"> <li>▪ Triggers for transmitting PDUs cannot be queued. If there are two or more triggers before the first trigger is evaluated by the Bus Configuration task, only the last trigger is evaluated and the PDU is transmitted according to this trigger.</li> <li>▪ After the transmission of a PDU is triggered, data such as ISignal values is written to the PDU when the related Bus Configuration task evaluates the trigger. The PDU is transmitted with this data. If there is a delay between the trigger and its evaluation by the Bus Configuration task, this might not be the data when the trigger occurred.</li> </ul>
<b>Limitation for using physical ISignal values as initial values of model imports</b>	When you create model port blocks for function outports of ISignal values, the physical ISignal values that are specified in the communication matrix can be used as the initial value of related model imports. This is possible only if you

create model port blocks via the Generate New Simulink Model Interface command.

---

**Instantiating Bus Configuration function blocks via copy & paste not supported**

---

You cannot instantiate Bus Configuration function blocks via the Copy and Paste or Paste Multiple commands.

**Limitations for importing working views that contain Bus Configuration function blocks**

You can import working views to a ConfigurationDesk application via CAFX (ConfigurationDesk application fragment) files. When a CAFX file contains Bus Configuration function blocks, the following limitations apply.

**No support of CAFX files exported with Release 2020-B or earlier** You cannot import CAFX files that were exported with dSPACE Release 2020-B or earlier. If you try to do so, the import is aborted and a warning message is displayed.

**No support of merging or replacing signal chains in working views** You cannot import the CAFX file via the Merge Signal Chain into Working View or Replace Signal Chain in Working View commands. If you try to do so, the import is aborted and a warning message is displayed.

You can import the CAFX file only via the Add Signal Chain to Working View command.

**Limitations for communication matrices** For the used communication matrices, the following limitations apply:

- The communication matrices that were used to configure the Bus Configuration function blocks must be available in the Buses Browser. If one of the required communication matrices is missing, the import is aborted and a warning message is displayed.
- If user-defined ISignal IPDUs and/or ISignals were used to configure the Bus Configuration function blocks, you must add the same user-defined ISignal IPDUs and ISignals to the related communication matrix in the Buses Browser, i.e., the following settings must match:
  - Related higher-level elements
  - Names
  - Direction (TX or RX)If a required user-defined ISignal IPDU or ISignal is missing, the import is aborted and a warning message is displayed.
- User-defined settings of communication matrix elements are not included in CAFX files. If communication matrix elements with user-defined settings were used to configure Bus Configuration function blocks, e.g., PDUs with user-defined cyclic timings or ISignals with modified base data types, the user-defined settings are lost when you import a CAFX file. Instead, the original communication matrix settings are used.

**Limitations for the target ConfigurationDesk application** Importing a CAFX file that contains Bus Configuration function blocks to a ConfigurationDesk application might cause problems when you generate bus simulation containers later on. To avoid these problems, adhere to the following limitations:

- Do not import the CAFX file to the ConfigurationDesk application from which it was exported.
- Do not import the CAFX file to the same ConfigurationDesk application more than once.

## Limitations for Bus Simulation Containers

### Limitations for generating bus simulation containers

#### Only Simulink implementation container (SIC) files as model implementation

If you work with behavior models, you must work with Simulink implementation container (SIC) files as a [model implementation](#). Other file formats, such as FMU or SLX files, are not supported. If an application process contains a model implementation that is no SIC file, no bus simulation container is generated for this application process and the Message Viewer displays a warning message.

**Limitations for SIC files** If you work with behavior models, the following limitations apply for the required SIC files:

- The SIC files must be generated with the Model Implementation Package for Simulink of the same release as the Bus Manager.
- You cannot use SIC files that are generated with TargetLink.
- You cannot use precompiled SIC files.
- The SIC files must not contain A2L variable descriptions.

If an unsupported SIC file is assigned to an application process, no bus simulation container is generated for this application process and the Message Viewer displays a warning message.

**Only one behavior model per application process** You can generate bus simulation containers only for application processes to which no or exactly one behavior model is assigned. Multiple behavior models per application process are not supported for bus simulation container generation. If multiple behavior models are assigned to an application process, no bus simulation container is generated for this application process and the Message Viewer displays a warning message.

### Limitations for using BSC files in ConfigurationDesk applications

When you add BSC files as a model implementation to a ConfigurationDesk application, the following limitations apply:

**BSC files and ConfigurationDesk from different dSPACE Releases** You can add BSC files to the Model Browser only if the BSC files are generated with the same dSPACE Release as ConfigurationDesk.

**No support of BSC files containing SIC files generated for**

**dsrt64.tlc** BSC files that include SIC files generated for the **dsrt64.tlc** system target file are not supported by ConfigurationDesk. You can use such SIC files to create BSC files. However, you cannot add the generated BSC files to a ConfigurationDesk application.

**No MTFX export for elements of BSC files** ConfigurationDesk does not support exporting a complete model topology containing a BSC file into an MTFX file. All parts of the model topology related to the BSC file, i.e., all subsystems and all model port blocks of the BSC file, are ignored during an MTFX export.

