

dSPACE XIL API Implementation

# Guide

For dSPACE XIL API .NET 2021-A

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.

© 2014 - 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 Document	7
Introduction	11
Contents of the Installed Software Package.....	11
General Concepts of the ASAM AE XIL Standard.....	13
Features of the dSPACE XIL API Implementation.....	16
Required Software and Hardware.....	17
Protecting Long-Term Tests Against Interrupting or Blocking.....	21
Using dSPACE XIL API .NET Implementation	23
Basics on Creating XIL API Client Applications.....	23
How to Create XIL API C# Projects.....	25
Using XIL API with Python.....	26
Using XIL API with MATLAB.....	28
Using the Framework	31
Basics on the Framework.....	31
Using the Testbench	35
Basics on the Testbench.....	35
Implementing an MAPort Application	39
Basics on the MAPort.....	39
Implementing an MAPort Client Application.....	44
Demo Projects for MAPort Client Applications.....	49
Implementing an EESPort Client Application	53
Basic Information on the EESPort Implementation.....	54
Basics on the EESPort.....	54
Creating dSPACE EESPort Configuration Files.....	58
Using the EESPortConfiguration API.....	62
Latencies when Performing Electrical Error Simulation.....	68

Monitoring the Switching Behavior of Electrical Error Simulation Hardware.....	69
dSPACE XIL API EESPort Demo.....	73
Basic Information on Simulating Electrical Errors in the Wiring.....	75
Basics on Failure Simulation.....	75
Basics on Failure Classes.....	76
Electrical Error Simulation with a Discrete FIU.....	76
Electrical Error Simulation with the Integrated SCALEXIO FIU.....	78
Safety Precautions for Simulating Electrical Errors with a SCALEXIO System.....	83
Hardware for Failure Simulation.....	85
Overview of the Failure Simulation Hardware.....	85
DS291 FIU Module.....	86
DS293 FIU Module, DS282 Load Module and DS289MK RSim Module.....	87
DS749 FIU Module.....	90
DS789 Sensor FIU Module.....	90
DS791 Actuator FIU Module.....	92
DS793 Sensor FIU.....	93
DS5355/DS5390 High Current FIU System.....	95
Hardware for Electrical Error Simulation on SCALEXIO Systems.....	96
Defining Failure Classes with Signal Files.....	99
Structure of Signal Files.....	100
Comment Area.....	101
Format Area.....	102
Signallist Area.....	103
How to Configure Signal Files.....	106
Evaluating Signal Files.....	107
Creating Error Configurations.....	108
Basic Information on Configuring Errors.....	108
Specifying Errors.....	111
Activating Electrical Error Simulation.....	114
Updating Electrical Error Simulation.....	116
Deactivating Electrical Error.....	117
Simulating Unstable Pin Failures (Loose Contacts) with DS793 Modules and SCALEXIO Systems.....	118
Simulating Loose Contacts or Switch Bouncing.....	118

<b>Working with XIL API .NET on Linux</b>	<b>121</b>
Getting Started with XIL API .NET on Linux.....	122
Introduction to the dSPACE XIL API .NET on Linux.....	122
How to Install the dSPACE XIL API .NET on Linux.....	123
How to Register a Platform.....	125
How to Load and Start a Simulation Application.....	127
How to Execute the Python Demo Scripts.....	128
How to Build and Run the C# Demo Applications.....	131
Reference Information on XIL API .NET on Linux.....	133
CmdLoader.....	133
Limitations and Migration Aspects When Working with XIL API .NET on Linux.....	136
Limitations for Working with XIL API .NET on Linux.....	136
Aspects of Migrating From Windows to Linux.....	137
<b>Appendix</b>	<b>139</b>
Limitations and Troubleshooting.....	140
Limitations.....	140
Troubleshooting.....	145
Migrating Python Scripts from Python 3.6 to Python 3.9.....	147
Main Changes in Python 3.9.....	147
Main Changes in Handling Python 3.9 with dSPACE Software.....	148
General Information on Using Python Installations.....	149
Technical Changes.....	150
Using the Message Reader API.....	152
Introduction to the Message Reader API.....	152
Reading dSPACE Log Messages via the Message Reader API.....	152
Supported dSPACE Products and Components.....	154
Example of Reading Messages with Python.....	155
Example of Reading Messages with C#.....	157
dSPACE.Common.MessageHandler.Logging Reference.....	158
ILogMessage Interface.....	159
ILogSession Interface.....	160
MessageReader Class.....	162
MessageReaderSettings Class.....	163
Severity Enumeration.....	165

Index

167

# About This Document

---

## Content

This document introduces you to dSPACE XIL API .NET Implementation, with instructions on installing it and general information on working with it.

### Note

#### Using dSPACE XIL API .NET on Linux

This document contains information on the features provided by the Windows-based version and the Linux-based version of dSPACE XIL API .NET. For the differences of the versions, e.g., the limitations when working with the Linux-based version, refer to [Working with XIL API .NET on Linux](#) on page 121.

dSPACE XIL API .NET Implementation contains the implementation of the Framework, Testbench, MAPort for model access and the EESPort for electrical error simulation.

In the context of dSPACE products that are not based on the ASAM AE XIL API standard, the electrical error simulation is called *failure simulation*. The errors are therefore *failures* that are executed on failure simulation hardware. Usually the hardware provides a failure insertion unit (FIU). You find both terms in this document.

For complete information on the API's capabilities, you should also read the installed documentation of the ASAM AE XIL 2.1.0 standard, and look at the available demo source code.

---

## Audience profile

This document is intended for engineers who want to implement applications based on the ASAM AE XIL standard using dSPACE systems for platform access (MAPort) and electrical error simulation (EESPort).

---

## Required knowledge

It is assumed that you have experience with .NET programming. Furthermore, knowledge in handling the host PC and the Microsoft Windows operating system is a prerequisite.

## Related Documents

Below is a list of documents that you are recommended to read when working with dSPACE XIL API .NET Implementation. The documents of the ASAM AE XIL standard are not contained in the dSPACE XIL API .NET on Linux installation. You find the documentation in the dSPACE XIL API .NET on Windows installation or you can request it from dSPACE Support.

### Note

The documents of the ASAM AE XIL standard are encrypted. The documentation can be viewed only if the dSPACE XIL API .NET installation has been decrypted in the dSPACE Installation Manager. For information on decryption, refer to [How to Decrypt Encrypted Archives of dSPACE Software Installations \(Managing dSPACE Software Installations !\[\]\(6605b201d6f14d9b3bcb8ab5f274d107\_img.jpg\)](#)).

The following documents of the ASAM AE XIL standard are available:

- [ASAM\\_AE\\_XIL\\_Generic-Simulator-Interface\\_BS-1-4-Programmers-Guide\\_V2-1-0.pdf](#), which contains basic information on the ASAM AE XIL standard.
- [ASAM\\_AE\\_XIL\\_Generic-Simulator-Interface\\_BS-2-4\\_CSharp-API-Technology-Reference-Mapping-Rules\\_V2-1-0.pdf](#), which contains the mapping rules describing the transformation from the generic UML model to C#-specific interfaces.

Use the links in dSPACE Help to access the ASAM AE XIL documents.

### Note




Legal Information on ASAM binaries and ASAM documentation  
dSPACE software also installs components that are licensed and released by ASAM e.V. (Association for Standardisation of Automation and Measuring Systems).

dSPACE hereby confirms that dSPACE is a member of ASAM and as such entitled to use these licenses and to install the ASAM binaries and the ASAM documentation together with the dSPACE software.



You are not authorized to pass the ASAM binaries and the ASAM documentation to third parties without permission. For more information, visit <http://www.asam.net/license.html>.

## 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.



Symbol	Description
<b>NOTICE</b>	Indicates a hazard that, if not avoided, could result in property damage.
<b>Note</b>	Indicates important information that you should take into account to avoid malfunctions.
<b>Tip</b>	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.

# Introduction

<b>Introduction</b>	Provides general information on the ASAM AE XIL API standard and its dSPACE-specific implementation.
---------------------	------------------------------------------------------------------------------------------------------

## Where to go from here

### Information in this section

<a href="#">Contents of the Installed Software Package.....</a>	<a href="#">11</a>
Overview on the installation and required licenses.	
<a href="#">General Concepts of the ASAM AE XIL Standard.....</a>	<a href="#">13</a>
Provides general information on the ASAM AE XIL API standard.	
<a href="#">Features of the dSPACE XIL API Implementation.....</a>	<a href="#">16</a>
Provides general information on the dSPACE-specific implementation of the ASAM AE XIL API standard.	
<a href="#">Required Software and Hardware.....</a>	<a href="#">17</a>
Provides general information on the required software and hardware for model access and electrical error simulation.	
<a href="#">Protecting Long-Term Tests Against Interrupting or Blocking.....</a>	<a href="#">21</a>
Long-term tests must not be interrupted by your host PC activating a power-saving state triggered by a user action or by Microsoft Windows.	

## Contents of the Installed Software Package

<b>Introduction</b>	By installing dSPACE XIL API .NET several software components are installed.
---------------------	------------------------------------------------------------------------------

<b>Components of the dSPACE XIL API .NET installation</b>	dSPACE XIL API .NET is installed by running <code>Install_Release.exe</code> from the dSPACE DVD 2. dSPACE XIL API .NET belongs to the Test Automation APIs product set.
-----------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The components of dSPACE XIL API .NET are installed in  
 C:\Program Files\Common Files\dSPACE\XIL API .NET\  
 <ProductVersion>.

Component	Location
ASAM binaries <sup>1)</sup>	C:\Program Files (x86)\ASAM e.V\ ASAM AE XIL API Standard Assemblies 2.1.0\bin The following binaries are available: <ul style="list-style-type: none"> <li>ASAM.XIL.Implementation.Framework</li> <li>ASAM.XIL.Implementation.FrameworkFactory</li> <li>ASAM.XIL.Implementation.ManifestReader</li> <li>ASAM.XIL.Implementation.Testbench</li> <li>ASAM.XIL.Implementation.TestbenchFactory</li> <li>ASAM.XIL.Implementation.XILSupportLibrary</li> <li>ASAM.XIL.Interfaces</li> </ul>
Example scripts for Framework, MAPort and EESPort	<InstallationPath>\Demos Use the Windows Start menu and click dSPACE XIL API .NET <Version> - Copy XIL API .NET <Version> Demos to copy the demos to a folder with modify rights.
User documentation	Use the Windows Start menu to open dSPACE Help (dSPACE XIL API .NET <Version>).

<sup>1)</sup> The original ASAM setup for the XIL API binaries is automatically called when you start the dSPACE XIL API .NET installation.

To analyze the log files created by dSPACE products for error or warning messages via script, you can use the Message Reader API, which is included in the dSPACE XIL API .NET installation. Refer to [Using the Message Reader API](#) on page 152.

## dSPACE XIL API .NET Implementation license

- You need the *XIL API MAPort* license to access hardware via the XIL API MAPort.  
 The license is contained in the *Platform API Package*.
- You need the *XIL API EESPort* license to run electrical error simulation (failure simulation) via the XIL API EESPort.  
 The license is contained in the *Failure Simulation Package*.

The available licenses are considered during run time and for decrypting the ASAM user documentation.

## Related topics

### Basics

[Working with XIL API .NET on Linux.....](#) 121

# General Concepts of the ASAM AE XIL Standard

## Introduction

The dSPACE XIL API .NET implementation is based on the ASAM AE XIL 2.1.0 standard. It is the successor of the ASAM AE XIL 2.0.1 standard and comes with some enhancements and modifications.

The main approach of this standard is the decoupling of test automation software and test hardware to allow the reuse of test cases for different hardware systems.

To understand the concepts of the ASAM AE XIL standard, it is recommended to read the ASAM user documentation, that is included in the dSPACE XIL API .NET installation, before applying the dSPACE-specific API implementation.

## Framework concept

The ASAM AE XIL API provides two layers that result in a common framework for initializing and configuring the different parts of the standard.

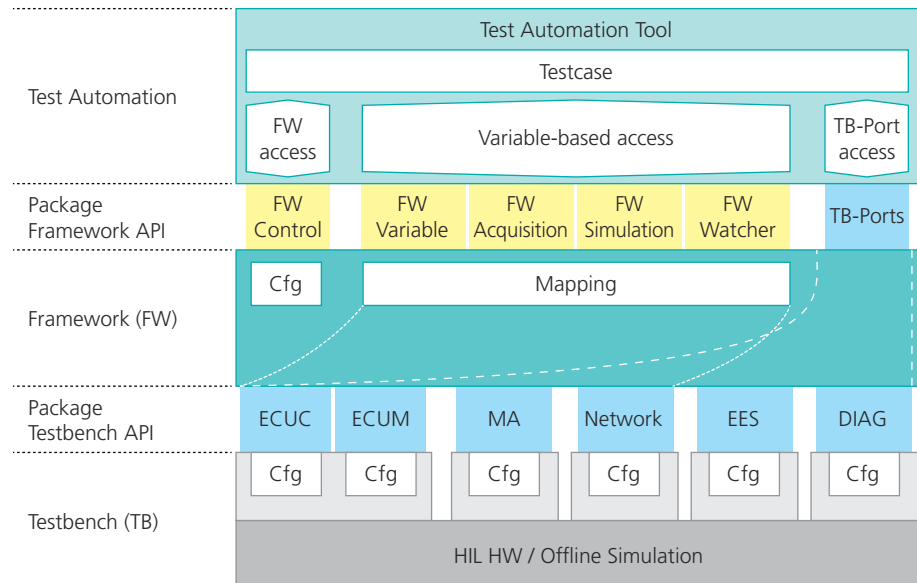
- Framework API

This API allows the exchange of the test automation software by a standardized access point for test cases, so a decoupling of test cases from real and virtual test systems takes place.

- Testbench API

This API separates the test hardware from the test automation software and allows a standardized access to the hardware.

The following figure is from the ASAM AE XIL Programmers Guide and shows the architecture.



**Port concept**

The different hardware devices you can access via the Testbench are grouped into ports.

The following ports are specified by the ASAM AE XIL standard:

- **Model Access Port (MAPort)**  
Provides access to the simulation model for reading and writing variables, and for capturing and generating signals. It allows the definition and stimulation of real-time capable signal descriptions.
- **ECU Access Port (ECUPort)**
  - The ECUMPort allows capturing and reading of measurement variables.
  - The ECUCPort allows access to ECU internal calibration values.
- **Diagnostic Port (DIAGPort)**  
Provides access to 3<sup>rd</sup> party ECU diagnostic tools.
- **Electrical Error Simulation Port (EESPort)**  
Provides access to electrical failure injection systems for electrical error simulation, for example, to simulate short circuits for ECU pins.
- **Network Access Port (NetworkPort)**  
Provides access to read and write values on the automotive network systems, such as CAN, LIN, or Flexray in a symbolic form.

**Data types**

The ASAM standard also provides standardized data types to decouple the data types used on the various hardware and software systems. The following table provides the mapping between the standardized data types defined in the ASAM UML model and the data types used with C#.

XIL API Data Types	C# Data Types
A_BOOLEAN	bool
A_INT8	sbyte
A_INT16	short
A_INT32	int
A_INT64 <sup>1)</sup>	long
A_UINT8	byte
A_UINT16	ushort
A_UINT32	uint
A_UINT64 <sup>1)</sup>	ulong
A_FLOAT32	float
A_FLOAT64	double
A_UNICODE2STRING	String
A_BYTEFIELD	IList<byte>

<sup>1)</sup> There are some limitations when you use 64-bit integer data types, refer to [Limitations](#) on page 140.

These are the basic scalar data types. Complex data types have to be implemented by using these basic data types.

#### Tip

You find further mapping information required to apply the ASAM AE XIL standard in a C# application in the *C# Technology Reference*.

## ASAM binaries and documentation

#### Note

Legal Information on ASAM binaries and ASAM documentation  
dSPACE software also installs components that are licensed and released by ASAM e.V. (Association for Standardisation of Automation and Measuring Systems).

dSPACE hereby confirms that dSPACE is a member of ASAM and as such entitled to use these licenses and to install the ASAM binaries and the ASAM documentation together with the dSPACE software.

You are not authorized to pass the ASAM binaries and the ASAM documentation to third parties without permission. For more information, visit <http://www.asam.net/license.html>.

After you have installed dSPACE XIL API .NET implementation, you can find some documents published by ASAM in dSPACE Help.

#### Note

The documents of the ASAM AE XIL standard are encrypted. The documentation can be viewed only if the dSPACE XIL API .NET installation has been decrypted in the dSPACE Installation Manager. For information on decryption, refer to [How to Decrypt Encrypted Archives of dSPACE Software Installations](#) (Managing dSPACE Software Installations ).