**Naming restrictions for SIC files and bus configurations** When you use BSC files in a ConfigurationDesk application, you must observe the following naming restrictions:

- If a BSC file contains a Simulink implementation container (SIC file), the name of the [model implementation](#) that the SIC file describes must be unique in the application process to which the BSC file is assigned. This applies to the following:
  - All model implementations assigned to the application process.
  - All model implementations described by SIC files that are included in other BSC files assigned to the application process.
- The names of bus configurations that are included in a BSC file must be unique in the application process to which the BSC file is assigned. This applies to all BSC files and Bus Configuration function blocks that are assigned to the application process.

**Implementing the bus communication of BSC files in real-time**

**applications** When implementing the bus communication of a BSC file in a real-time application, the following restrictions apply:

- You must map all the Configuration ports of a BSC file to suitable bus function blocks (CAN, LIN) to implement the bus communication in a real-time application.
- You must not map the Configuration ports of a BSC file to bus function blocks that are used in one of the following ways:
  - The bus function block specifies the [bus access](#) for [bus access requests](#) of bus configurations.
  - The bus function block is referenced by one or more ECU Interface Configuration function blocks.
- If a BSC file contains Configuration ports resulting from LIN bus access requests, at least one LIN function block must be assigned to the same application process as the BSC file. This applies even if you do not want to implement LIN communication in the real-time application.

**Limitations for using bus simulation containers on simulation platforms**

Bus simulation containers can be used only on dSPACE platforms (e.g., VEOS, SCALEXIO systems), including dSPACE platforms that support RTMaps, such as MicroAutoBox Embedded PC or MicroAutoBox Embedded SPU. Bus simulation containers cannot be used on third-party platforms.

---

**Limitations for using BSC files with VEOS**

When you use BSC files with VEOS, the following limitations apply:

- To use a BSC file in an offline simulation application with VEOS running on a Linux operating system, the BSC file must contain a behavior model. The behavior model must be an SIC file generated for the `dsrt64.tlc` system target file. BSC files without a behavior model or containing an SIC file generated for another system target file are not supported.
  - You can use BSC files that contain SIC files generated for the `dsrt64.tlc` system target file only with VEOS running on a Linux operating system. You cannot use such BSC files with VEOS running on a Windows operating system.
  - When an offline simulation application that contains one or more imported BSC files runs on VEOS and you stop and restart the application, the simulation results might be incorrect. To avoid this, you must unload the offline simulation application before you restart it.
- 

**Limitations for using automation scripts for offline simulation and real-time simulation**

Task information variables are located at different places in offline simulation applications and real-time applications. Therefore, you must adapt automation scripts for VEOS and SCALEXIO/MicroAutoBox III if the scripts unconditionally evaluate CallCounter, OverrunCount, or TurnaroundTime variables.



# Appendix

## Where to go from here

## Information in this section

Mapping of Bus Manager PDU Types to Communication Standard PDU Types.....	335
Optimized Set of Variables for Bus Configuration Function Ports.....	338
Resolving the Bus Configuration Conflict: No Valid Application Process Assigned.....	341
How to Assign Behavior Models to Separate Application Processes.....	344

## Mapping of Bus Manager PDU Types to Communication Standard PDU Types

### Introduction

The Bus Manager supports various PDU types according to AUTOSAR, FIBEX, DBC, and LDF. To provide a consistent representation of the supported PDU types, the Bus Manager uses PDU-specific element types that are derived from the AUTOSAR PDU entities. PDUs of other communication standards are mapped to these element types.

### Mapping of PDU types

The following table provides an overview of the mapping of the PDU types in the Bus Manager to the PDU types provided by the AUTOSAR, FIBEX, DBC, and LDF standards.

Bus Manager PDU Types			AUTOSAR		FIBEX / DBC / LDF
PDU Element Type	Category	Diagnostic PDU Type/ XCP Configuration	Entity	Category	
Bus Container IPDU	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	ContainerIPdu	–	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
Bus DCM IPDU	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: DIAG-REQUEST</li> <li>▪ XCP Configuration: –</li> </ul>	DcmIPdu	DIAG-REQUEST (diagPduType property)	<ul style="list-style-type: none"> <li>▪ FIBEX: DIAG-REQUEST</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: DIAG-RESPONSE</li> <li>▪ XCP Configuration: –</li> </ul>	DcmIPdu	DIAG-RESPONSE (diagPduType property)	<ul style="list-style-type: none"> <li>▪ FIBEX: DIAG-RESPONSE</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
Bus Extended Multiplexed IPDU	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	–	–	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: Signal attribute: SG_MUL_VAL_</li> <li>▪ LDF: –</li> </ul>
Bus General-Purpose IPDU	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	GeneralPurposePdu	DLT	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	GeneralPurposePdu	SOMEIP_SEGMENTED_IPDU	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	XCP	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: XCP_PRE_CONFIGURED</li> </ul>	GeneralPurposePdu	XCP (plus XCP_PRE_CONFIGURED frame category)	<ul style="list-style-type: none"> <li>▪ FIBEX: XCP_PRE_CONFIGURED</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	XCP	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: XCP_RUNTIME_CONFIGURED</li> </ul>	GeneralPurposePdu	XCP (plus XCP_RUNTIME_CONFIGURED frame category)	<ul style="list-style-type: none"> <li>▪ FIBEX: XCP_RUNTIME_CONFIGURED</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
Bus General-Purpose PDU	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	GeneralPurposePdu	DolP	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	GeneralPurposePdu	SD	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	GLOBAL_TIME	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	GeneralPurposePdu	GLOBAL_TIME	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
Bus ISignal IPDU	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	ISignalIPdu	–	<ul style="list-style-type: none"> <li>▪ FIBEX: APPLICATION</li> <li>▪ DBC: BO_</li> <li>▪ LDF: Frame</li> </ul>
Bus Multiplexed IPDU	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	MultiplexedIPdu	–	<ul style="list-style-type: none"> <li>▪ FIBEX: APPLICATION</li> <li>▪ DBC: BO_ with mode signals</li> <li>▪ LDF: –</li> </ul>
Bus NMPPDU	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	NmPdu	–	<ul style="list-style-type: none"> <li>▪ FIBEX: NM</li> <li>▪ DBC: –</li> </ul>