The following documents of the ASAM AE XIL standard are available:

- [ASAM\\_AE\\_XIL\\_Generic-Simulator-Interface\\_BS-1-4-Programmers-Guide\\_V2-1-0.pdf](#), which contains basic information on the ASAM AE XIL standard.
- [ASAM\\_AE\\_XIL\\_Generic-Simulator-Interface\\_BS-2-4\\_CSharp-API-Technology-Reference-Mapping-Rules\\_V2-1-0.pdf](#), which contains the mapping rules describing the transformation from the generic UML model to C#-specific interfaces.

Use the links in dSPACE Help to access the ASAM AE XIL documents.

## ASAM demos

The ASAM standard package also contains demos. These are not contained in the dSPACE XIL API .NET installation. However, you find demo samples developed for the dSPACE XIL API .NET implementation in `<InstallationPath>\Demos`.

**ASAM wiki**

For brief information on the supported version, refer to the Wiki site provided by ASAM, refer to <https://wiki.asam.net/display/STANDARDS/ASAM+XIL>.

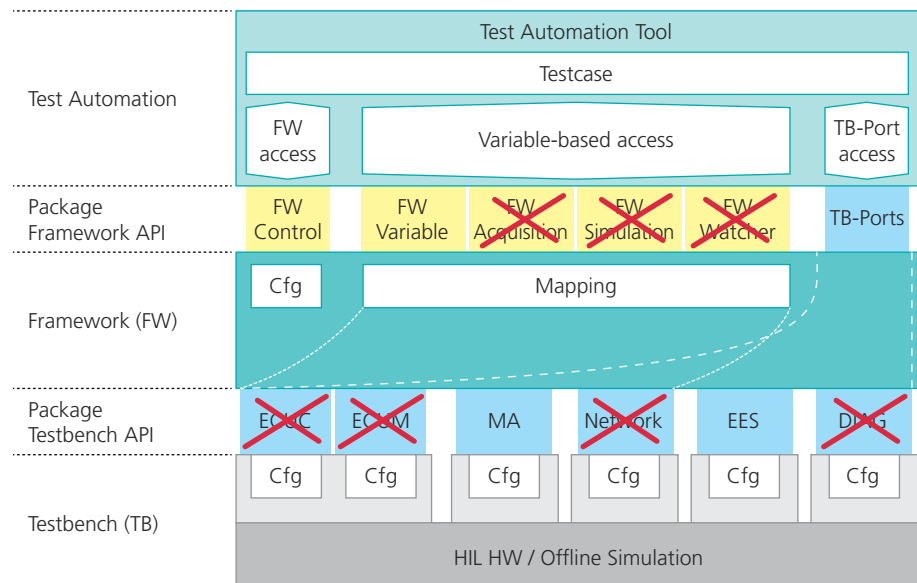
## Features of the dSPACE XIL API Implementation

**Introduction**

The dSPACE XIL API .NET implementation does not support the entire functionality of the ASAM AE XIL API. The supported features are described here.


**Supported layer**

The dSPACE XIL API .NET implementation provides the Testbench API and the Framework API restricted to the features for mapping variable names, platform access via variable objects, and unit conversion. The Mapping feature of the Framework lets you specify a mapping between an alias name used in your test environment and the concrete identifier, i.e., the model path for the model variable in your real-time application you want to access. It also supports unit conversion, i.e., the unit used in your test environment can be converted to the unit used in your real-time application. For example, you can convert from km/h to miles/h.



If you try to use methods of not supported Framework packages, a `FrameworkException` is thrown with the `eCOMMON_NOT_IMPLEMENTED` error code.



<b>Supported ports</b>	<p>The dSPACE XIL API .NET implementation supports:</p> <ul style="list-style-type: none"> <li>▪ Model Access Port (MAPort)</li> <li>▪ Electrical Error Simulation Port (EESPort)</li> </ul>
<b>Data types</b>	<p>The dSPACE XIL API .NET implementation supports any data type that is used by an MAPort or EESPort instance, except for curve and map.</p>
<b>dSPACE-specific enhancements</b>	<p>If there are enhancements or modifications to the API standard, you find them in the <a href="#">dSPACE XIL API Reference</a> .</p> <p>There you also find detailed information on configuration settings required to access dSPACE hardware.</p>
<b>Demo samples</b>	<p>The dSPACE XIL API .NET implementation provides demo samples for the MAPort in C#, Python and M and the EESPort in C# and Python. You will also find simulation applications that you can execute on dSPACE platforms when working with the MAPort.</p> <p>You can find the demo samples in <code>&lt;InstallationPath&gt;\Demos</code>.</p> <p>Use the Windows Start menu and click dSPACE XIL API .NET &lt;Version&gt; - Copy XIL API .NET &lt;Version&gt; Demos to copy the demos to a folder with modify rights.</p>

## Required Software and Hardware

<b>Introduction</b>	<p>Depending on your use case, there are some software and hardware requirements to be fulfilled.</p>
<b>Development software</b>	<p>The dSPACE XIL API .NET implementation is based on the Microsoft .NET Framework 4.8. If you run the dSPACE XIL API .NET setup, the Microsoft .NET Framework 4.8 is installed to your PC. The dSPACE XIL API .NET implementation can also be used with the earlier Microsoft .NET Framework versions as of 4.5.1.</p> <p>The dSPACE XIL API .NET implementation only supports 64-bit clients.</p> <p>The dSPACE XIL API .NET implementation only supports XIL API 2.1.0 server.</p> <p>For the installed C# demos, you find solution files (.SLN) that you can open in Microsoft Visual Studio 2019 or newer.</p>

The available demo scripts in Python can be opened via a Python editor. The demo scripts are based on Python 3.9.

The M demos can be used with MATLAB R2019a or newer.

## Additional dSPACE software products

dSPACE provides several software products to implement real-time applications and to experiment. The required tools depend on your use scenarios and on your hardware.

**Building a simulation application** The control algorithm/ECU you want to test is to be implemented beforehand.

- For dSPACE modular systems, RapidPro systems, MicroAutoBox II, and MicroLabBox, you implement a Simulink model in MATLAB using dSPACE RTI blocksets.
- For SCALEXIO systems and MicroAutoBox III, you implement the I/O model in ConfigurationDesk and the behavior model in MATLAB/Simulink.
- For offline simulation using VEOS, you implement the virtual ECU in SystemDesk or TargetLink and the environment model in MATLAB/Simulink.

**Downloading the simulation application to the hardware** The built simulation application (real-time application for dSPACE hardware or offline simulation application for VEOS) must be downloaded to the platform. This is done by using the **Configure** method of the MAPort or by using the Framework with an MAPort configured in the Framework configuration file. It is required that no other dSPACE software is accessing the currently running application. Before you can download the simulation application to a platform, you have to register the available platforms on your host PC.

- To register a platform, you can use a dSPACE software product that provides the Platform Manager and supports the required platform.

### Tip

The Platform Manager of ControlDesk supports any dSPACE platform.


**Using stimulus signals** Optionally, you can create an STZ file that provides the stimulus signals, via ControlDesk.



**Experimenting** To experiment with the simulation application interactively, you can use ControlDesk. This allows you to access variables and parameters for measuring and calibrating.

**Electrical error simulation** When you use a simulator with discrete FIU, such as a dSPACE Simulator Mid-Size based on DS2211, you do not need further dSPACE tools for configuring the EESPort and the error configurations.

When you use a simulator with integrated SCALEXIO FIU, you have to configure the failure system in ConfigurationDesk. There, you can also generate the real-time application that is to be referenced in the port configuration file. For more information, refer to [Specifying Failure Simulation in ConfigurationDesk \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(f60b7a900783ac3fd531bfd9c111be6d\_img.jpg\)](#)).

ControlDesk provides a graphical user interface for electrical error simulation. It is intended only for the manual operation of the HIL simulator's failure

simulation hardware. For working with the user interface for electrical error simulation via XIL API, the licenses in the *Failure Simulation Package* are required. The port configuration and error configuration files created in ControlDesk can be used with dSPACE XIL API .NET implementation. For more information, refer to [ControlDesk Electrical Error Simulation via XIL API EESPort](#) .

**Automating tests** As an alternative to scripting your tests with the dSPACE XIL API .NET implementation, you can use AutomationDesk, which provides the XIL API and XIL API Convenience libraries for graphically building test sequences for platform access and electrical error simulation. With AutomationDesk's Project Manager, you can manage test projects with all its relevant data, results and reports. For more information, refer to [Accessing Simulation Platforms via the XIL API Convenience Library](#) (AutomationDesk [Accessing Simulation Platforms](#) ) and [Basics and Instructions](#) (AutomationDesk [Simulating Electrical Errors](#) .

#### Minimum software requirements

As a minimum requirement to manage the dSPACE platforms, you can use the dSPACE Platform Management API that is contained in your dSPACE XIL API .NET installation. If you want to work with a graphical user interface, use the license-free version of ControlDesk.

#### Required hardware

The dSPACE MAPort can be used to access dSPACE platforms. Examples of configuration files are contained in the dSPACE XIL API .NET installation for physical hardware and offline simulation with VEOS.

The dSPACE EESPort requires dSPACE electrical error simulation hardware. For electrical error simulation with discrete FIU, additional dSPACE FIU boards must be connected to your simulator hardware. SCALEXIO FIU hardware is on-board electrical error simulation hardware that you configure by using ConfigurationDesk. For more information, refer to [Implementing an EESPort Client Application](#) on page 53.

For connecting the discrete FIU, there might be further components required according to the connection type used, i.e. RS232 or CAN interface.

#### Note

##### Reduced Performance by Using External RS232 Converters

You are strongly recommended to use a physical RS232 port of the host PC to control the failure simulation hardware. If external RS232 ports are missing, try to use an internal RS232 port of the host PC's motherboard. Software triggers and dynamic errors are not supported if you use an external RS232 converter. Communication via an external RS232 converter is also time-critical and can cause communication errors.

If there is no alternative to using an external RS232 converter:

- Use the IOLAN DS1 from Perle as an Ethernet-to-RS232 converter. For configuring this tool, refer to <http://www.dspace.com/go/eth2rs232>.
- Otherwise, use a USB-to-RS232 converter with an FTDI chipset and the newest FTDI driver, refer to <http://www.ftdichip.com/FTDrivers.htm>.

### Firmware compatibility guidelines

### Firmware and real-time application

Firmware is downward-compatible.

This means:

- You can use the firmware from a dSPACE Release in the following cases:
  - The *firmware is of the same dSPACE Release* with which the real-time application was built.
  - The *firmware is of a newer dSPACE Release* than the dSPACE Release with which the real-time application was built.
- You cannot use the firmware from a dSPACE Release if the *firmware is of an older dSPACE Release* than the dSPACE Release with which the real-time application was built.

### Hardware dependency of the required firmware version

- If you work with DS1007, DS1202 MicroLabBox, MicroAutoBox III, or SCALEXIO, use the firmware version that matches the dSPACE Release you are working with.

Host PC		Compatible Firmware Version				
dSPACE Release	Real-Time Testing Version	SCALEXIO	MicroAutoBox III	DS1202 MicroLabBox	DS1007	VEOS
RLS2021-A	5.0	5.1 5.0	5.1	2.16	3.16	5.2
RLS2020-B	4.4	5.0	5.0	2.14	3.14	5.1
RLS2020-A	4.3	4.6	4.6	2.12	3.12	5.0
RLS2019-B	4.2	4.5	4.5	2.10	3.10	4.5
RLS2019-A	4.1	4.4	–	2.8	3.8	4.4
RLS2018-B	4.0	4.3	–	2.6	3.6	4.3
RLS2018-A	3.4	4.2	–	2.4	3.4	4.2
RLS2017-B	3.3	4.1	–	2.2	3.2	4.1
RLS2017-A	3.2	4.0	–	2.0	3.0	4.0
RLS2016-B	3.1	3.5	–	1.7	2.6	3.7
RLS2016-A	3.0	3.4	–	1.5	2.4	3.6
RLS2015-B	2.6	3.3	–	1.3	2.2	3.5
RLS2015-A	2.5	3.2	–	–	2.0	3.4
RLS2014-B	2.4	3.1	–	–	–	3.3
RLS2014-A	2.3	3.0	–	–	–	3.2
RLS2013-B	2.2	2.3	–	–	–	3.1

- If you work with any other dSPACE real-time hardware, use the newest firmware version available.

For up-to-date information on firmware updates, refer to <http://www.dspace.com/go/firmware>.

## Protecting Long-Term Tests Against Interrupting or Blocking

---

**Avoiding power-saving states** Long-term tests must not be interrupted by a host PC activating a power-saving state triggered by a user action or by Microsoft Windows. It is recommended to disable all power-saving states, for example, the sleep or the hibernation state during long-term tests.



# Using dSPACE XIL API .NET Implementation

## Where to go from here

## Information in this section

<a href="#">Basics on Creating XIL API Client Applications.....</a>	<a href="#">23</a>
Provides information on assembly references, exception handling, error handling, and properties.	
<a href="#">How to Create XIL API C# Projects.....</a>	<a href="#">25</a>
Gives you instructions for creating XIL API C# projects.	
<a href="#">Using XIL API with Python.....</a>	<a href="#">26</a>
Provides information on using XIL API with Python.	
<a href="#">Using XIL API with MATLAB.....</a>	<a href="#">28</a>
Provides information on using XIL API with M scripts.	

## Basics on Creating XIL API Client Applications

### Introduction

There is some basic information relevant for any XIL API client application you want to create.

### Assembly references

You can use the XIL API functionality in a client application only if it references the following assemblies:

- To work with the XIL API Framework:  
ASAM.XIL.Implementation.FrameworkFactory, Version 2.1.0.0  
The Public Key Token is **d64b0277da8a2c74**.
- To work with the XIL API Testbench:  
ASAM.XIL.Implementation.TestbenchFactory, Version 2.1.0.0  
The Public Key Token is **fc9d65855b27d387**.
- In both cases you have to reference the ASAM XIL interfaces assembly:

ASAM.XIL.Interfaces, Version 2.1.0.0

The Public Key Token is **bf471dff114ae984**.

---

### Exception handling

It is recommended to use **try-finally** constructs in your application to handle exceptions. This allows you to terminate your XIL API client application in a predefined manner.

- Internally, the resources of your script should have a definite location to be freed.
- Externally, the connected hardware should be driven in a secure state before the application is quit by an exception.

#### CAUTION

To avoid personal injury or hardware damage by unexpected behavior of connected external devices, you must use the **Dispose** method of **MAPort**, **EESPort** and **Capture** objects or the **Shutdown** method of the **Framework** in the **finally** path to free the instantiated resources.

---

### Error handling

It is also recommended to use **try-finally** constructs in your application to handle error messages via the **TestbenchPortException**. This allows you to get detailed information on problems occurred.

The **TestbenchPortException** and the **FrameworkException** have the following properties:

- **Code**  
Unique error code defined by ASAM
- **CodeDescription**  
Textual description of the error code with basic information
- **VendorCode**  
Specific error code defined by the manufacturer
- **VendorCodeDescription**  
Description for the manufacturer-specific vendor code with detailed information

You can find examples of implementing error handling in the demo samples.

---

### Code commonly used by any port

The XIL API provides either the **Framework** or the **Testbench** that are used to implement the basic functionalities for configuration and instantiating the various ports. There is therefore a common code section for the **MAPort** and the **EESPort**. For more information, refer to [Using the Framework](#) on page 31 or [Using the Testbench](#) on page 35.

---

### Public and private properties

If you use an IDE for implementing your application, most editors display the public and private properties available for the currently selected object or class.



Use only public properties that are documented in the ASAM AE XIL documentation. Private properties might be changed.

## How to Create XIL API C# Projects

<b>Objective</b>	There are some common instructions to be mentioned for creating an XIL API project in C#.
<b>Using demo projects</b>	If you want to use a dSPACE XIL API demo project, you have to prepare and compile it in the way described below. Beforehand, you have to copy the complete demo folder to a working folder, where you have write access.
<b>Preconditions</b>	<p>The following software must be installed on your PC:</p> <ul style="list-style-type: none"> <li>▪ Microsoft Visual Studio 2019 or later</li> <li>▪ .NET Framework 4.8, including MSBuild 16.0 (dSPACE XIL API .NET also supports .NET Framework versions as of 4.5.1)</li> <li>▪ dSPACE XIL API .NET</li> </ul>
<b>Method</b>	<p><b>To create an XIL API C# project</b></p> <ol style="list-style-type: none"> <li>1 Create a new Visual Studio 2019 project by selecting Visual C# – Windows – Console Application.</li> </ol> <div style="background-color: #f0f0f0; padding: 10px; margin: 10px 0;"> <p><b>Note</b></p> <p>Select .NET Framework 4.8 on top of the New Project dialog.</p> </div> <ol style="list-style-type: none"> <li>2 In the Solution Explorer, click Add Reference from the References entry's context menu. The Add Reference dialog opens.</li> <li>3 Choose the Browse page and add the required DLL files to your project. <ul style="list-style-type: none"> <li>▪ ASAM.XIL.Interfaces</li> <li>▪ ASAM.XIL.Implementation.FrameworkFactory or ASAM.XIL.Implementation.TestbenchFactory</li> </ul> </li> <li>4 Use the Program.cs file to implement your C# client application.</li> <li>5 Select x64 as the platform target in the build configuration of your application to create a 64-bit client application.</li> <li>6 Start the build process to compile and link your client application.</li> </ol>

**Result**

You have prepared a Visual Studio project and built an executable XIL API client application.

**Note**

- The XIL API client application can only be executed on a PC with the same version of dSPACE XIL API .NET installed.
- To avoid resource leaks of the services running in your simulation application or blocked access to the simulation platform, do not abort the application before the **Dispose** methods of MAPort, EESPort and Capture instances have been executed. If you are using the Framework, do not abort the application before the **Shutdown** method of the Framework has been executed. This is also relevant when you run the XIL API client application in debug mode.

## Using XIL API with Python

**Introduction**

You can use the dSPACE XIL API .NET implementation also with Python scripts based on Python 3.9.

The combination of Python and C#/.NET is realized by using *Python for .NET*, which is included in the Python distribution by dSPACE. For more information on this module, refer to <https://pythonnet.github.io/>.

For information on migrating from Python 3.6 to Python 3.9, refer to [Migrating Python Scripts from Python 3.6 to Python 3.9](#) on page 147.

**Using Python**

The following script examples show you the steps for integrating the XIL API features to a Python script.

**Importing required modules to the interpreter namespace** The following program listing shows how you can import required modules to the interpreter namespace.

```
#-----
# Import python library modules to be used.
# The os module is used for path concatenations.
# The sys module is used to get the command line arguments.
#-----
import os
import sys
```

**Importing .NET assemblies** To get access to the XIL API object model, you have to import the .NET assemblies.

```
# Import .NET assemblies (using Python for .NET)
import clr
clr.AddReference("System")
import System
# Load ASAM assemblies from the global assembly cache (GAC)
assemblyString = "ASAM.XIL.Interfaces, Version=2.1.0.0, Culture=neutral, PublicKeyToken=bf471dff114ae984"
clr.AddReference(assemblyString)
assemblyString = "ASAM.XIL.Implementation.TestbenchFactory, Version=2.1.0.0, Culture=neutral,
PublicKeyToken=fc9d65855b27d387"
clr.AddReference(assemblyString)
# To work with the Framework you have to load the following ASAM assembly
# assemblyString = "ASAM.XIL.Implementation.FrameworkFactory, Version=2.1.0.0, Culture=neutral,
PublicKeyToken=d64b0277da8a2c74"
# clr.addReference(assemblyString)
```

After these assemblies have been loaded, namespaces and types of the XIL API can be imported. Usually only a few types need to be imported explicitly because most of the XIL API objects are created via factory methods and not via constructors of the implementing classes.

```
# Import the TestbenchFactory class from the .NET assemblies
from ASAM.XIL.Implementation.TestbenchFactory.Testbench import TestbenchFactory
from ASAM.XIL.Interfaces.Testbench.Common.Error import TestbenchPortException
# To work with the Framework you have to import the following classes
# from ASAM.XIL.Implementation.FrameworkFactory.Framework import FrameworkFactory
# from ASAM.XIL.Interfaces.Framework.Error import FrameworkException
```

When you use the MAPort, you have to additionally import the following XIL API .NET classes.

```
from ASAM.XIL.Interfaces.Testbench.Common.Capturing.Enum import DurationUnit, CaptureState
```

When you use the EESPort, you have to additionally import the following XIL API .NET classes.

```
from ASAM.XIL.Interfaces.Testbench.EESPort.Error import EESPortException
from ASAM.XIL.Interfaces.Testbench.EESPort.Enum import EESPortState, ErrorCategory, ErrorType, LoadType, TriggerType
```

**Creating a Testbench** The following script shows you how to instantiate a Testbench and an MAPort. The procedure for the other XIL API classes is similar.

```
...
def ExecuteDemo():
    # Enclose XIL API class objects with try:-finally: block, to ensure that
    # these objects will be deleted at the end of the function.
    try:
        TBFactory = TestbenchFactory()
        TB = TBFactory.CreateVendorSpecificTestbench("dSPACE GmbH", "XIL API", "2021-A")
    ...
    # Create MAPort
    DemoMAPort = TB.MAPortFactory.CreateMAPort("DemoMAPort")
    MAPortConfigFile = r"<InstallationPath>\Demos\MAPort\Common\PortConfigurations\MAPortConfigSCALEXIO.xml"
    MAPortConfig = DemoMAPort.LoadConfiguration(MAPortConfigFile)
    DemoMAPort.Configure(MAPortConfig, False)
    ...
```

The exceptions thrown by the Testbench can be caught and re-raised.

```
except TestbenchPortException as ex:
    print ("A TestbenchPortException occurred:")
    print ("CodeDescription: %s" % ex.CodeDescription)
    print ("VendorCodeDescription: %s" % ex.VendorCodeDescription)
    raise
finally:
    ...
```

#### Note

The imported XIL API types can be used in Python except for collection classes, such as lists, tuples or dictionaries. These must be manually converted. The following example shows the conversion of a Python list into a .NET array.

```
DemoVariablesList = [BatteryVoltage, TurnSignalLever]
DemoVariablesList.extend([TurnSignalFrontLeft,
TurnSignalFrontRight])
VariableArray = Array[str](DemoVariablesList)
...
```

```
DemoCapture = DemoMAPort.CreateCapture(masterTask)
DemoCapture.Variables = VariableArray
```

A conversion is also required to use, for example, a .NET list as a Python list.

```
DemoPyCaptureVariables = list(DemoCapture.Variables)
```

#### Main application

The main application basically consists of the specified function call only.

```
if __name__ == "__main__":
    ExecuteDemo()
```

## Using XIL API with MATLAB

#### Introduction

You can use the dSPACE XIL API .NET implementation also with M scripts in MATLAB.

#### Using MATLAB

The following script examples show you the steps for integrating the XIL API features to an M script.

To get access to the XIL API object model, you have to load the ASAM assemblies for the Framework or the Testbench.

After these assemblies have been loaded, namespaces and types of the XIL API can be imported. Usually only a few types need to be imported explicitly because most of the XIL API objects are created via factory methods and not via constructors of the implementing classes.

## Working with the Framework

```
NET.addAssembly('ASAM.XIL.Interfaces, Version=2.1.0.0, Culture=neutral, PublicKeyToken=bf471dff114ae984');
NET.addAssembly('ASAM.XIL.Implementation.FrameworkFactory, Version=2.1.0.0, Culture=neutral,
PublicKeyToken=d64b0277da8a2c74');
% Import the XIL API classes from the .NET assemblies
import ASAM.XIL.Implementation.FrameworkFactory.Framework.*;
```

The following script shows you how to instantiate a Framework with an MAPort contained in the framework configuration. The procedure for the other XIL API classes is similar.

```
...
% Enclose XIL API class objects with try:-catch: block, to ensure that
% these objects will be deleted at the end of the function.
try:
    FWFactory = FrameworkFactory();
    FW = FWFactory.CreateVendorSpecificFramework('dSPACE GmbH', '2021-A');
    FWConfigFile = r"<InstallationPath>\Demos\MAPort\Common\FrameworkConfigurations\FrameworkConfigVEOS.xml";
    FWConfig = FW.LoadConfiguration(FWConfigFile);
    FW.Init(FWConfig);
...
% Reading and writing variables directly or via the MAPort
...
```

The exceptions thrown by the Framework can be caught and re-raised.

```
catch e
    if (exist('FW', 'var'))
        FW.Shutdown(false);
    end
    if (isa(e, 'NET.NetException'))
        if (isa(e.ExceptionObject, 'ASAM.XIL.Interfaces.Framework.Common.Error.FrameworkException'))
            %-----
            % Display the XIL API exception information to get the cause of an error
            %-----
            fprintf('XIL API exception occurred:\n');
            fprintf('Code: %d\n', e.ExceptionObject.Code);
            fprintf('CodeDescription: %s\n', char(e.ExceptionObject.CodeDescription));
            fprintf('VendorCode: %d\n', e.ExceptionObject.VendorCode);
            fprintf('VendorCodeDescription: %s\n', char(e.ExceptionObject.VendorCodeDescription));
        else
            fprintf('.NET exception occurred\n');
            fprintf('Message: %s\n', char(e.message));
        end
    end
    rethrow(e);
end
```

## Working with the Testbench

```
NET.addAssembly('ASAM.XIL.Interfaces, Version=2.1.0.0, Culture=neutral, PublicKeyToken=bf471dff114ae984');
NET.addAssembly('ASAM.XIL.Implementation.TestbenchFactory, Version=2.1.0.0, Culture=neutral,
PublicKeyToken=fc9d65855b27d387');
% Import the XIL API classes from the .NET assemblies
import ASAM.XIL.Implementation.TestbenchFactory.Testbench.*;
import ASAM.XIL.Interfaces.Testbench.*;
import ASAM.XIL.Interfaces.Testbench.Common.Error.*;
% Additional classes for the MAPort
import ASAM.XIL.Interfaces.Testbench.Common.CaptureResult.*;
import ASAM.XIL.Interfaces.Testbench.Common.Capturing.*;
import ASAM.XIL.Interfaces.Testbench.Common.ValueContainer.*;
import ASAM.XIL.Interfaces.Testbench.MAPort.*;
```

The following script shows you how to instantiate a Testbench and an MAPort. The procedure for the other XIL API classes is similar.

```
...
% Enclose XIL API class objects with try:-finally: block, to ensure that
% these objects will be deleted at the end of the function.
try:
    TBFactory = TestbenchFactory();
    TB = TBFactory.CreateVendorSpecificTestbench('dSPACE GmbH', 'XIL API', '2021-A');
    MAPortFactory = TB.MAPortFactory;
    % Create MAPort
    DemoMAPort = MAPortFactory.CreateMAPort('DemoMAPort');
    MAPortConfigFile = r"<InstallationPath>\Demos\MAPort\Common\PortConfigurations\MAPortConfigSCALEXIO.xml";
    MAPortConfig = DemoMAPort.LoadConfiguration(MAPortConfigFile);
    DemoMAPort.Configure(MAPortConfig, False);
...
    % Reading and writing variables via the MAPort
...
    DemoMAPort.Dispose();
```

The exceptions thrown by the Testbench can be caught and re-raised.

```
catch e
    if (exist('demoMAPort', 'var'))
        demoMAPort.Dispose();
    end
    if (isa(e, 'NET.NetException'))
        if (isa(e.ExceptionObject, 'ASAM.XIL.Interfaces.Testbench.Common.Error.TestbenchPortException'))
            %-----
            % Display the XIL API exception information to get the cause of an error
            %-----
            fprintf('XIL API exception occurred:\n');
            fprintf('Code: %d\n', e.ExceptionObject.Code);
            fprintf('CodeDescription: %s\n', char(e.ExceptionObject.CodeDescription));
            fprintf('VendorCode: %d\n', e.ExceptionObject.VendorCode);
            fprintf('VendorCodeDescription: %s\n', char(e.ExceptionObject.VendorCodeDescription));
        else
            fprintf('.NET exception occurred\n');
            fprintf('Message: %s\n', char(e.message));
        end
    end
    rethrow(e);
end
```

# Using the Framework

## Basics on the Framework

### Introduction

The Framework is the central point in your script for instantiating ports and managing variables.

### Framework factory

The Framework factory allows you to implement test code that is independent of any vendor-specific implementation details. The vendor-specific adaptations are to be made centrally in the configuration of the Framework. With the mapping feature, you can specify alias names for variables from your real-time application, which you want to access in your test code. If you replace the real-time application you have to modify the model paths only in the framework configuration instead of editing your source code. To adapt the units used in the real-time application and in your test application, you can specify unit conversions in the framework configuration.

### Framework configuration

The Framework configuration contains the following elements:

- MappingFileList  
You can add references to mapping files that provide, e.g., variable mappings or unit conversions.
- PortDefinitionList  
You can add references to port configuration files and specify their order of initializing and terminating and the initial states of the ports. In the port definition, you also specify the vendor name, the product name and the product version.

For more information, refer to [Framework Configuration \(dSPACE XIL API Reference !\[\]\(d0262bbe9d2356661a2e89321dfcc781\_img.jpg\)](#)).

For limitations, refer to [Limitations](#) on page 140.

---

## Framework features

The Framework provides access to the ports and the functionality grouped in namespaces, that are common for all ports.

The Framework provides the following namespaces:

- **Configuration**  
Contains classes which are used for accessing the parameters of a Framework configuration, such as the list of mapping files, or the port definitions.
- **MappingInfoAPI**  
Contains classes which are used to access the information from the mapping files specified in the Framework configuration.
- **Variables**  
Contains classes which represent variables on the simulation platform and container classes (so called *Quantities*) for reading from and writing to these variables.
- **Error**  
Contains the Framework-related exception classes and error codes used to signal errors.

---

## Programming instructions

If an error occurs when calling any of the XIL API functions, an exception is thrown. To properly handle any exceptions and to see the enclosed error messages, a try-catch-finally construct should be used.



You make the Framework available in your application by adding the following code to it.

```
using ASAM.XIL.Interfaces.Framework;
using ASAM.XIL.Implementation.FrameworkFactory.Framework;
using ASAM.XIL.Interfaces.Framework.Common.Error;

class Port_Basics
{
    public void Execute()
    {
        ...
        try
        {
            IFrameworkFactory FWFactory = new FrameworkFactory();
            IFramework FW = FWFactory.CreateVendorSpecificFramework("dSPACE GmbH", "2021-A");
            ...
        }
        catch (FrameworkException ex)
        {
            Console.WriteLine("An XIL API FrameworkException occurred:");
            Console.WriteLine("Error description: {0}", ex.CodeDescription);
            Console.WriteLine("Detailed error description: {0}", ex.VendorCodeDescription);
        }
        catch (System.Exception ex)
        {
            Console.WriteLine("A system exception occurred:");
            Console.WriteLine("Error description: {0}", ex.Message);
        }
        finally
        {
            // Shutdown the Framework
            FW.Shutdown()
            ...
        }
    }
}
```

The Framework instance provides the following factory class.

Factory Class	Namespace
QuantityFactory	ASAM.XIL.Interfaces.Framework.Variables.Quantities

## Demos

You find examples for Framework configuration files and source code for several use cases in <InstallationPath>\Demos\Framework.



# Using the Testbench

## Basics on the Testbench

---

### Introduction

The Testbench is the central point in your script for instantiating ports.

---

### Testbench factory

The Testbench factory supports the approach to make most of your XIL API application regardless of the vendor-specific implementations. The vendor-specific adaptations are to be made centrally in the configuration of the Testbench.

In contrast to the Framework, the Testbench does not provide the variable management with the mapping and unit conversion features. The behavior of the ports, such as the order of initialization, must be explicitly specified in the code. Thus, using the Testbench is less flexible than using the Framework.

---

### Testbench features

The Testbench provides access to the ports and the functionality grouped in namespaces, that are common for all ports.

The Testbench provides the following namespaces:

- ValueContainer  
Contains classes which are designed to store values of different types, for example, scalar, matrix or map values.
- Error  
Contains common classes used for error handling and the error codes.
- DocumentHandling  
Contains classes which are used to read from the file system or write to the file system in different file formats.
- WatcherHandling  
Contains classes for defining trigger conditions used for capturing.
- Capturing and CaptureResult  
Contains classes for capturing and handling the result of captures.

- **Signal and SignalGenerator**  
Contains classes to describe signals for general use and especially for stimulation use cases.

### Programming instructions

The Testbench itself provides no functionality related to the device access. You always need a port instantiation to access a device with your XIL API application.

If an error occurs when calling any of the XIL API functions, an exception is thrown. To properly handle any exceptions and to see the enclosed error messages, a try-catch-finally construct should be used.

You make the Testbench available in your application by adding the following code to it.

```
using ASAM.XIL.Interfaces.Testbench;
using ASAM.XIL.Implementation.TestbenchFactory.Testbench;
using ASAM.XIL.Interfaces.Testbench.Common.Error;

class Port_Basics
{
    public void Execute()
    {
        ...
        try
        {
            ITestbenchFactory TBFactory = new TestbenchFactory();
            ITestbench TB = TBFactory.CreateVendorSpecificTestbench("dSPACE GmbH", "XIL API", "2021-A");
            ...
        }
        catch (TestbenchPortException ex)
        {
            Console.WriteLine("An XIL API TestbenchPortException occurred:");
            Console.WriteLine("Error description: {0}", ex.CodeDescription);
            Console.WriteLine("Detailed error description: {0}", ex.VendorCodeDescription);
        }
        catch (System.Exception ex)
        {
            Console.WriteLine("A system exception occurred:");
            Console.WriteLine("Error description: {0}", ex.Message);
        }
        finally
        {
            // Dispose any instances of MAPort, Capture, and EESPort
            ...
        }
    }
}
```

The Testbench instance must be used to create instances of the MAPort and EESPort classes, but also for classes in the Common namespace.