Bus Manager PDU Types			AUTOSAR		FIBEX / DBC / LDF
PDU Element Type	Category	Diagnostic PDU Type/ XCP Configuration	Entity	Category	
					<ul style="list-style-type: none"> <li>▪ LDF: –</li> </ul>
Bus NPDU	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	NPdu	–	<ul style="list-style-type: none"> <li>▪ FIBEX: TPL</li> <li>▪ DBC: –</li> <li>▪ LDF: Diagnostic_frames (MasterReq, SlaveResp)</li> </ul>
Bus Secured IPDU	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	SecuredIPdu	–	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
Bus User-Defined IPDU	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	UserDefinedIPdu	–	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	BAP	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	UserDefinedIPdu	BAP (cdd type property)	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	DIAG-STATE	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	UserDefinedIPdu	DIAG-STATE (cdd type property)	<ul style="list-style-type: none"> <li>▪ FIBEX: DIAG-STATE</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	SERVICE	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	UserDefinedIPdu	SERVICE (cdd type property)	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	XCP	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: XCP_PRE_CONFIGURED</li> </ul>	UserDefinedIPdu	XCP (cdd type property) plus XCP_PRE_CONFIGURED (frame category)	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	XCP	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: XCP_RUNTIME_CONFIGURED</li> </ul>	UserDefinedIPdu	XCP (cdd type property) plus XCP_RUNTIME_CONFIGURED (frame category)	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
Bus User-Defined PDU	–	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	UserDefinedPdu	–	<ul style="list-style-type: none"> <li>▪ FIBEX: OTHER</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	BAP	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	UserDefinedPdu	BAP (cdd type property)	<ul style="list-style-type: none"> <li>▪ FIBEX: BAP</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	DIAG-STATE	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	UserDefinedPdu	DIAG-STATE (cdd type property)	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	SERVICE	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: –</li> </ul>	UserDefinedPdu	SERVICE (cdd type property)	<ul style="list-style-type: none"> <li>▪ FIBEX: SERVICE</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>
	XCP	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: XCP_PRE_CONFIGURED</li> </ul>	UserDefinedPdu	XCP (cdd type property) plus XCP_PRE_	<ul style="list-style-type: none"> <li>▪ FIBEX: –</li> <li>▪ DBC: –</li> <li>▪ LDF: –</li> </ul>

Bus Manager PDU Types			AUTOSAR		FIBEX / DBC / LDF
PDU Element Type	Category	Diagnostic PDU Type/ XCP Configuration	Entity	Category	
				CONFIGURED (frame category)	
XCP	<ul style="list-style-type: none"> <li>▪ Diagnostic PDU Type: –</li> <li>▪ XCP Configuration: XCP_RUNTIME_CONFIGURED</li> </ul>	UserDefinedIPdu	XCP (cdd type property) plus XCP_RUNTIME_CONFIGURED (frame category)		

**Tip**

In automation scripts, you can access the Bus Manager PDU types via their primary role. The primary role of a PDU is the PDU element type without spaces or hyphens, and adapted capitalization, for example:

- BusISignalIPdu
- BusContainerIPdu
- BusGeneralPurposeIPdu

For more information, refer to [Automating Bus Manager Features \(ConfigurationDesk Automating Tool Handling\)](#).

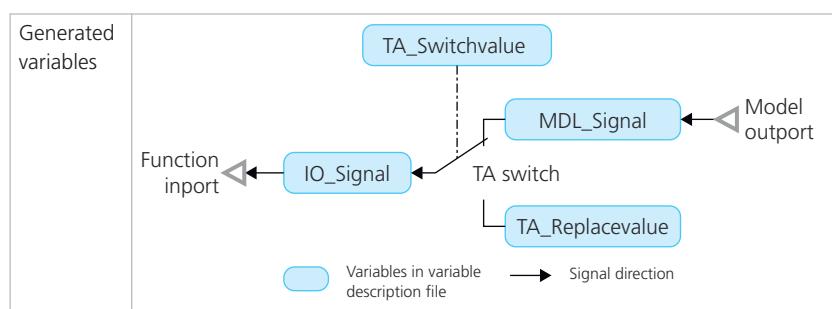
## Optimized Set of Variables for Bus Configuration Function Ports

### Introduction

For each function port with enabled test automation support, variables are generated into the variable description file when you generate bus simulation containers. When the Variable description property of a bus configuration is set to Optimized, an optimized set of variables is generated. The generated variables depend on various aspects, such as the function port type and the mapping to a model port.