Factory Class	Namespace
MAPortFactory	ASAM.XIL.Interfaces.Testbench.MAPort
EESPortFactory	ASAM.XIL.Interfaces.Testbench.EESPort
CapturingFactory	ASAM.XIL.Interfaces.Testbench.Common.Capturing
SignalFactory	ASAM.XIL.Interfaces.Testbench.Common.Signal

Factory Class	Namespace
SignalGeneratorFactory	ASAM.XIL.Interfaces.Testbench.Common.SignalGenerator
SymbolFactory	ASAM.XIL.Interfaces.Testbench.Common.Symbol
ValueFactory	ASAM.XIL.Interfaces.Testbench.Common.ValueContainer
WatcherFactory	ASAM.XIL.Interfaces.Testbench.Common.WatcherHandling



# Implementing an MAPort Application

**Introduction** The Model Access Port (MAPort) provides access to a simulation platform for reading and writing variables and for generating stimulus signals.

**Where to go from here**

**Information in this section**

<a href="#">Basics on the MAPort.....</a>	<a href="#">39</a>
The Model Access Port requires XIL API namespaces and a configuration file to access a supported simulation platform.	
<a href="#">Implementing an MAPort Client Application.....</a>	<a href="#">44</a>
Provides information on implementing model access.	
<a href="#">Demo Projects for MAPort Client Applications.....</a>	<a href="#">49</a>
The XIL API .NET installation contains simulation applications and demo projects for various model access use cases.	

**Information in other sections**

<a href="#">Working with XIL API .NET on Linux.....</a>	<a href="#">121</a>
---------------------------------------------------------	---------------------

## Basics on the MAPort

**Introduction** The Model Access Port requires XIL API namespaces and a configuration file to access a supported simulation platform.

**Preparing the model access** You can access the simulation platform only, if the simulation platform is already registered. Functions for registering the platform are not contained in the MAPort implementation.

The MAPort package usually requires at least the following namespaces:

- ASAM.XIL.Interfaces.Common.Capturing
- ASAM.XIL.Interfaces.Common.SignalGenerator
- ASAM.XIL.Interfaces.Common.CaptureResult
- ASAM.XIL.Interfaces.Testbench.Common.ValueContainer
- ASAM.XIL.Interfaces.Testbench.Common.Error

## Configuring the MAPort

Before you can instantiate an MAPort object, you have to create a Framework or Testbench object that provides the required vendor-specific information. For more information, refer to [Using the Framework](#) on page 31 or [Basics on the Testbench](#) on page 35.

The Testbench instance allows you to create an MAPort factory for creating and handling MAPort objects. An MAPort object instantiated by the Framework or a Testbench gets the basic information on the simulation platform and simulation application to be used via a configuration XML file.

For more information on the elements and attributes of an MAPort configuration file, refer to [MAPort Configuration](#) (dSPACE XIL API Reference [\[1\]](#)).

**Example of an MAPort configuration file** With the following configuration file, you can access the real-time application that is specified in the variable description file (SDF file) running on a modular system based on DS1007.

In the **PlatformName** key, you have to enter the platform name that you specified during registration of the platform.

```
<?xml version="1.0" encoding="utf-8"?>
<PortConfigurations xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <MAPortConfig>
    <SystemDescriptionFile>D:\Work\RTApplications\throttlecontroldemo.sdf</SystemDescriptionFile>
    <PlatformName>DS1007</PlatformName>
  </MAPortConfig>
</PortConfigurations>
```

**Supported dSPACE platforms** The following dSPACE platforms are supported by the dSPACE XIL API .NET implementation.

For information on the dSPACE platforms supported by XIL API .NET on Linux, refer to [How to Register a Platform](#) on page 125.

dSPACE Platform	Default Platform Name
Modular system based on DS1006	DS1006
Modular system based on DS1007	DS1007
MicroAutoBox <sup>1)</sup>	DS1401
MicroAutoBox III	DS1403
MicroLabBox	DS1202
DS1104 R&D Controller Board <sup>2)</sup>	DS1104
SCALEXIO system with SCALEXIO Processing Unit or DS6001 Processor Board (real-time applications based on QNX and RTLinux)	SCALEXIO



dSPACE Platform	Default Platform Name
VEOS (for offline simulation)	VEOS
Multiprocessor system based on DS1006	Multiprocessor

<sup>1)</sup> The MicroAutoBox platform type supports the MicroAutoBox II.

<sup>2)</sup> When using this platform, you have to note a limitation, refer to [Limitations](#) on page 140.

By executing the **LoadConfiguration** method, you create a configuration object, that is then used by the MAPort object's **Configure** method.

By executing the **Configure** method, the real-time application is loaded to the registered platform. An already loaded application will only be overwritten, if the **forceConfig** parameter is set to **True**. The port state then changes to **eSIMULATION\_STOPPED**.

#### Note

If you use a multiprocessor system based on DS1006, and the multiprocessor real-time application was loaded before the XIL API server was started, the real-time application is reloaded independently of the **forceConfig** parameter. The running real-time application is then stopped.

### Reading and writing variables

You can use the **Read** and **Write** method to read a variable from the simulator platform or to write a modified variable to it. These are methods to access a single variable at a single point of time.

You can get the names of the variables that are available in the variable description file by using the **VariableNames** property.

With the **CheckVariableNames** method, which is introduced with XIL API 2.1.0, you can check the existence of the variables by their variable paths. The method returns a list of variable paths that are not found in the current port configuration.

### Supported variable path formats

By default, dSPACE XIL API MAPort uses the ControlDesk format for variable paths. The ControlDesk 3.x format is still supported. The table shows some examples.

Type	ControlDesk 3.x Format of Variable Paths	ControlDesk Format of Variable Paths
<b>Using Single-Processor Models</b>		
Scalar	'Model Root/double/Subsystem/Gain1/Out1'	'Platform()://Model Root/double/Subsystem/Gain1/Out1' <sup>1)</sup>
Vector	'Model Root/double/Subsystem/Gain1x3/Out1[1,3]'	'Platform()://Model Root/double/Subsystem/Gain1x3/Out1[2]' <sup>2)</sup>
Matrix	'Model Root/double/Subsystem/Gain2x3/Value[2,3]'	'Platform()://Model Root/double/Subsystem/Gain2x3/Value[1][2]'
Structure <sup>3)</sup>	—	'Platform()://Model Root/Structures/Struct.SubStruct.DoubleArray[0]'

Type	ControlDesk 3.x Format of Variable Paths	ControlDesk Format of Variable Paths
<b>Using Multiprocessor Models</b>		
Scalar	'MP_SubApplicationName/Model Root/double/Subsystem/Gain1/Out1'	'Platform():/MP_SubApplicationName/Model Root/double/Subsystem/Gain1/Out1' <sup>1), 4)</sup>
Vector	'MP_SubApplicationName/Model Root/double/Subsystem/Gain1x3/Out1[1,3]'	'Platform():/MP_SubApplicationName/Model Root/double/Subsystem/Gain1x3/Out1[2]' <sup>2)</sup>
Matrix	'MP_SubApplicationName/Model Root/double/Subsystem/Gain2x3/Value[2,3]'	'Platform():/MP_SubApplicationName/Model Root/double/Subsystem/Gain2x3/Value[1][2]'
Structure <sup>3)</sup>	–	'Platform():/MP_SubApplicationName/Model Root/Structures/Struct.SubStruct.DoubleArray[0]'

<sup>1)</sup> 'Platform' is replaced by the relevant platform name. A model path is then, e.g., 'VEOS():/Model  
Root/double/Subsystems/Gain1/Out1'. The platform name has no effect on the model path itself.

<sup>2)</sup> In ControlDesk format, an array index starts with 0, in ControlDesk 3.x format, it starts with 1.

<sup>3)</sup> Introduced with dSPACE Release 2015-B.

<sup>4)</sup> 'MP\_SubApplicationName' is replaced by the name of the relevant subapplication. A model path is then, e.g.,  
'ds1006():/slave\_appl/Model Root/double/Subsystems/Gain1/Out1'.

## Using MP systems

If you want to create a SignalGenerator for an MP system, you have to specify the application component to which the stimulus sequence will be downloaded. For this, you have to add the *ApplicationProcessor* key to the *CustomProperties* property of the SignalGenerator. For details, refer to the XIL API stimulus examples.

When you use an MP system, the *TaskName* parameter has to be set in each relevant method as the combination of MemberApplication and TaskName. For example, if the MemberApplication is called **masterApp1** and you want to access the host service, you have to specify **masterApp1/HostService** as the task name.

## Capturing variables

You can use the capturing feature to continuously measure time-dependent signals. Because this feature is also used by the ECUMPort, it is contained in the **Common.Capturing** namespace of the Testbench.

After you have created a Capture object, you can configure its behavior. The duration of a capture can be specified by time or by trigger conditions for starting and stopping a capture. Triggers are based on XIL API watcher objects.

The data to be captured must be related to a task in your simulation application. You can get the tasks that are available in the variable description file, by using the MAPort's *TaskNames* property.

For storing the results of a capture, you get a CaptureResult object. With a CaptureResultWriter object you can save the results to a file, for example, in MDF format.

**Note**

If you are running a multicore or multiprocessor system, and you capture only one subapplication, the contents of the resulting MDF file (MF4) match the contents of a single-processor application. The **SignalGroupName** then only contains the task name and not additionally the name of the subapplication.

You can use the **SignalGroupNames** property of a **CaptureResult** object to dynamically get the value of the **SignalGroupName**.

**Generating signals**

To stimulate model variables or to provide a reference signal, you can create signals in a **SignalDescription** object. A **SignalDescriptionSet** consists of one or more **SignalDescription** objects. With the **SignalGenerator** object, you can apply a signal description to a model variable.

A signal is described by segments of a specific type and duration.



Segment Type	Meaning
ConstSegment	Provides a constant value during the segment duration.
RampSegment	Provides a signal following a linear equation.
RampSlopeSegment	Provides a signal following a linear equation.
SineSegment	Provides a signal with a periodical sine waveform.
PulseSegment	Provides a signal with a periodical rectangle waveform.
SawSegment	Provides a signal with a periodical saw tooth waveform.
NoiseSegment	Provides a signal with gaussian noise.
ExpSegment	Provides a signal following an exponential curve.
IdleSegment	Provides an idle signal allowing the model to control the variable value.
SignalValueSegment	Provides a signal that replays numerical data.
DataFileSegment	Provides a signal that replays numerical data from a file.
LoopSegment	Repeats the contained segments for the specified count value.
OperationSegment	Combines two segments by the specified mathematical operation. Addition and multiplication is supported.

**Note**

For executing stimulus signals on the simulation platform, the following conditions must be fulfilled:

- The real-time application must be built with the **Enable real-time testing** option set. Refer to [Code Generation Dialog \(Model Configuration Parameters Dialogs\) \(RTI and RTI-MP Implementation Reference !\[\]\(d27edc55493507da2f9b8c7a52b3b96f\_img.jpg\)](#)).
- Real-Time Testing must be installed on your host PC.

Signal descriptions can be created and saved to an STI or STZ file using a `SignalGeneratorWriter` object. An STZ file is a zipped STI file. With a `SignalGeneratorReader` object, you can read an STI or STZ file containing signal descriptions.

A more comfortable way to create a signal description is to use the Signal Editor in ControlDesk or AutomationDesk. Refer to [ControlDesk Signal Editor](#)  or [Handling Signals \(AutomationDesk Implementing Signal-Based Tests\)](#) .

## Implementing an MAPort Client Application

### Introduction

A simple MAPort client application can be created with a basic section in your source code for MAPort instantiation and additional sections for variable handling and capturing.

### Basic implementation of an MAPort client application

Independently of the use case for the MAPort, the source code has to contain the creation of the MAPortFactory and the MAPort object. You will also find the creation of the TestbenchFactory and the Testbench object.

The following source codes show you the general concepts of implementing an MAPort client application using C#/.NET.

For detailed implementation features and source code in the other programming languages Python and M, refer to the demos. There you also find demos how to use the Framework.

```
using System;
using System.IO;
using System.Collections.Generic;
using ASAM.XIL.Interfaces.Testbench;
using ASAM.XIL.Interfaces.Testbench.Common.Error;
using ASAM.XIL.Interfaces.Testbench.Common.ValueContainer;
using ASAM.XIL.Interfaces.Testbench.Common.ValueContainer.Enum;
using ASAM.XIL.Implementation.TestbenchFactory.Testbench;
using ASAM.XIL.Interfaces.Testbench.MAPort;
class MAPort_Using
{
    // Specify the path to the MAPort configuration file
    private string MAPortConfigFile = @"D:\Work\MAPortConfig.xml";
```

```

public void Execute()
{
    IMAPort MyMAPort = null;
    try
    {
        // Create the Testbench
        ITestbenchFactory TBFactory = new TestbenchFactory();
        ITestbench TB = TBFactory.CreateVendorSpecificTestbench("dSPACE GmbH", "XIL API", "2021-A");

        // Create and configure MAPort object
        MyMAPort = TB.MAPortFactory.CreateMAPort("MyMAPort");
        IMAPortConfig maPortConfig = MyMAPort.LoadConfiguration(MAPortConfigFile);
        // Establish a connection to the platform, assuming the simulation application is already running
        MyMAPort.Configure(maPortConfig, false);
        // Start the simulation
        if (MyMAPort.State != MAPortState.eSIMULATION_RUNNING)
        {
            MyMAPort.StartSimulation();
        }
        // Further actions (reading, writing, capturing)
        ...
    }
    // Exception handling
    catch (TestbenchPortException ex)
    {
        Console.WriteLine(ex.ToString());
        Console.WriteLine(ex.VendorCodeDescription)
    }
    catch (System.Exception sysEx)
    {
        Console.WriteLine(sysEx.ToString());
    }
    finally
    {
        // Free the allocated memory of the MAPort instance
        if (null != MyMAPort)
        {
            MyMAPort.Dispose();
        }
    }
}

static void Main(string[] args)
{
    MAPort_Using demo = new MAPort_Using();
    demo.Execute();
}
}

```

### Implementing read and write access

You can add code to the basic section for implementing access to a single variable of the simulation application. You can use the **Read** method for reading the value of a variable and the **Write** method to write a value to a variable. You can check beforehand whether the variable you want to access is readable or writable. The **GetDataType** method returns the data type of the specified variable.

```

...
// Get the names of the available variables
IList<string> AvailableVariables = MyMAPort.VariableNames;
// Read the value of the second variable in the list
MyVar2 = MyMAPort.AvailableVariables[1];
bool MyVar_Readable = MyMAPort.IsReadable(MyVar2);
DataType MyVar_DataType = MyMAPort.GetDataType(MyVar2);
// Assume that the data type is float
IFloatValue ValueVar2 = (IFloatValue)MyMAPort.Read(MyVar2);
// Check, whether the variable is writable and write it back to the model variable
bool MyVar_Writable = MyMAPort.IsWritable(MyVar2);
MyMAPort.Write(MyVar2, ValueVar2);
...

```

**Note**

If you are using a multiprocessor platform (MP system), the names of the separated submodels are used as a prefix for the variable names.