### Function imports mapped to model ports

For function imports that are mapped to model ports, the following applies:



Initial function port value	Depends on the setting of the Initial switch setting property, which is used as the initial value of the <code>TA_Switchvalue</code> variable.
Description	You can switch the signal source between the original signal ( <code>MDL_Signal</code> ) and the substitute signal ( <code>TA_Replacevalue</code> ).

**Function imports not mapped to model ports** For function imports that are not mapped to model ports, the generation of variables depends on the following cases:

**Case A** For case A, the following applies:

Condition	Model access of the function port set to Disabled.
Generated variables	<p>The diagram shows a horizontal flow of signals. On the left is a blue rounded rectangle labeled "Function inport". To its right is a blue rounded rectangle labeled "IO_Signal". To the right of "IO_Signal" is another blue rounded rectangle labeled "TA_Replacevalue". Two arrows point from "TA_Replacevalue" to "IO_Signal": one is a solid arrow pointing left, and the other is a dashed arrow pointing right. Below the diagram is a legend: a blue rounded rectangle with a white center is labeled "Variables in variable description file", and a blue arrow pointing right is labeled "Signal direction".</p>
Initial function port value	Value specified for the Initial substitute value property, which is used as the initial value of the <code>TA_Replacevalue</code> variable.
Description	The signal source of the function port is the substitute signal ( <code>TA_Replacevalue</code> ). The signal source cannot be changed. The setting of the Initial switch setting property has no effects on the function port.

**Case B** For case B, the following applies:

Condition	Model access of the function port set to Enabled.
Generated variables	<p>The diagram shows a more complex signal flow. At the top is a blue rounded rectangle labeled "TA_Switchvalue". A dashed arrow points from "TA_Switchvalue" down to a blue rounded rectangle labeled "MDL_Signal". From "MDL_Signal", a solid arrow points left to a blue rounded rectangle labeled "IO_Signal". To the right of "IO_Signal" is another blue rounded rectangle labeled "TA_Replacevalue". A legend below the diagram shows a blue rounded rectangle with a white center and a blue arrow pointing right, both labeled "Variables in variable description file" and "Signal direction" respectively.</p>
Initial function port value	Depends on the setting of the Initial switch setting property, which is used as the initial value of the <code>TA_Switchvalue</code> variable.
Description	You can switch the signal source between the original signal ( <code>MDL_Signal</code> ) and the substitute signal ( <code>TA_Replacevalue</code> ). The value specified for the

Initial value property of the function port is used as the original signal value.

### Tip

Function ports that are not mapped to model ports are available at the port interface of bus simulation containers. Because all variables are generated for the function port, you can use it without restrictions, e.g., if you use the bus simulation container in ConfigurationDesk or the VEOS Player. For more information, refer to [Port Interface of Bus Simulation Containers](#) on page 256.

#### Function outports mapped to model ports

For function outports that are mapped to model ports, the following applies:

Generated variables	
Initial function port value	Value specified for the Initial value property, which is used as the initial value of the <code>IO_Signal</code> variable.
Description	The signal source of the function port is the original signal ( <code>IO_Signal</code> ). This signal value is provided to the model port. The signal source cannot be changed. The setting of the Initial switch setting property has no effects on the function port.

#### Function outports not mapped to model ports

For function outports that are not mapped to model ports, the following applies:

Generated variables	
Initial function port value	Value specified for the Initial value property, which is used as the initial value of the <code>IO_Signal</code> variable.
Description	The signal source of the function port is the original signal ( <code>IO_Signal</code> ). The signal source cannot be changed. The setting of the Initial switch setting property has no effects on the function port.

### Tip

Because the function port is not mapped to a model port, there is no recipient for the function port value in the behavior model. Therefore, the generated variables can be reduced to a minimum. However, you can access the function port via experiment software, and the function port is available at the port interface of generated bus simulation containers.

340

Bus Manager (Stand-Alone) Implementation Guide

May 2021

---

<b>Restrictions for the optimized set of variables</b>	For the function ports of the following bus configuration features, no optimized set of variables can be generated: <ul style="list-style-type: none"> <li>▪ Frame Capture Data feature</li> <li>▪ Filter Control feature</li> <li>▪ LIN Schedule Table feature</li> </ul> Instead, the complete set of variables is always generated for the function ports of these features, even if the Variable description property is set to Optimized.
<b>Further information</b>	For more information, refer to <a href="#">Accessing Function Ports with Enabled Test Automation Support in Variable Description Files</a> on page 124.