**Implementing data capture**

You can add code to the basic section for implementing data capture of a specific task in the simulation application.

```

using ASAM.XIL.Interfaces.Testbench.Common.Capturing;
using ASAM.XIL.Interfaces.Testbench.Common.Capturing.Enum;
using ASAM.XIL.Interfaces.Testbench.Common.CaptureResult;
using ASAM.XIL.Interfaces.Testbench.Common.WatcherHandling;
public void Execute()
{
    ...
    ICapture MyCapture = null;
    ...
    // Create Capture
    MyCapture = MyMAPort.CreateCapture(Task);
    // Get the available task names
    IList<string> AvailableTasks = MyMAPort.TaskNames;
    // Set the variables to be captured
    MyCapture.Variables = new List<string>()
    {
        MyVar2,
        MyVar7
    };
    ...
}
// Create a dictionary providing keys for the condition variables
Dictionary<string, string> MyStartWatchDefines = new Dictionary<string, string>();
MyStartWatchDefines.Add("RefPos", MyVar2);
// Specify start condition, triggered by variable
// Trigger if the reference position rises above 45
IConditionWatcher MyStartWatcher = TB.WatcherFactory.CreateConditionWatcher("posedge(RefPos,45)", MyStartWatchDefines);
// create TimeSpanDuration
ITimeSpanDuration MyStartTriggerDelay = TB.DurationFactory.CreateTimeSpanDuration(0.0);
MyCapture.SetStartTrigger(MyStartWatcher, MyStartTriggerDelay);
// Specify stop condition, fix duration
IDurationWatcher MyStopWatcher = TB.WatcherFactory.CreateDurationWatcherByTimeSpan(0.3);
ITimeSpanDuration MyStopTriggerDelay = TB.DurationFactory.CreateTimeSpanDuration(0.0);
MyCapture.SetStopTrigger(MyStopWatcher, MyStopTriggerDelay);

```

```

// Specify how to handle the capture result, write to internal memory
ICaptureResultMemoryWriter MyResultWriter = TB.CapturingFactory.CreateCaptureResultMemoryWriter();
// Start the capture
MyCapture.Start(MyResultWriter);
// Stop the capture in case of an exception
// Example of a finally path of a corresponding try-finally construct
...
finally
{
    ...
    MyCapture.Stop();
    ...
}
// Access the capture result values
ICaptureResult MyCaptureResult = MyCapture.CaptureResult;
ISignalValue Result_Var2 = MyCaptureResult.ExtractSignalValue(Task, MyVar2);
// Get the x-axis values
IFloatVectorValue XValues = (IFloatVectorValue)Result_Var2.XVector;
// Get the variable values
IFloatVectorValue Var2Values = (IFloatVectorValue)Result_Var2.FcnValues;
// Output a single value on position i
Console.WriteLine(XValues.Value[i], Var2Values.Value[i]);
// Free the allocated memory
// Example of a finally path of a corresponding try-finally construct
...
finally
{
    ...
    if (null != MyCapture)
    {
        MyCapture.Dispose();
    }
}

```

**Tip**

To get a valid task name for the **CreateCapture** method, you can use the **MAPortVariableInfo.GetAvailableTasks()** method. The task names of multicore and multiprocessor applications must include the name of the subapplication, e.g., *MySubApplication/MyTask*.

**Note**

The **Dispose** method is to be used for

- An MAPort instance
- A Capture instance
- An EESPort instance, if you are using the electrical error simulation feature of XIL API .NET.

A SignalGenerator instance requires the **DestroyOnTarget** method to be called to free the allocated resources.

By default, the complete capture result is stored in the memory. If you have a storage problem because of a high amount of variables or a long measurement duration, you can specify to reduce the size of the stored capture result. This is done via the **KeepCompleteCaptureResult** setting in the MAPort configuration file. If you do not add this setting to the configuration file, the

default value 1 is used, i.e., the complete capture result is stored in the memory. If you set the value to 0, you have to regularly get the measurement data with a `Capture.Fetch(false)`. Using the `Capture.CaptureResult` property leads to an exception.

A file writer, such as the `CaptureResultMDFWriter`, works independently of this setting. Its default behavior is not to keep the complete capture result.

## Implementing advanced features

The basic information of the following features is listed here. For more information, refer to the demo projects and the ASAM documentation.

### ■ CaptureEvents

A `CaptureResult` can contain different types of events:

- `eDATAFRAMESTART`: Start of the capture
- `eDATAFRAMESTOP`: Stop of the capture
- `eSTARTTRIGGER`: Timestamp of the start trigger
- `eSTOPTRIGGER`: Timestamp of the stop trigger
- `eCLIENTEVENT`: Timestamp of a user-defined event

During capture, a client event can be set by using

`Capture.TriggerClientEvent(<EventID>, <EventDescription>)`.

With the `Capture.Events` property you can get the list of all captured events.

### ■ TargetScript

Instead of the `SignalGenerator`, you can use the `TargetScript` element to manage a real-time script for stimulation.

With the `TargetScript.Load` method, you can load an RTT sequence, i.e., a Python file created with Real-Time Testing. For more information, refer to [Implementing RTT Sequences \(Real-Time Testing Guide\)](#).

Optionally, you can create an XML file with the same name as the target script for configuration. For information on the XML schema definition, refer to [TargetScript Configuration \(dSPACE XIL API Reference\)](#).

### ■ DownloadParameterSets

With the `MAPort.DownloadParameterSets` method, you can load the specified parameter set files to write the contained parameter values to the model variables.

You have to note the following prerequisites:

- You can use only CDFX files as parameter set files that you can create with the Parameter Editor in ControlDesk.
- The `EnableScaling` setting must be set to `true`, because CDFX supports only physical values.
- You can use the `DownloadParameterSets` feature only with the MATLAB releases that are supported by the current dSPACE Release.

## Demo projects

For an overview on the available demo projects, refer to [Demo Projects for MAPort Client Applications](#) on page 49.



## Demo Projects for MAPort Client Applications

### Introduction

In the installation folder of XIL API .NET, you find some simulation applications and demo projects for various use cases. The source code is available in C#, Python and M.

### Available simulation applications

To demonstrate the execution of an MAPort client application, you need a simulation application running on the configured simulation platform.

In `<InstallationPath>\Demos\SampleExperiments`, you find the turn signal demo applications and related files for the following dSPACE platforms:

- DS1006 and DS1006MP
- DS1007 and DS1007MP
- DS1202 and DS1202MC (MicroLabBox)  
MicroLabBox can be used as multicore platform.
- DS1401 (MicroAutoBox II)
- DS1403 and DS1403MC (MicroAutoBox III)  
MicroAutoBox III can be used as multicore platform.
- SCALEXIO, SCALEXIOMC and SCALEXIOMP  
SCALEXIO can be used as multicore and multiprocessor platform.
- DS6001, DS6001MC and DS6001MP  
The DS6001 Processor Board is a SCALEXIO platform that can be used as multicore and multiprocessor platform.
- VEOS, VEOSMC, VEOSVECU, and VEOSVECUMC  
VEOS can be used with multiple environment VPUs.

You have to specify the port configuration file for the corresponding simulation platform depending on the demo application. You find the port configuration files in `<InstallationPath>\Demos\MAPort\Common\PortConfigurations`.

### C# demo projects

`<InstallationPath>\Demos\MAPort\C#` contains demo projects for the various use cases. Read the comments in the files to learn which code lines you have adapt to your working environment.

#### Note

Copy the complete folder to a folder where you have write access, for example to the working folder for your Visual Studio projects. You can use the Copy XIL API .NET Demos shortcut in the Windows Start menu to do this.

By opening the **MAPortDemos.sln** solution file in Visual Studio, you have access to the available demo projects for editing and building.

Demo Name	Description
1_ReadWrite	This demo shows you how to read and write variables on the simulation platform.
2_BasicCapturing	This demo shows you how to capture two model variables started by a trigger. The capture result is stored in the internal memory.
3_DurationWatcherCapturing	This demo shows you how to control the capturing via duration watcher.
4_ConditionWatcherCapturing	This demo shows you how to control the capturing via condition watcher.
5_CapturingToMDFFile	This demo shows you how to store the capture result to an MDF file.
6_FetchingDataWhileCapturing	This demo shows you how to get a current capture result.
7_StimulusFromSTZFile	This demo shows you how to use an STZ stimulus file. The <b>DemoStimulus.stz</b> file that is required by this demo is contained in <b>&lt;InstallationPath&gt;\Demos\MAPort\C#</b> .
8_StimulusFromModifiedSignal	This demo shows you how to modify a stimulus signal and to write the capture result to an MDF file.
8b_StimulusFromDataFileSegment	This demo shows you how to use the created MDF file as input for the signal generator.
9_StimulusFromCreatedSignal	This demo shows you how to create a stimulus signal and save it to an STZ file.
10_ValueContainer	This demo shows you how to use the value container classes.
11_CDFXHandling	This demo shows you how to write multiple variable values to the platform by using the <b>DownloadParameterSets</b> method. The CDFX-file to be loaded can be created in ControlDesk.
12_StimulusParameterizedSignals	This demo shows you how to stimulate model variables with a signal read from an STI file. In the <b>SignalDescriptionSet</b> , parameters are specified that are used to set the values of some model variables during initialization. By changing the values of the parameters, the corresponding values of the model variables are changed too.
13_CheckVariableNames	This demo shows you how to use the <b>CheckVariableNames</b> method to evaluate the existence of variables in the currently loaded real-time application.
14_TargetScript	This demo shows you how to use a Real-Time Testing application as target script. This lets you access and modify variables on a dSPACE platform at real-time. The definitions of the parameters used in the target script must be specified in a separate XML file, which must have the same name as the target script.
15_TriggerClientEvents_Memory	This demo shows you how to trigger client events while a capture is running. In this demo script the capture data is stored in memory.
16_TriggerClientEvents_MDFFile	This demo shows you how to trigger client events while a capture is running. In this demo script the capture data is stored in an MDF file.

The demo projects must be compiled before you can execute them. For information on how to do this, refer to [How to Create XIL API C# Projects](#) on page 25.

---

**Python demo scripts**

<InstallationPath>\Demos\MAPort\Python contains scripts similar to the C# demo scripts implemented in the Python programming language. After you have modified the relevant code lines to your working environment, you can execute a script with the Python interpreter without compiling it beforehand.

---

**M demo scripts**

<InstallationPath>\Demos\MAPort\Matlab contains scripts similar to the C# demo scripts implemented in the M programming language. After you have modified the relevant code lines to your working environment, you can execute a script in MATLAB.

---

**Related topics****Basics**

[Implementing an MAPort Client Application..... 44](#)



# Implementing an EESPort Client Application

Where to go from here

Information in this section

Basic Information on the EESPort Implementation.....	54
Basic Information on Simulating Electrical Errors in the Wiring.....	75

## Basic Information on the EESPort Implementation

### Where to go from here

### Information in this section

<a href="#">Basics on the EESPort.....</a>	<a href="#">54</a>
To provide basic information on EESPort configuration and simulation.	
<a href="#">Creating dSPACE EESPort Configuration Files.....</a>	<a href="#">58</a>
Provides information on the contents of an EESPort configuration file and how to create it.	
<a href="#">Using the EESPortConfiguration API.....</a>	<a href="#">62</a>
Provides information on how to create a dSPACE EESPort configuration file by using the EESPortConfiguration API.	
<a href="#">Latencies when Performing Electrical Error Simulation.....</a>	<a href="#">68</a>
Simulating electrical errors is limited by the latencies of the used simulation system.	
<a href="#">Monitoring the Switching Behavior of Electrical Error Simulation Hardware.....</a>	<a href="#">69</a>
When performing electrical error simulation, you can monitor and trace the switching behavior of the electrical error simulation hardware.	
<a href="#">dSPACE XIL API EESPort Demo.....</a>	<a href="#">73</a>
Demonstrates the basic features of the XIL API EESPort implementation.	

## Basics on the EESPort

### Introduction

To provide basic information on EESPort configuration and simulation.

**Warning**

Performing electrical error simulation might be dangerous. You have to note the following safety precautions.

**⚠ WARNING****Risk of unexpected high currents and voltages due to electrical error simulation**

During electrical error simulation, high currents and voltages might be present on board channels and/or connector pins, which are not expected. This can result in death, personal injury, fire, and/or damage to the simulator and connected external devices.

- Setup a test zone to avoid contact to electrical components of the simulator and to moveable physical components controlled by the simulator.

**Note**

Especially signal measurement channels (such as channels connected in parallel or interconnected reference lines) can carry high currents.

**Preparing the access to failure insertion units**

The Electrical Error Simulation Port (EESPort) provides access to a failure insertion unit (FIU) for simulating errors in the ECU wiring.

Usually, it is required to reference further namespaces provided by the Testbench in your source code, such as the Error namespace.

**Using integrated SCALEXIO FIU** You can access the simulation platform with integrated SCALEXIO FIU only, if the simulation application is already running. Functions for registering the platform and downloading the simulation application to it are not contained in the EESPort implementation.

**Creating an EESPort instance**

First you have to create a Testbench instance that provides the vendor-specific configuration of the XIL API .NET implementation. The Testbench instance allows you to create an EESPort factory for creating and handling EESPort objects. The EESPort object gives you access to its methods and properties. For further information on the Testbench, refer to [Basics on the Testbench](#) on page 35.

The following code shows you, which code is to be added to the basic code to instantiate an EESPort object.

```
using ASAM.XIL.Interfaces.Testbench.EESPort;
using ASAM.XIL.Interfaces.Testbench.EESPort.Enum;
using ASAM.XIL.Interfaces.Testbench.EESPort.Error;

public void Execute()
{
    IEESPort MyEESPort = null;
    try
    {
        ...
        // Create EESPort based on the Testbench TB
        MyEESPort = TB.EESPortFactory.CreateEESPort("MyEESPort");
        ...
    }
    finally
    {
        ...
        // Free the instantiated object if it is no longer required
        MyEESPort.Dispose();
    }
    ...
}
```

## Configuring the EESPort

An instantiated EESPort object gets the basic information on the electrical error simulation hardware (EES hardware) to be used via the port configuration file in XML format. The port configuration file must contain the data for the interface used for the EES hardware connection. Depending on the selected driver type further data is to be specified.

The configuration file also contains the path to the signal list of your simulator, that provides information on the available signals and potentials.

You can specify mapping information for potentials and signals available with your simulator to make your C# client applications reusable for different simulators. While the potential mapping is required for platforms providing several potentials, the signal mapping is optional.

Via the real-time configuration you can configure the variables to be used for monitoring the switching behavior of the specified error sets.

For further information, refer to [Creating dSPACE EESPort Configuration Files](#) on page 58.

You can find examples of EESPort configurations in  
 <InstallationPath>\Demos\EESPort\CommonData\PortConfigurations.  
 To modify the demos, you have to copy them via Start - dSPACE XIL API .NET  
 <Version> - Copy XIL API .NET Demos to a folder with modify rights.



**Supported dSPACE platforms**

The following dSPACE platforms are supported by the dSPACE XIL API EESPort implementation:

- dSPACE Simulator Mid-Size based on DS2210
- dSPACE Simulator Mid-Size based on DS2211
- dSPACE Simulator Full-Size (custom simulator and SCALEXIO system with discrete FIU)
- SCALEXIO system (with integrated SCALEXIO FIU)

For these platforms, dSPACE provides EES hardware. For an overview on the EES hardware, refer to [Hardware for Failure Simulation](#) on page 85.

**Specifying the error configuration**

The errors to be tested must be specified in an error configuration. You can either create an error configuration within your script or load an error configuration from your file system. Specified errors can be combined in error sets. Afterwards, you have to download the error configuration to the hardware and activate it.

For further information, refer to [Basic Information on Configuring Errors](#) on page 108.

**Starting an electrical error simulation**

After the EESPort is configured and the error configuration is downloaded, the electrical error simulation is ready to run. The signal errors (pin failures) are activated in the order specified in your error configuration after you have triggered the execution of their related error set.

**Tip**

In the ASAM AE XIL API standard, the configuration of errors and the activation of errors is separated. After the error configuration is downloaded to the real-time hardware, each contained error set is activated by a related trigger. The time delay for activating an error therefore only depends on the latency of the error simulation hardware.

**Note**

dSPACE XIL API .NET EESPort implementation supports only manual and software triggers.

For further information, refer to [Activating Electrical Error Simulation](#) on page 114.

**Terminating an electrical error simulation**

It is recommended to use **try-finally** constructs in your application to handle exceptions. This allows you to terminate your XIL API application in a predefined manner.

- Internally, the resources of your script should have a definite location to be freed.

- Externally, the connected hardware should be driven in a secure state before the application is quit by an exception.

#### ⚠ CAUTION

To avoid personal injury or hardware damage, you must use the **Dispose** method in the **finally** path to free the instantiated resources.

To terminate an electrical error simulation, you must firstly deactivate the error configuration. Then, you can unload the error configuration and disconnect the EESPort from the EES hardware.

For further information, refer to [Deactivating Electrical Error](#) on page 117.

## Creating dSPACE EESPort Configuration Files

### Introduction

Provides information on the contents of a dSPACE EESPort configuration file and how to create it.

### Contents of a port configuration file

For each simulator architecture you have to provide an individual port configuration file (PORTCONFIG file). The contents must suit to the connected simulator and its FIU hardware. However, by replacing the port configuration file, it might be possible, that you do not need to modify your EESPort application when you want to run the electrical error simulation on another simulator with a similar architecture. You have to specify the interface and the signal list to be used. While the signal mapping is optional, the potential mapping is required for most of the simulation platforms.

You can create the EESPort configuration file by using the EESPortConfiguration API. For more information, refer to [Using the EESPortConfiguration API](#) on page 62.

**Specifying the driver type** The FIU hardware of a system with discrete FIU is connected by a serial interface (RS232) or by a CAN interface to the electrical error simulation system. With the **DriverType** attribute the interface is to be specified.

When using a SCALEXIO system, the integrated SCALEXIO FIU is internally connected, and the driver type is to be specified as *SCLX*.

#### Example of a port configuration file for a system with discrete

**FIU** With the following port configuration file, you can access a DS749 FIU of a dSPACE Simulator Mid-Size via serial interface. The available signals are stored in a signal list in CSV format. This is an example for a system with discrete FIU.

```
<?xml version="1.0" encoding="utf-8"?>
<PortConfigurations>
  <EESPortConfig>
    <HardwareConfigurations>
      <Drivers>
        <Driver ID="1" DriverType="RS232">
          <COMPort>COM1</COMPort>
        </Driver>
      </Drivers>
      <HardwareConfiguration ID="1" SignalListPath="signallist mid-size ds749.csv" DriverId="1" />
      <PotentialMapping>
      </PotentialMapping>
      <SignalMapping>
      </SignalMapping>
    </HardwareConfigurations>
  </EESPortConfig>
</PortConfigurations>
```

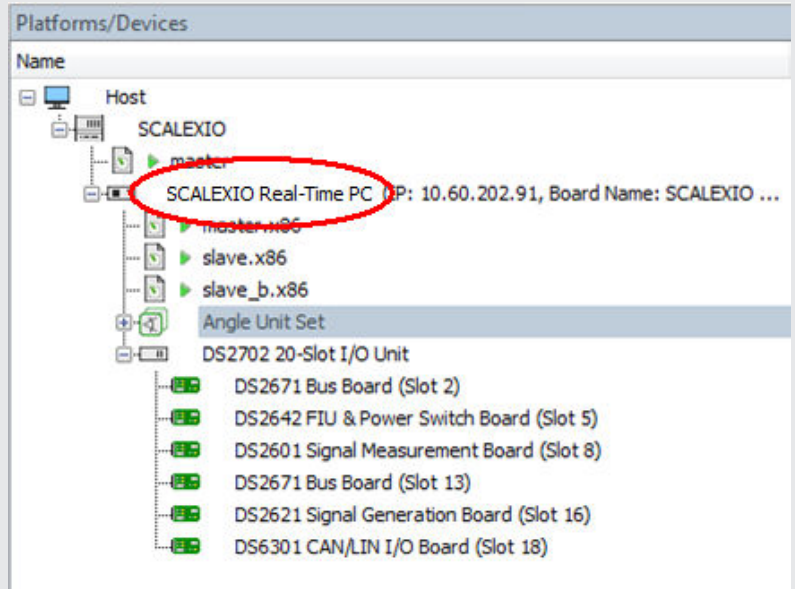
#### Example of a port configuration file for a system with integrated

**SCALEXIO FIU** If you use a system with integrated SCALEXIO FIU for failure simulation, the port configuration file looks slightly different. Instead of specifying the interface to the FIU hardware, you have to specify the platform name and the real-time application running on the SCALEXIO system. The real-time application includes the required signal list and the FIU configuration data.

```
<?xml version="1.0" encoding="utf-8"?>
<PortConfigurations>
  <EESPortConfig>
    <HardwareConfigurations>
      <Drivers>
        <Driver ID="1" DriverType="SCLX">
          <PlatformName>SCALEXIO Real-Time PC</PlatformName>
          <RealTimeApplicationFile>SCALEXIO_Application.rta</RealTimeApplicationFile>
        </Driver>
      </Drivers>
      <HardwareConfiguration ID="1" SignalListPath="SCALEXIO_Application.rta" DriverId="1" />
      <PotentialMapping>
      </PotentialMapping>
      <SignalMapping>
      </SignalMapping>
    </HardwareConfigurations>
  </EESPortConfig>
</PortConfigurations>
```

**Note**

You must specify the name of the registered SCALEXIO processing unit as `PlatformName`.



**Mappings in the port configuration file** The port configuration file provides two mapping sections.

- **PotentialMapping**

The potential mapping is required if your simulation has errors against Potential, Ground or Batt. Via the potential mapping, a potential name is mapped to a unique identifier and a potential type. By entering the related potential type, you specify the potentials used for *Short to Batt* (specified type is *Ubat*) and *Short to Ground* (specified type is *Gnd*) errors.

You are recommended to use potential mapping for dSPACE failure simulation hardware. Potential mapping is not required for all the hard-wired potentials of your failure simulation hardware, because the dSPACE XIL API implementation knows these potentials.

Observe the following hardware dependencies:

- Potential mapping is not required for DS291, DS749, DS5355/DS5390, and DS1450.
- You can specify one Ubat potential type for the following failure simulation hardware:
  - DS293
  - DS789
  - DS791

- DS793
- SCALEXIO
- You can specify one GND potential type for the following failure simulation hardware:
  - DS293
  - DS791
  - DS793

```
<PotentialMapping>
  <Potential ID="0" Name="Pot1" Type="Potential" HardwareConfigurationId="1" />
  <Potential ID="1" Name="Pot2" Type="Ubat" HardwareConfigurationId="1" />
  <Potential ID="2" Name="Pot3" Type="Gnd" HardwareConfigurationId="1" />
</PotentialMapping>
```

#### Note

##### Using SCALEXIO

The value of the **Name** attribute in the potential mapping is a combination of the ECU name of the SCALEXIO Power Switch and the pin name, e.g., **Name="Power Switch 1\VBAT"**.

#### SignalMapping

The signal mapping is required if you want to address a signal with the XIL API by a user-defined string. The input and output signals of a simulator are specified by the names of the pins and ECUs connected to the signals. The signal mapping in the port configuration combines this information.

Each signal to be specified has a separate entry.

The following example shows the entry for *Signal 1* that relates to *Pin\_2* of the *ECU\_1*.

```
<SignalMapping>
  <Signal MappingName="Signal 1" ECUName="ECU_1" PinName="Pin_2" />
</SignalMapping>
```

Now, you can specify the signal in an abstract way. This might allow you to use the same error configuration with another simulator only by replacing the signal mapping section in your port configuration file.

```
// Creating an InterruptError without signal mapping
factory.CreateInterruptError("ECU_1\Pin_2");

// Creating an InterruptError with signal mapping
factory.CreateInterruptError("Signal 1");
```

**Real-time configuration in the port configuration file** The port configuration file provides a real-time configuration section for specifying the monitoring of the switching behavior.

#### RealTimeConfiguration

The real-time configuration is required if you want to monitor the switching behavior of your simulation. Via the platform name and the path to the system description file, you specify the variables to be monitored on the platform.

The **Tracing** section lets you enable monitoring. You can either use default entries for the monitoring variables or customized variables. For customized variable entries, you can, for example, configure an existing model variable in your variable description file as **Value**.

The **SoftwareTrigger** section lets you set the sampling rate for polling the trigger variable of a trigger condition.

The following example shows the entry for a system with integrated SCALEXIO FIU using default variables.

```
<RealTimeConfiguration PlatformName="SCALEXIO Real-Time PC" SystemDescriptionFilePath="Scalexio_Application.sdf">
  <Tracing Enabled="true">
    <Variable Value="XIL API/EESPort/Error Activated" Type="ErrorActivated" />
    <Variable Value="XIL API/EESPort/Active ErrorSet" Type="ActiveErrorSet" />
    <Variable Value="XIL API/EESPort/Error Switching" Type="ErrorSwitching" />
    <Variable Value="XIL API/EESPort/Flags" Type="Flags" />
    <Variable Value="XIL API/EESPort/Trigger" Type="Trigger" />
  </Tracing>
  <SoftwareTrigger PollingInterval="0.001" />
</RealTimeConfiguration>
```

### Loading the port configuration

By executing the **LoadConfiguration** method, you create a configuration object, that is then used by the EESPort object's **Configure** method. Now, you have finished the configuration of the EESPort.

```
// Load port configuration file, adapt the path to your environment
var portConfig = MyEESPort.LoadConfiguration("D:\Work\MidSizeBasedOnDS2211.portconfig");

// Apply the configuration to the EESPort
MyEESPort.Configure(portConfig);
...
```

### Related topics

#### Basics

[Using the EESPortConfiguration API..... 62](#)

#### References

[dSPACE EESPort Configuration File \(dSPACE XIL API Reference !\[\]\(ab4e2b3fc7e7887b7a72f548aa6f5e60\_img.jpg\)\)](#)

## Using the EESPortConfiguration API

### Introduction

Provides information on how to create a dSPACE EESPort configuration file by using the EESPortConfiguration API.

### Basics on the EESPort configuration file

For basic information on the EESPort configuration file, refer to [Creating dSPACE EESPort Configuration Files](#) on page 58.

## Using the EESPortConfiguration API

The EESPortConfiguration API provides methods to create, edit, save and load the EESPort configuration files. If the specified EESPort configuration file (PORTCONFIG file) already exists, it is overwritten.

You find a demo in

<InstallationPath>\Demos\EESPort\Python\EESPortPortConfig.py. To use the EESPortConfiguration API in an application, you have to reference the assembly dSPACE.XIL.Testbench.EESPort.Interfaces.Extended.dll (Version 5.0.0.0, PublicKeyToken=f9604847d8afbfb).

You have to implement the following steps in your application for creating an EESPort configuration file.

### Creating a new EESPort

```
testbenchFactory = TestbenchFactory()
testbench = testbenchFactory.CreateVendorSpecificTestbench(vendorName, productName, productVersion)
eesPort = testbench.EESPortFactory.CreateEESPort("Demo EESPortConfiguration API")
```

**Adding an OnError event handler** The OnError eventhandler is called when an error occurs during simulation, e.g., a software trigger timeout exception. Do not throw any exceptions in the OnError eventhandler. For more information, refer to the **SoftwareTrigger.py** demo script.

```
dsEESPort.OnError += OnErrorEventHandler
```

### Creating a new EESPort configuration

```
dsEESPort = IDEEESPort(eesPort)
dsEESPortConfig = dsEESPort.CreateConfiguration()
```

**Adding a hardware configuration** Depending on the connected EESPort hardware, you have to specify the configuration type that includes the driver type of the hardware.

Configuration Type	Meaning
MidSizeFullSizeVariant1	The hardware is using the serial interface (RS232).
FullSizeVariant2	The hardware is using a CAN interface.
SCALEXIO	The hardware is integrated in the SCALEXIO system.

Depending on the specified configuration type, you can use the following properties to configure your EESPort.

Properties	Meaning
MidSizeFullSizeVariant1	SignalListFilePath
	EcuVersion
	COMPort

Specifies the path to the signal list. The path can be absolute or relative to the path of the configuration file.

Specifies the ECU version of the signal list. The parameter is optional and its default value is 1.

Specifies the port when using an RS232 interface:

- COM1
- COM2
- ...

Properties		Meaning
FullSizeVariant2	SignalListFilePath	Specifies the path to the signal list. The path can be absolute or relative to the path of the configuration file.
	EcuVersion	Specifies the ECU version of the signal list. The parameter is optional and its default value is 1.
	VendorName	Specifies the vendor name of the CAN interface supported by dSPACE hardware. <ul style="list-style-type: none"> <li>▪ dSPACE</li> <li>▪ Eberspaecher</li> <li>▪ Kvaser</li> <li>▪ Vector Informatik</li> <li>▪ Unknown vendor</li> </ul>
	InterfaceName	Specifies the interface name of the CAN hardware. <ul style="list-style-type: none"> <li>▪ Calibration Hub</li> <li>▪ DCI-CAN/LIN1</li> <li>▪ DCI-CAN2</li> <li>▪ Leaf</li> <li>▪ Memorator Professional</li> <li>▪ USBcan II</li> <li>▪ USBcan Professional</li> <li>▪ PCAN-miniPCle FD</li> <li>▪ CANcaseXL</li> <li>▪ VN1610</li> <li>▪ VN1611</li> <li>▪ VN1630</li> <li>▪ VN1640</li> <li>▪ VN5610</li> <li>▪ VN5610A</li> <li>▪ VN7600</li> <li>▪ VN8900</li> <li>▪ Virtual</li> <li>▪ Unknown interface</li> </ul>
	ChannelIdentifier	Specifies the channel identifier of the CAN hardware.
	SerialNumber	Specifies the serial number of the hardware as integer.
	PlatformName	Specifies the platform name of the SCALEXIO system used when you registered the platform.
	RTAFilePath	Specifies the path to the application file. The path can be absolute or relative to the path of the configuration file.
	OverrideAccess	To specify whether to automatically disconnect another client that might be connected to the SCALEXIO failure simulation hardware when you configure the XIL API EESPort. <ul style="list-style-type: none"> <li>▪ <b>OverrideAccess="false"</b>: The previously connected EESPort client keeps connected to the SCALEXIO FIU hardware.</li> </ul>
SCALEXIO		



Properties	Meaning
	<ul style="list-style-type: none"> <li>▪ <b>OverrideAccess="true"</b>: The newly configured EESPort client is connected to the SCALEXIO FIU hardware. The disconnected client is not able to simulate errors.</li> </ul>

The following examples show how to specify the different configuration types.

MidSizeFullSizeVariant1:

```
dsConfiguration = IDSMidSizeFullSizeVariant1Configuration( \
    dsEESPortConfig.Configurations.Add(ConfigurationType.MidSizeFullSizeVariant1))
dsConfiguration.SignalListFilePath = signalListFilePathFullSizeVariant1
dsConfiguration.ECUVersion = 1
dsConfiguration.COMPort = "COM3"
```

FullSizeVariant2:

```
dsConfiguration = IDSTFullSizeVariant2Configuration( \
    dsEESPortConfig.Configurations.Add(ConfigurationType.FullSizeVariant2))
dsConfiguration.CANAPIVersion = DSCANAPIVersion.v20
dsConfiguration.VendorName = CANVendorNames.VectorInformatik
dsConfiguration.InterfaceName = CANInterfaceNames.CANcaseXL
dsConfiguration.ChannelIdentifier = "1"
dsConfiguration.SerialNumber = "0"
```

SCALEXIO:

```
dsConfiguration = IDSScalexioConfiguration( \
    dsEESPortConfig.Configurations.Add(ConfigurationType.SCALEXIO))
dsConfiguration.PlatformName = "Scalexio Realtime PC"
dsConfiguration.RTAFilePath = rtaFilePathSCALEXIO
dsConfiguration.OverrideAccess = False
```

**Adding a signal mapping** The signal mapping is optional.

```
dsConfiguration.Signals.Add(MappingName, ECUName, PinName)
```

**Adding a potential mapping** The potential mapping supports the following potential types.

Potential Type	Meaning
Ubat	Potential used for <i>Short to Batt</i>
Gnd	Potential used for <i>Short to Ground</i>
Potential	If you use a configurable potential.

```
dsConfiguration.Potentials.Add(PotentialName, PotentialType.Ubat)
dsConfiguration.Potentials.Add(PotentialName, PotentialType.Gnd)
dsConfiguration.Potentials.Add(PotentialName, PotentialType.Potential)
```

**Adding variable tracing** Variable tracing is configured in the port configuration file by specifying the **RealTimeConfiguration** properties.

Properties		Meaning
RealTimeConfiguration		
PlatformName		Specifies the platform name used when you registered the platform.
SystemDescriptionFilePath		Specifies the path of the variable description file (SDF file) that contains the tracing variables for monitoring the switching behavior of the failure simulation hardware.
Enabled		Enables the monitoring of the switching behavior and the triggering by software. This property is ignored when you set the EESPort to Offline mode. In offline mode, monitoring the switching behavior and triggering by software are not supported.
Tracing		
	ActiveErrorSetVariable	Specifies the name of the measurement variable representing the activated error set. The default name is <b>XIL API/EESPort/Active ErrorSet</b> .
	ErrorActivatedVariable	Specifies the name of the measurement variable representing the activation state of one or more errors. The default name is <b>XIL API/EESPort/Error Activated</b> .
	ErrorSwitchingVariable	Specifies the name of the measurement variable representing the switching state of an error. The default name is <b>XIL API/EESPort/Error Switching</b> .
	FlagsVariable	This variable is used internally. Do not use this variable to monitor the switching behavior of the failure simulation hardware. The default name is <b>XIL API/EESPort/Flags</b> .
	TriggerVariable	This variable is used internally. Do not use this variable to monitor the switching behavior of the failure simulation hardware. The default name is <b>XIL API/EESPort/Trigger</b> .
SoftwareTrigger		
	PollingInterval	Specifies the sample rate for polling the trigger variable of a trigger condition in seconds.

```
# Set platform name and system description file path
dsEESPortConfig.RealTimeConfiguration.PlatformName = "<PlatformName>"
dsEESPortConfig.RealTimeConfiguration.SystemDescriptionFilePath = "<SDFPath>"
# Enable tracing
dsEESPortConfig.RealTimeConfiguration.Enabled = True      # or False to disable
# Set trace variable paths
dsEESPortConfig.RealTimeConfiguration.Tracing.ActiveErrorSetVariable = "XIL API/EESPort/Active ErrorSet"
dsEESPortConfig.RealTimeConfiguration.Tracing.ErrorActivatedVariable = "XIL API/EESPort/Error Activated"
dsEESPortConfig.RealTimeConfiguration.Tracing.ErrorSwitchingVariable = "XIL API/EESPort/Error Switching"
dsEESPortConfig.RealTimeConfiguration.Tracing.FlagsVariable = "XIL API/EESPort/Flags"
dsEESPortConfig.RealTimeConfiguration.Tracing.TriggerVariable = "XIL API/EESPort/Trigger"
# Set polling interval for software trigger
dsEESPortConfig.RealTimeConfiguration.SoftwareTrigger.PollingInterval = "0.001"
```

### Saving the created EESPort configuration

```
dsEESPortConfig.Save(portConfigPath, productVersion)
```

### Loading an existing EESPort configuration

```
dsEESPortConfig.Load(portConfigPath)
```

**Disposing the created objects** Your application for creating an EESPort configuration file must free the created objects in a try-finally construct when finishing or terminating.