---

## Resolving the Bus Configuration Conflict: No Valid Application Process Assigned

---

<b>Overview</b>	This conflict occurs if a bus configuration is assigned to no or more than one application process. Therefore, this conflict occurs each time you start implementing bus communication in the signal chain because the bus configurations are not assigned to an application process yet. In most cases, you resolve this conflict automatically when you complete implementing the bus communication in the signal chain (e.g., as described in <a href="#">Implementing Bus Communication in the Signal Chain Using the Bus Manager</a> on page 233). If the conflict is not resolved after you completed the implementation of the bus communication in the signal chain or the conflict occurs after you modified the implemented bus communication, the conflict causes and remedies that apply to your case depend on the following use scenarios: <ul style="list-style-type: none"> <li>▪ <a href="#">Use scenario 1: Working without a behavior model</a> on page 341</li> <li>▪ <a href="#">Use scenario 2: Working with a behavior model</a> on page 342</li> <li>▪ <a href="#">Use scenario 3: Working with multiple behavior models (multicore application)</a> on page 343</li> </ul>
<b>Use scenario 1: Working without a behavior model</b>	<p><b>Conflict cause</b> If you work without a behavior model, a possible conflict cause is that no application process is available.</p> <p><b>Remedy</b> Assign the bus configuration to one application process. For example, perform the following actions:</p> <ul style="list-style-type: none"> <li>▪ Create an application process with a default task. Refer to <a href="#">How to Create Application Processes that Provide Default Tasks</a> on page 264.</li> <li>▪ Assign the bus configuration to the application process. Refer to <a href="#">How to Manually Assign Bus Configurations to Application Processes</a> on page 265.</li> </ul>

---

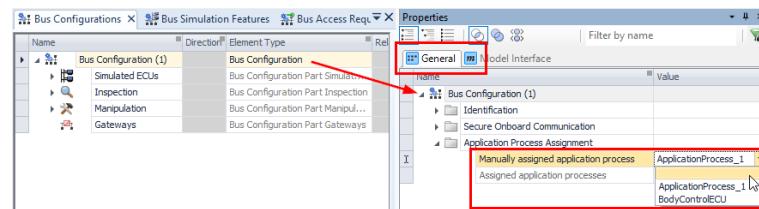
**Use scenario 2: Working with a behavior model**

**Conflict cause** If you work with a behavior model, possible conflict causes are:

- The behavior model is not available in the ConfigurationDesk application.
- The behavior model is available in the ConfigurationDesk application, but no application process exists.
- None of the behavior model's data ports are mapped to function ports of the bus configuration.
- A data port of the behavior model is mapped to a function port of the bus configuration. Due to a manually assigned application process, the bus configuration is assigned to two application processes, to the manually assigned application process and to the application process of the behavior model.

**Remedy** Assign the bus configuration to one application process. For example, perform the following actions:

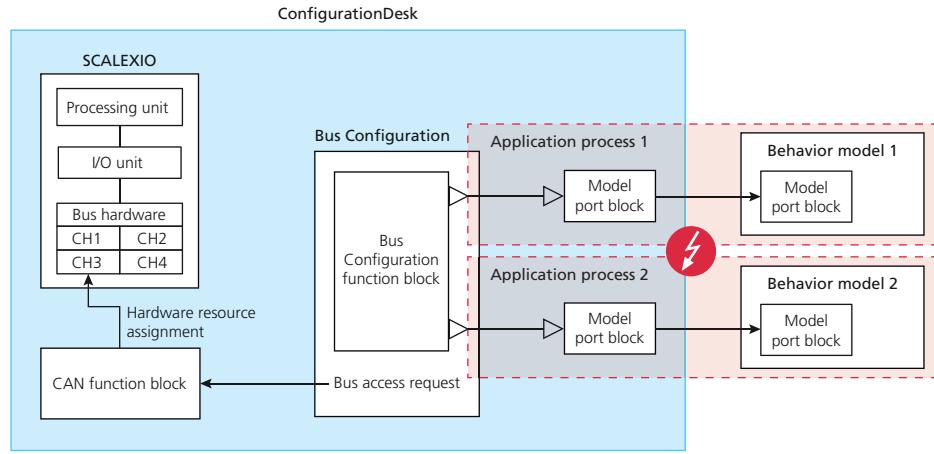
- If not already done, add the behavior model to the ConfigurationDesk application. Refer to [How to Add Behavior Models to a ConfigurationDesk Application](#) on page 244.
- Ensure that an application process is created: Right-click the model in the Model Browser and select **Create Preconfigured Application Process** from the context menu.
- Map at least one data port of the behavior model to a function port of the bus configuration. For an example of mapping ports to bus configurations, refer to [Example of Mapping Model Ports to Bus Configuration Ports via Tables](#) on page 249.
- Undo the manual assignment to an application process: Select the bus configuration, for example, in the Bus Configurations table and select the blank entry from the list of the **Manually assigned application process** property in the Properties Browser.



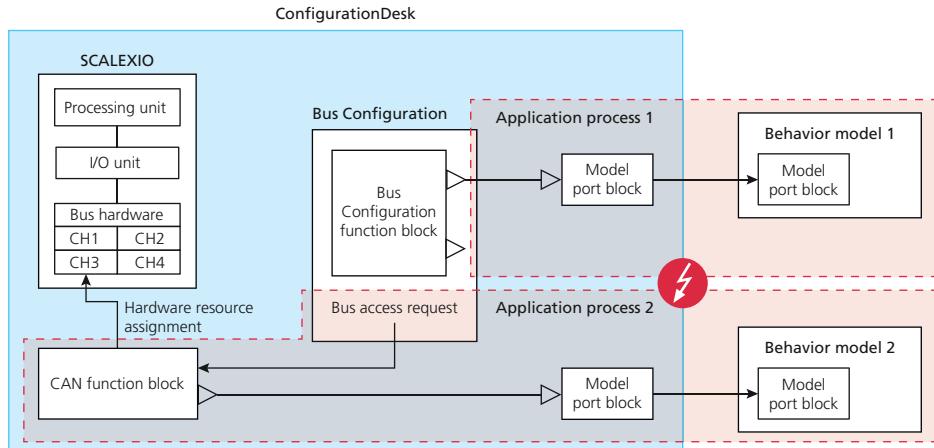
**Use scenario 3: Working with multiple behavior models (multicore application)**

**Conflict cause** If you work with multiple behavior models, possible conflict causes are:

- The bus configuration is mapped to two or more behavior models that are assigned to different application processes, as shown in the following example.



- The bus configuration and an assigned bus function block (CAN, LIN) are mapped to different behavior models that are assigned to different application processes, as shown in the following example.



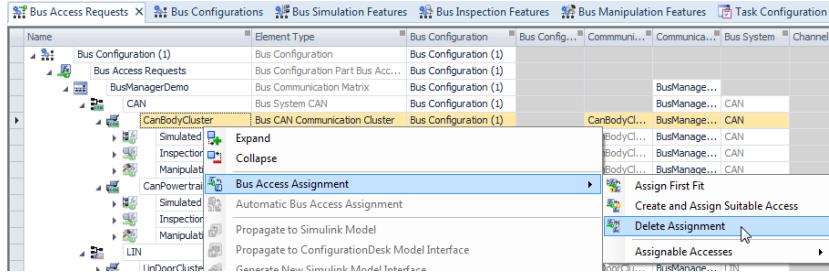
**Tip**

You can identify the assignment of behavior models to application processes via the Build Configuration table, for example.

**Remedy** Assign the bus configuration to one application process. For example, perform the following actions:

- Map all the function ports of the bus configuration to model ports of only one behavior model.

- Remove the assigned bus accesses: Open the Bus Access Requests table, e.g., via Switch Controlbars on the View ribbon. In the Bus Access Requests table, right-click the related communication cluster node and select Bus Access Assignment - Delete Assignment from the context menu.



## Further information

For more details on application processes and tasks, refer to [Modeling Executable Applications and Tasks \(ConfigurationDesk Real-Time Implementation Guide\)](#).

## How to Assign Behavior Models to Separate Application Processes

### Objective

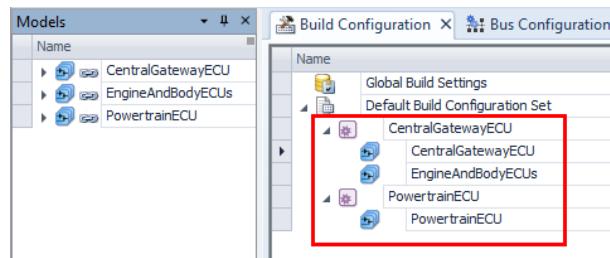
To generate a bus simulation container for an application process no or exactly one behavior model must be assigned to the application process. The behavior model must be a Simulink implementation container. If two or more behavior models are assigned to the application process, you must change the assignment and assign each behavior model to a separate application process before you can generate bus simulation containers.

### Method

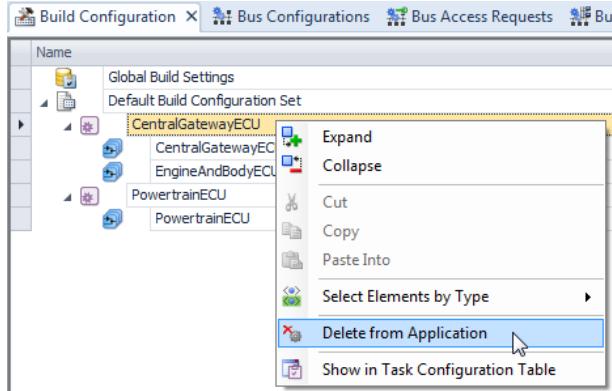
#### To assign behavior models to separate application processes

- Open the Build Configuration table, for example, via Switch Controlbars on the View ribbon.

The Build Configuration table displays the available application processes and the assigned behavior models, as shown in the following example.



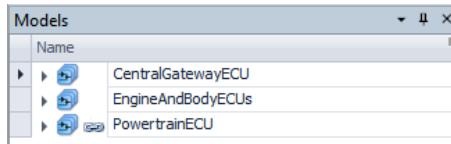
- 2** Right-click an application process to which multiple behavior models are assigned and select **Delete from Application** from the context menu.



The application process is deleted from the ConfigurationDesk application.

- 3** Repeat step 2 for all application processes to which multiple behavior models are assigned.  
**4** Open the Model Browser.