For the above described commands, the termination code looks like this.

```
try:
    ...
finally:
    eesPortConfig      = None
    errorConfiguration = None
    testbench          = None
    testbenchFactory   = None

    eesPort.Dispose()
    eesPort = None
```

### Tip

It is recommended to look at the demo application `EESPortPortConfig.py` in `<InstallationPath>\Demos\EESPort\Python` for more information.

## Related topics

### Basics

[Creating dSPACE EESPort Configuration Files..... 58](#)

### References

[dSPACE EESPort Configuration File \(dSPACE XIL API Reference !\[\]\(2bae76de5ebbd5c4d7d47162f1673734\_img.jpg\)](#))

## Latencies when Performing Electrical Error Simulation

### Introduction

Simulating electrical errors is limited by the latencies of the used simulation system.

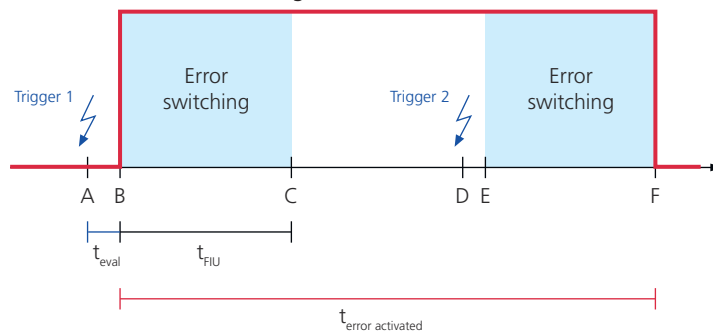
### Basics on latencies

The simulation system used for electrical error simulation consists of a host PC, a communication network and a failure simulation hardware. The physical limits of these components specify the latencies and limits for simulating electrical errors.

In addition to the switching behavior of the failure simulation hardware, i.e., the relays and semiconductor switches used, the latencies for electrical error simulation mainly depend on the host PC's performance and CPU usage and the communication processes between the host PC and the failure simulation hardware.

### Details on the switching behavior

The following simplified illustration shows the different latencies and delays that are relevant when simulating electrical errors.



The points A and D mark the times, when a manual trigger is set on the host PC or when the condition of a software trigger is fulfilled by the real-time application. The points B and E are the times, when the failure simulation hardware begins to switch or remove an error. In the points C and F the switching process of the failure simulation hardware is definitely finished.

The phases A to B ( $= t_{eval}$ ) and D to E are latencies caused by the host PC and the communication process with the failure simulation hardware. The phases B to C ( $= t_{FIU}$ ) and E to F define the switching processes on the failure simulation hardware. During these phases, the failure simulation hardware is in an estimated transition state and the error is switched physically. During the phase B to F ( $= t_{error\ activated}$ ), the original electrical signal may be disturbed either by the switching process or the switched error. Only during the phase C to E, the signal is definitely disturbed by the error.

The  $t_{error\ activated}$  factor indicates the physical limit for the duration of dynamic errors that can be simulated with a simulation system.

Not shown in the illustration are the times for (pre)configuring the failure simulation hardware (before points A and D) because during these phases the signal is not yet manipulated.

For the specific values for  $t_{eval}$ ,  $t_{FIU}$ , and  $t_{error\ activated}$  when using failure simulation hardware from dSPACE, refer to [Latencies when Performing Electrical Error Simulation \(dSPACE XIL API Reference !\[\]\(2e897e890e69d81eae4503a8342c36b0\_img.jpg\)\)](#).

## Related topics

### Basics

[Monitoring the Switching Behavior of Electrical Error Simulation Hardware..... 69](#)

### References

[Latencies when Performing Electrical Error Simulation \(dSPACE XIL API Reference !\[\]\(830769b31eeeaca920791081939ff8ba\_img.jpg\)\)](#)

# Monitoring the Switching Behavior of Electrical Error Simulation Hardware

## Introduction

When performing electrical error simulation, you can monitor the switching behavior of the electrical error simulation hardware, e.g., in ControlDesk.

## Variables for monitoring the switching behavior

**Using the default tracing variables** When you build the simulation application, the following tracing variables are generated into the related variable description file.

Variable Name	Purpose	Values
ActiveErrorSet	To display the number of the error set that is currently activated.	<ul style="list-style-type: none"> <li>0 = No error set is activated</li> <li>1 ... 500 = Number of the error set that is currently activated</li> </ul>
ErrorActivated	To indicate whether one or more errors are activated.	<ul style="list-style-type: none"> <li>0 = No error is activated</li> <li>1 = One or more errors are activated</li> </ul>
ErrorSwitching	To indicate the undefined transition state when switching the failure simulation hardware.	<ul style="list-style-type: none"> <li>0 = No switching is in process. The failure simulation hardware is in the steady state.</li> <li>1 = Switching is in process. The failure simulation hardware is in the transition state. The error is switched physically during this transition state.</li> </ul>
Flags	This variable is used internally. Do not use this variable for monitoring.	—
Trigger	This variable is used internally to detect software triggers.	—

These tracing variables for monitoring are located in the **XIL API/EESPort** subgroup. For multiprocessor models, the subgroup is generated into each submodel separately.

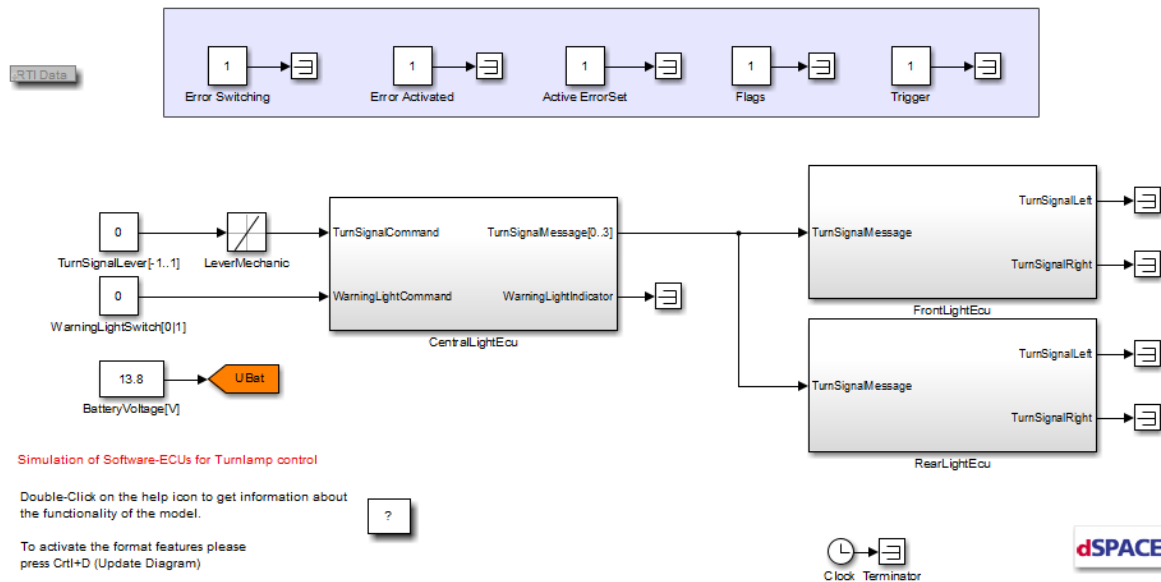
**Note**

- The tracing variables are not available for DS1104 platforms.
- If you execute multiple EESPorts in parallel, you must ensure that the tracing variables of the EESPorts have different name paths or variable names. Refer to [Using alternative variables](#).
- If you use the integrated SCALEXIO FIU in a SCALEXIO multi-processor system, you must use the tracing variables of the SCALEXIO processing unit that is specified to control the electrical error simulation. (Although the tracing variables are generated into each application process, they are only written on the processing unit that controls the electrical error simulation.)

**Tip**

In addition to the measurement variables described in this topic, variable description (TRC) files of SCALEXIO systems that use the integrated SCALEXIO FIU also contain the **Diagnostics/Failure Simulation** subgroup. This subgroup provides further variables for monitoring the switching behavior of the SCALEXIO failure simulation hardware. Refer to [Monitoring Additional Behavior of an Integrated SCALEXIO FIU \(ControlDesk Electrical Error Simulation via XIL API EESPort !\[\]\(950a62bbddad88d64435fd35607dfc42\_img.jpg\)](#)).

**Using alternative variables** If you want to use alternative variables to monitor the switching behavior of the failure simulation hardware, you have to prepare your Simulink model manually. It is recommended to add Simulink Constant blocks to the model to provide representations for the EESPort tracing variables in the **Model Root** subgroup of the variable description file. These Constant blocks must be terminated and must not have connections to other blocks that might modify their values. Refer to the following illustration.



When you specify the port configuration (PORTCONFIG) file for the EESPort, you have to replace the default tracing variables with your alternative variables.

Refer to the following example.

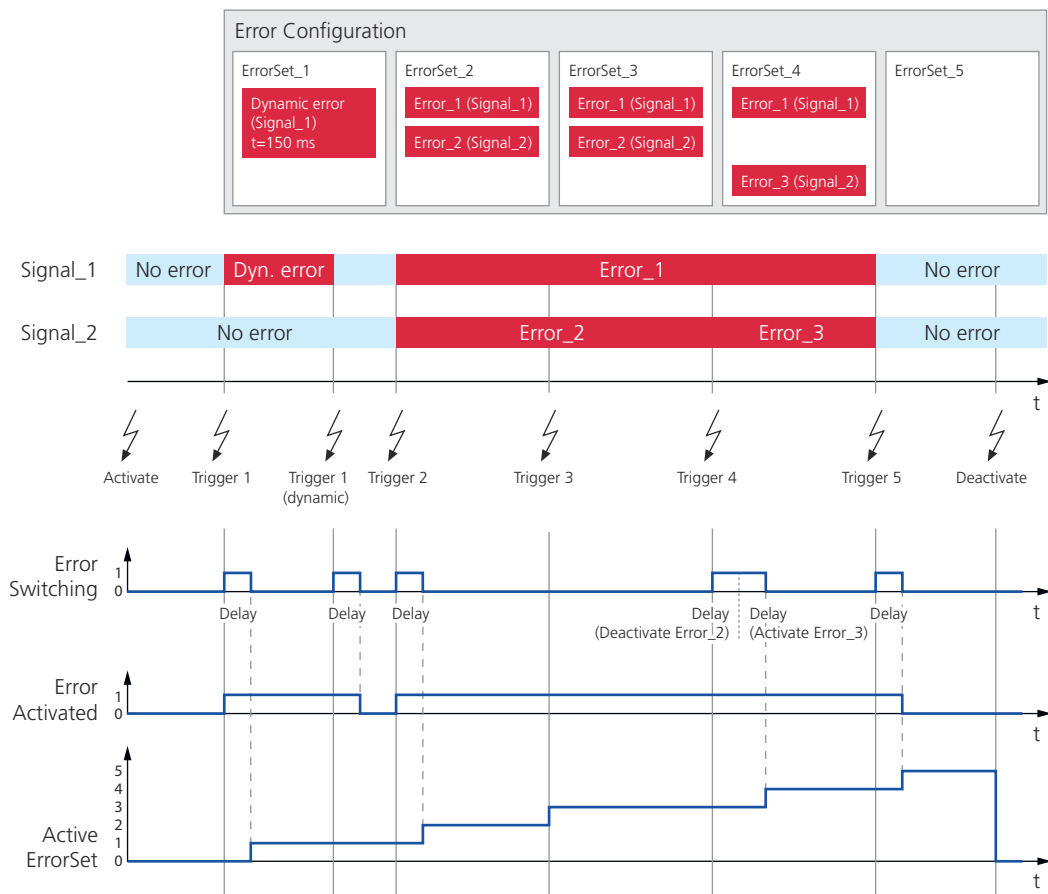
```
<RealTimeConfiguration PlatformName="SCALEXIO Real-Time PC" SystemDescriptionFilePath="../../Variable Descriptions/turnlamp.sdf/turnlamp.sdf">
  <Tracing Enabled="true">
    <Variable Value="Platform()://XIL API/EESPort/Error Activated" Type="ErrorActivated" />
    <Variable Value="Platform()://XIL API/EESPort/Active ErrorSet" Type="ActiveErrorSet" />
    <Variable Value="Platform()://XIL API/EESPort/Error Switching" Type="ErrorSwitching" />
    <Variable Value="Platform()://XIL API/EESPort/Flags" Type="Flags" />
    <Variable Value="Platform()://XIL API/EESPort/Trigger" Type="Trigger" />
  </Tracing>
</RealTimeConfiguration>
```

#### Note

If you perform electrical error simulation with an integrated SCALEXIO FIU, you can not specify alternative variables. You must use the default tracing variables.

#### Example

Switching the failure simulation hardware consumes time for communication and relay switching. The following example shows how the various measurement variables will display the estimated switching behavior of the failure simulation hardware in response to different commands (e.g., triggers) from the host PC where the XIL API EESPort is executed.



The example shows an error configuration with five error sets that disturb two signals with errors. The displayed switching behavior of the failure simulation hardware mainly depends on hardware-dependent delay times (e.g., for relay switching or communication) which can differ from each other.

### Configuring the monitoring

You can configure the monitoring in the EESPort configuration file. Usually, the defaults provided by the EESPortConfiguration API are applicable. You have to customize the configuration only, if you want to use mixed electrical error simulation hardware, or if you want to use existing variables in a real-time application with different variable names. For using mixed electrical error simulation hardware, you have to create two or more EESPort instances on the same platform. For more information, refer to [Creating dSPACE EESPort Configuration Files](#) on page 58.

### Related topics

#### Basics

[Latencies when Performing Electrical Error Simulation.....](#) 68



## dSPACE XIL API EESPort Demo

### Description of the C# demo project

The EESPort C# demo project shows you XIL API and its EESPort features in connection with dSPACE electrical error simulation hardware. As your simulator probably does not match the hardware configurations used for the demo project, you cannot use it to perform electrical error simulation, but you can gain an impression of how electrical error simulation works.

**Required licenses** Working with this demo requires:

- A license for the *XIL API EESPort*
- A license for *SCALEXIO Failure Simulation* (if you work with SCALEXIO failure simulation units)

### Demo project overview

In `<InstallationPath>\Demos\EESPort` you find three subfolders:

- C#
- CommonData
- Python

**C#** This folder contains the Visual Studio solution `EESPortExamples.sln`. If you open it, you find the following projects in the Solution Explorer.

- Basics  
Shows you how to implement the creation of an EESPort instance and the handling of error configuration sets created in the code.
- ErrorConfiguration  
Shows you how to implement the creation of different types of errors and how to save the error specifications to an error configuration file. The `Load` method of an error configuration file is also demonstrated.
- EESPortPortConfig  
Shows you how to create and modify port configurations.
- EESPortSignalsAndPotentials  
Shows you how to use dSPACE EESPort custom interfaces to get signals and potentials in an EESPort.
- EESPortPortConfigGenerator  
Shows you how to generate a port configuration file with default signal and potential mapping.
- SoftwareTrigger  
Shows you how to implement an error configuration with errors triggered by condition and by duration.

**CommonData** This folder contains some configuration files required for electrical error simulation.

- ErrorConfigurations  
This folder contains an example of an error configuration file in XML format according to the ASAM XIL API standard. It is a simple error configuration with four manually triggered error sets.

- PortConfigurations

This folder contains examples of signal lists and port configuration files for different dSPACE platforms. The signal lists are available in XLS and CSV format (using systems with discrete FIU) or they are included in the real-time application (using SCALEXIO systems). The port configuration files are in XML format.