The Model Browser displays the available behavior models. Behavior models that are assigned to application processes are indicated by a chain symbol.



- 5** Right-click a behavior model that is not assigned to an application process and select **Create Preconfigured Application Process - In Existing Processing Unit Application - <name of processing unit application>** from the context menu.

A new application process is created and the behavior model is assigned to this application process. The application process is named after the behavior model. If model ports of the behavior model are mapped to function ports of a bus configuration, the bus configuration is automatically assigned to the application process.

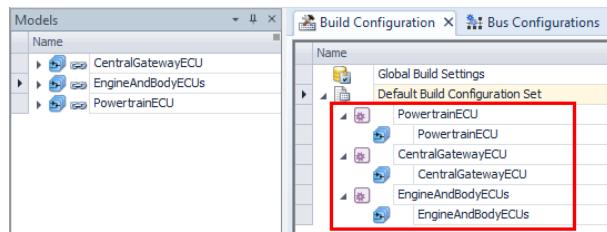
- 6** Repeat step 5 for all behavior models that are not assigned to application processes.

#### Note

If function ports of a bus configuration are mapped to model ports of different behavior models, conflicts occur. You must resolve the conflicts before you can generate bus simulation containers. To do so, map the function ports of the bus configuration to only one behavior model.

**Result**

Each behavior model is assigned to a separate application process.



**Next step**

You can now generate bus simulation containers. Refer to [How to Generate Bus Simulation Containers](#) on page 259.

# 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.....	348
B.....	348
C.....	351
D.....	354
E.....	355
F.....	357
G.....	358
H.....	358
I.....	359
L.....	360
M.....	361
N.....	364
O.....	364
P.....	364
R.....	366
S.....	367
T.....	368
U.....	369

V.....	370
W.....	370
X.....	371

## 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 PDUs](#) 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 Import blocks which are connected to Data Outport blocks. Hence, it also bundles the signals that are received by these blocks. If Data Import blocks are executed within the same [task](#) and belong to the same [communication package](#), their data imports are read simultaneously. If Data Outport blocks that are connected to the Data Import 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 outports to the behavior model.

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

**Data outport** A port that supplies data from behavior model signals to ConfigurationDesk's function imports.

In a multimodel application, data outports also can be used to supply data to a data import 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](#) in the [signal chain](#). 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 import** 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 outport** 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 import

(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 Outport and Data Import 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 import
- 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 imports 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 imports](#) 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., [ISignal 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.

## S

**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 import** 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 Import block or a Data Outport 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.

---

**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](#).

## X

---

**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.



**A**

adding  
 behavior model 244  
 bus communication to signal chain 238  
 bus configuration 236  
 communication matrix 233  
 application process 263  
 assigning  
 communication matrix to bus  
 configuration 238  
 authentic IPDUs 41  
 automation interface 97

**B**

behavior model  
 generating bus simulation containers with a behavior model 33  
 generating bus simulation containers without a behavior model 34  
 working without a behavior model 263  
 BSC 253  
 generating 259  
 bus access 91  
 bus access request 91  
 bus simulation container 255  
 Bus Access Requests bus configuration part 69  
 bus communication defined in communication matrix 19  
 bus configuration  
 adjusting to a changed communication matrix 291  
 assigning communication matrix elements 75  
 basics 63  
 Bus Access Requests part 69  
 configurable node names 65  
 deprecated node 291  
 features 116  
 filter for mappable ports 71  
 frame capture 66  
 function block 71  
 Gateways part 66  
 handling conflicts 99  
 Inspection part 66  
 Manipulation part 66  
 parts 66  
 removing communication matrix elements 83  
 Simulated ECUs part 66  
 specifying frame captures 107  
 specifying gateways 109  
 tables 66  
 Bus Configuration Enable feature 230  
 bus configuration features  
 adding 118  
 basics 116  
 Bus Configuration Enable 230  
 Communication Controller Enable 203  
 configuring function ports 120  
 Counter Signal 141

Filter Control 227  
 Frame Access 190  
 Frame Capture Data 222  
 Frame Gateway Direction 226  
 Frame Length 197  
 GTS Time Base Data 216  
 GTS Transmission Control 214  
 GTS Validation 218  
 ISignal Group End-to-End Protection Status 152  
 ISignal Offset Value 149  
 ISignal Overwrite Value 146  
 ISignal Value 139  
 LIN Schedule Table 207  
 LIN Wake-Up 205  
 manipulating bus communication 131  
 PDU Cyclic Timing Control 159  
 PDU Enable 155  
 PDU Length 165  
 PDU Raw Data 157  
 PDU RX Interrupt 183  
 PDU RX Status 167  
 PDU Trigger 162  
 PDU User Code 170  
 SecOC 175  
 SecOC Authenticator Invalidation 177  
 SecOC Freshness Overwrite Value 180  
 Suspend Frame Transmission 200  
 Bus Custom Code interface 84  
 Bus Manager  
 automation interface 97  
 CAN bus features 47  
 container IPDUs 39  
 end-to-end protection 43  
 events 94  
 gateway use scenarios 23  
 global time synchronization 43  
 implementing secure onboard communication 87  
 inspection use scenarios 22  
 LIN bus features 52  
 manipulation use scenarios 22  
 multiplexed IPDUs 39  
 overview 15  
 runnable functions 94  
 secure onboard communication 41  
 simulation use scenarios 20  
 supported PDU types 37  
 supported signal data types 37  
 tasks 94  
 user code 84  
 variants 15  
 bus simulation container  
 basics 253  
 bus access request 255  
 generating 259  
 port interface 256

**C**

CAN  
 aspects of supported features 47

CAN FD 47  
 container IPDUs 39  
 extended signal multiplexing 51  
 J1939 48  
 multiplexed IPDUs 39  
 secure onboard communication 41  
 cluster 19  
 coded signal value 54  
 Common Program Data folder 10, 352  
 Communication Controller Enable feature 203  
 communication matrix  
 adding elements 267  
 assigning elements to bus configurations 75  
 basics 58  
 configurable elements 267  
 defining bus communication 19  
 generating missing ECUs 60  
 removing elements from bus configurations 83  
 replacing 291  
 supported file formats 58  
 user-defined settings 267  
 computation methods 54  
 configurable node names 65  
 conflicts  
 bus configuration 99  
 communication matrix 99  
 container IPDUs 39  
 Counter Signal feature 141  
 cryptographic IPDUs 41

**D**

deprecated communication matrix node 291  
 Documents folder 10, 354  
 dynamic simulation 20

**E**

E2E protection  
 Bus Manager 43  
 ECU simulation 20  
 ECUs  
 generating during communication matrix import 60  
 element identification via node names 65  
 end-to-end protection  
 Bus Manager 43  
 events  
 provided by the Bus Manager 94  
 extended signal multiplexing 51

**F**

failure gateway 23  
 Filter Control feature 227  
 filters  
 communication matrix 62  
 Frame Access feature 190  
 frame capture 66  
 Frame Capture Data feature 222  
 frame captures  
 filters 111

- specifying 107  
**Frame Gateway Direction feature** 226  
**frame gateways**  
  filters 111  
  specifying 109  
**Frame Length feature** 197  
**function ports**  
  configuring 120
- G**
- gateways**  
  filters 111  
  specifying 109  
  use scenarios 23  
**Gateways bus configuration part** 66  
**generating**  
  bus simulation container 259  
**global time synchronization**  
  Bus Manager 43  
**GTS Time Base Data feature** 216  
**GTS Transmission Control feature** 214  
**GTS Validation feature** 218
- I**
- initial values 120  
**inspection**  
  use scenarios 22  
**Inspection bus configuration part** 66  
**instantiating**  
  PDUs and ISignals 81  
**ISignal**  
  instance 81  
  signal conversion 54  
**ISignal Group End-to-End Protection Status feature** 152  
**ISignal Offset Value feature** 149  
**ISignal Overwrite Value feature** 146  
**ISignal Value feature** 139
- J**
- J1939  
  basics 48  
  bridge ECUs 60
- L**
- LIN  
  aspects of supported features 52  
**LIN Schedule Table feature** 207  
**LIN Wake-Up feature** 205  
**Local Program Data folder** 10, 360
- M**
- manipulating bus communication 131  
  execution sequence 137  
  recalculate end-to-end protection information 135  
  recalculate SecOC information 135  
**manipulation**
- use scenarios 22  
**Manipulation bus configuration part** 66  
**model access** 121  
  enabling 242  
**multiplexed IPDUs** 39
- N**
- network node 19  
**node names** 65
- O**
- overview of the Bus Manager 15
- P**
- PDU**  
  instance 81  
  PDU Cyclic Timing Control feature 159  
  PDU Enable feature 155  
  PDU Length feature 165  
  PDU Raw Data feature 157  
  PDU RX Interrupt feature 183  
  PDU RX Status feature 167  
  PDU Trigger feature 162  
  PDU User Code feature 170  
  physical signal value 54  
**port interface**  
  bus simulation container 256
- R**
- recalculate  
  end-to-end protection information 135  
  SecOC information 135  
**replacing a communication matrix** 291  
**restbus simulation** 20  
**runnable functions**  
  provided by the Bus Manager 94
- S**
- saturation values 123  
**SecOC Authenticator Invalidation feature** 177  
**SecOC feature** 175  
**SecOC Freshness Overwrite Value feature** 180  
**secure onboard communication**  
  Bus Manager 41  
  implementing in executable applications 87  
**secured IPDUs** 41  
**Simulated ECUs bus configuration part** 66  
**simulation**  
  dynamic 20  
  ECU 20  
  restbus 20  
  static 20  
  use scenarios 20  
**static simulation** 20  
**Suspend Frame Transmission feature** 200
- T**
- tasks
- provided by the Bus Manager 94  
**test automation support** 122
- U**
- use scenarios of the Bus Manager 20  
**user code** 84  
**user interface**  
  Bus Manager-specific elements 27
- V**
- variants of the Bus Manager 15
- W**
- workflow**  
  configuring bus communication and generating bus simulation containers 31  
  generating bus simulation containers with a behavior model 33  
  generating bus simulation containers without a behavior model 34  
  working without a behavior model 263