- FullSizeVariant1

Provides data for a dSPACE Simulator Full-Size with DS291 FIU module.

- FullSizeVariant2

Provides data for a dSPACE Simulator Full-Size with DS293 FIU module.

- MidSizeBasedOnDS2210

Provides data for a dSPACE Simulator Mid-Size based on DS2210 with DS749 FIU module and DS789 FIU module. You find port configuration files for a simulator using one of the two FIU modules or both.

- MidSizeBasedOnDS2211

Provides data for a dSPACE Simulator Mid-Size based on DS2211 with DS791 FIU module and DS793 FIU module. You find a port configuration file for a simulator using both FIU modules.

- MidSizeBusBasedOnDS2211

Provides data for a dSPACE Simulator Mid-Size based on DS2211 with a DS1450 Bus FIU Board.

- SCALEXIO

Provides data for a SCALEXIO system with its integrated SCALEXIO FIU. The relevant information is included in the files generated in the build process.

The port configuration file differs from the files for systems with discrete FIU.

**Python** This folder contains the above-mentioned C# projects in the Python programming language. You can execute the Python scripts by using a Python Interpreter.

- Basics.py
- ErrorConfiguration.py
- EESPortPortConfig.py
- EESPortPortConfigGenerator.py
- EESPortSignalsAndPotentials.py
- SoftwareTrigger.py

# Basic Information on Simulating Electrical Errors in the Wiring

## Where to go from here

## Information in this section

<a href="#">Basics on Failure Simulation.....</a>	<a href="#">75</a>
Provides basic information on electrical error simulation using XIL API EESPort as electrical error simulation system.	
<a href="#">Hardware for Failure Simulation.....</a>	<a href="#">85</a>
Shows the hardware for failure simulation.	
<a href="#">Defining Failure Classes with Signal Files.....</a>	<a href="#">99</a>
For non-SCALEXIO systems, you define the failure classes allowed for failure simulation in the simulator's signal file.	
<a href="#">Creating Error Configurations.....</a>	<a href="#">108</a>
Provides information on creating an error configuration.	
<a href="#">Simulating Unstable Pin Failures (Loose Contacts) with DS793 Modules and SCALEXIO Systems.....</a>	<a href="#">118</a>
With a DS793 Sensor FIU or in a SCALEXIO system, you can simulate failures that are not permanent (pulsed switching), i.e., loose contacts or switch bouncing.	

## Basics on Failure Simulation

### Introduction

Provides basic information on electrical error simulation using XIL API EESPort as electrical error simulation system. In the hardware context, the *electrical error simulation* is called *failure simulation*.

## Where to go from here

## Information in this section

<a href="#">Basics on Failure Classes.....</a>	<a href="#">76</a>
Provides basic information on the kinds of failures which can be simulated on the wiring of an ECU pin.	
<a href="#">Electrical Error Simulation with a Discrete FIU.....</a>	<a href="#">76</a>
To perform electrical error simulation, a dSPACE hardware-in-the-loop (HIL) simulator can be extended by a discrete failure insertion unit (FIU).	
<a href="#">Electrical Error Simulation with the Integrated SCALEXIO FIU.....</a>	<a href="#">78</a>
To perform electrical error simulation, a SCALEXIO system can contain an integrated SCALEXIO failure insertion unit (FIU).	

Safety Precautions for Simulating Electrical Errors with a SCALEXIO System.....	83
Provides safety precautions for simulating electrical errors with a SCALEXIO system.	

## Basics on Failure Classes

### Introduction

By using a failure simulation system, you can simulate failures in the wiring of an electronic control unit (ECU). For example, you can simulate situations in which an ECU pin is shorted to ground or the battery voltage, or an ECU pin is not connected (cable break, open circuit).

The failures are simulated via failure insertion units (FIU). The FIUs contain relays or semiconductor switches which switch the signal from the ECU pin to simulate the failure. The failure types which can be simulated depend on the FIU in your simulator.

For information on which failure classes can be simulated, refer to [Failure Classes \(dSPACE XIL API Reference !\[\]\(c694a3ff3b077d76910920a6a1593ab4\_img.jpg\)](#)).

### Related topics

#### Basics

Electrical Error Simulation with a Discrete FIU.....	76
Electrical Error Simulation with the Integrated SCALEXIO FIU.....	78

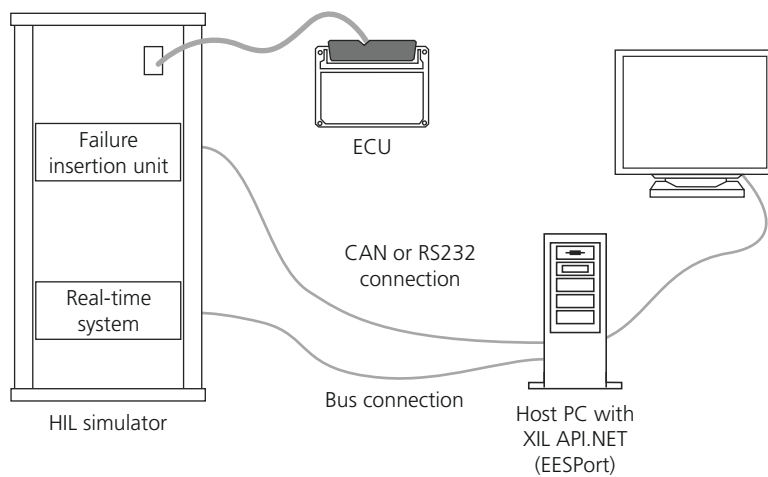
## Electrical Error Simulation with a Discrete FIU

### Introduction

To perform electrical error simulation, a dSPACE hardware-in-the-loop (HIL) simulator can be extended by a discrete failure insertion unit (FIU).

### Overview

The following illustration shows a HIL simulator that is extended by a discrete failure insertion unit (FIU). This schematic is also valid for a SCALEXIO system that does not use the concept of the integrated SCALEXIO failure insertion unit (FIU).



The important components of a failure simulation system are the failure insertion unit and the host PC with dSPACE XIL API .NET and its EESPort implementation.


**Failure insertion unit** The HIL simulator must be equipped with a discrete failure insertion unit (FIU). The FIU is connected via a CAN or RS232 serial connection, depending on its type. For an overview of the FIUs, refer to [Hardware for Failure Simulation](#) on page 85.

**Host PC with dSPACE XIL API .NET** The failure simulation is controlled via a host PC with dSPACE XIL API .NET and its EESPort implementation. It allows you to configure the errors for the ECU pins and finally switch the FIUs.

## Supported interfaces

The discrete failure insertion unit can be controlled by CAN interfaces and RS232 interfaces:

**CAN interfaces** dSPACE driver are installed with dSPACE XIL API .NET. Drivers for other CAN interfaces must be installed by the user.

For a complete overview on supported CAN hardware, refer to [dSPACE EESPort Configuration File](#) (dSPACE XIL API Reference .

**RS232 interfaces** COM ports 1 ... N of the host PC

#### Note

##### Reduced Performance by Using External RS232 Converters

You are strongly recommended to use a physical RS232 port of the host PC to control the failure simulation hardware. If external RS232 ports are missing, try to use an internal RS232 port of the host PC's motherboard. Software triggers and dynamic errors are not supported if you use an external RS232 converter. Communication via an external RS232 converter is also time-critical and can cause communication errors.

If there is no alternative to using an external RS232 converter:

- Use the IOLAN DS1 from Perle as an Ethernet-to-RS232 converter. For configuring this tool, refer to <http://www.dspace.com/go/eth2rs232>.
- Otherwise, use an USB-to-RS232 converter with an FTDI chipset and the newest FTDI driver, refer to <http://www.ftdichip.com/FTDrivers.htm>.

#### Signal file

For HIL simulators with discrete FIUs, the electrical errors (also named failure classes) allowed for electrical error simulation are specified in the *signal file* that comes with the simulator.

For more information, refer to [Defining Failure Classes with Signal Files](#) on page 99.

#### Related topics

##### Basics

<a href="#">Basics on Failure Classes.....</a>	<a href="#">76</a>
<a href="#">Electrical Error Simulation with the Integrated SCALEXIO FIU.....</a>	<a href="#">78</a>

##### References

<a href="#">Hardware for Failure Simulation.....</a>	<a href="#">85</a>
------------------------------------------------------	--------------------

## Electrical Error Simulation with the Integrated SCALEXIO FIU

#### Introduction

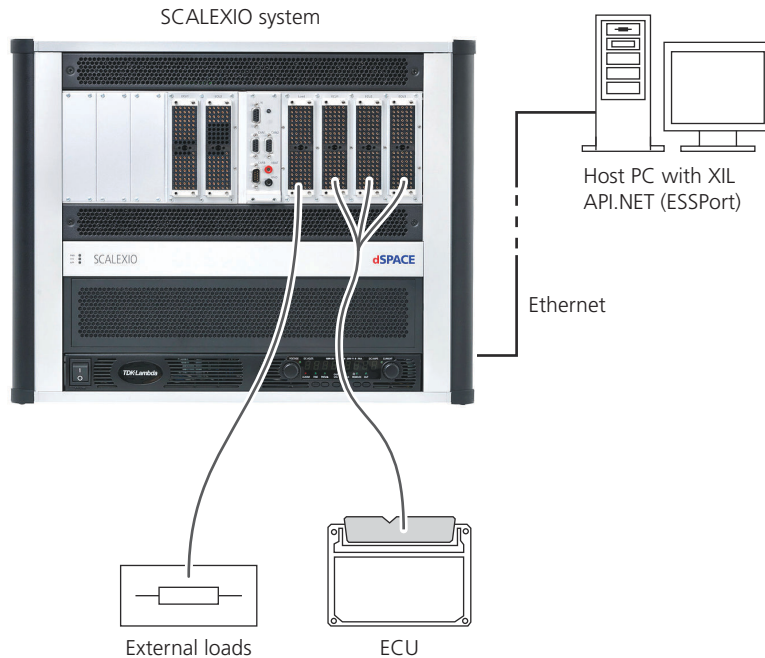
To perform electrical error simulation, a SCALEXIO system can contain an integrated SCALEXIO failure insertion unit (FIU).

## Overview

The important components for electrical error simulation are:

- A SCALEXIO failure insertion unit (FIU) that consists of several hardware components. Refer to Components of the integrated SCALEXIO FIU.
- A Host PC with dSPACE XIL API .NET and its EESPort implementation.

The following illustration shows an example of a SCALEXIO system that contains a SCALEXIO FIU. It is connected to an ECU, external loads, and the host PC.



## Required licenses

To simulate electrical errors, two kinds of licenses are required:

- A license for using the EESPort of dSPACE XIL API .NET
- A license for using the components of the SCALEXIO FIU

For more information, refer to [License Required for Electrical Error Simulation \(SCALEXIO – Hardware and Software Overview !\[\]\(3cb60d42b10e53f9522bb0b392c1c4cd\_img.jpg\)](#)).

## Components of the integrated SCALEXIO FIU

To perform electrical error simulation via the integrated SCALEXIO FIU, your SCALEXIO system must provide the required failure simulation hardware, which consists of the following components.

**Central failure insertion unit** The central failure insertion unit (central FIU), which is located, for example, on an FIU & power switch board or an I/O unit.

**Failure routing units** Individual failure routing units (FRUs), which are located on the signal and bus lines of the individual channels of SCALEXIO

MultiCompact I/O units and boards and HighFlex I/O boards. (Standard SCALEXIO I/O boards do not have any failure routing units.)

**Failrails** The failrails, which connect the central FIU to the individual failure routing units (FRUs) via the backplanes of the SCALEXIO slot units or I/O units.

**Failrail segment switches (optional)** One or more (optional) failrail segment switches, which are used for selectively connecting/disconnecting SCALEXIO slot or I/O units to/from the common failrails of the SCALEXIO system.

For more information, refer to [Electrical Error Simulation Concept \(SCALEXIO Hardware Installation and Configuration !\[\]\(6605b201d6f14d9b3bcb8ab5f274d107\_img.jpg\)](#)).

## Electrical errors

The integrated SCALEXIO failure simulation hardware lets you switch the following electrical errors:

- *Open circuit* to simulate a broken wire or loose contact
- *Short circuit*
  - To ground (KL31)
  - To power switch channels with battery voltage (KL15, KL30)
  - Between channels of the same signal category (e.g., a short circuit between signal measurement channels)
  - Between channels of different signal categories (e.g., a short circuit between signal measurement channels and signal generation channels)
  - Between signal measurement channels or signal generation channels and bus channels
- Pulsed switching for signals of signal measurement channels and signal generation channels (e.g., to simulate loose contacts or relay contact bouncing)

### Note

You cannot use pulsed switching for bus signals or when simulating multiple electrical errors.

Except for the *Analog In 2* and *Analog Out 2* channel types, switching electrical errors is performed only for the signal/bus lines, not for reference lines.

## Load or signal disconnection

In most cases, you can disconnect the loads or signals during failure simulation (*load rejection*). For example, to protect sensitive loads of signal measurement channels. Limitations apply only when you work with multi-pin failures (refer to [Switching short circuits between multiple signals and/or bus channels](#) on page 82).

Loads and signals can be disconnected by switching semiconductor switches or relays. For technical reasons, semiconductor switches cannot be used in all cases. Relays must switch the load or signal disconnection in the following cases:

- Short circuit between two signal generation or signal measurement channels



- Short circuit of a bus channel to any other channel type
- Multiple failures when activation by relay is necessary

**Note**

The relay might be carrying a current when you are using it to disconnect the loads or signals. Note the related warnings listed in [Safety Precautions for Simulating Electrical Errors with a SCALEXIO System](#) on page 83.

To switch a relay that might be carrying a current, you must set the corresponding property of the channel in ConfigurationDesk. Otherwise you cannot switch this kind of failure in combination with this channel.

**Pulsed switching**

The central FIU uses semiconductor switches for switching the failures. It is able to switch very fast (pulsed switching). This makes it possible to simulate loose contacts or defined switch bouncing. However, pulsed switching is not possible in all combinations of channel types.

Pulsed switching is not supported for

- Failures when bus channels are involved
- Multiple failures when activation by relay is necessary

**Switching multiple electrical errors**

The integrated SCALEXIO failure simulation hardware supports switching multiple electrical errors at the same time or in succession. For example, you can simulate an open circuit for one channel and a short circuit for another channel at the same time, without deactivating the first error.

**Note**

When multiple electrical errors are simulated, switching can be done by relays that are carrying a current. Note the related warnings listed in [Safety Precautions for Simulating Electrical Errors with a SCALEXIO System](#) on page 83.

**Allowed electrical errors** You can simulate the following electrical errors at the same time:

- Any number of open circuits (interrupts)
- Multiple short circuits to one power switch channel (short to battery voltage or another potential)
  - Up to 10 short circuits with a DS2642 FIU & Power Switch Board
  - Up to 6 short circuits with a DS2680 I/O Unit
- Multiple short circuits to ground (short to GND)
  - Up to 10 short circuits with a DS2642 FIU & Power Switch Board
  - Up to 6 short circuits with a DS2680 I/O Unit

- Multiple short circuits between signal measurement, signal generation and bus channels (short to pins, multi-pin errors (refer to [Switching short circuits between multiple signals and/or bus channels](#) on page 82))
  - Short circuits between up to 10 channels (one multi-pin error)
  - Two multi-pin errors in parallel ( $2 \times 10$  channels)
- Combinations of any number of open circuits (interrupts) with:
  - Multiple short circuits to one power switch channel (short to VBAT)
  - Multiple short circuits to ground (short to GND)
  - Multiple short circuits between channels (short to pins)
  - One error (short circuit to any potential or open circuit) of a signal that uses current enhancement
  - One error (short circuit to any potential or open circuit) with pulsed switching

#### Limitations

- In general, switching multiple electrical errors is limited by the allowed maximum current of the channels and failrails involved.
- Switching multiple electrical errors with channels that use current enhancement is not supported (except for channel multiplication of the Power Switch 1 channel type of the DS2642 FIU & Power Switch Board).
- Pulsed switching is supported for one signal only.

---

#### Switching short circuits between multiple signals and/or bus channels

**Multi-pin errors** Multi-pin errors let you simulate a short circuit between three or more signal channels and/or bus channels. The channels can be located on the same or different boards or I/O units. You can simulate a short circuit between:

- Channels of the same signal category (e.g., four signal generation channels)
- Channels of different signal categories (e.g., three signal generation channels and two signal measurement channels)
- Signal channels and bus channels (e.g., two signal generation channels, one signal measurement channel, and one bus channel)

**Switching multi-pin errors** The both failrails of a SCALEXIO system are used automatically by the XIL API EESPort according to the specified error configuration.

Multi-pin errors can be switched only by relays. Therefore, you must use ConfigurationDesk to allow the activation by FRU relays for each involved channel.

**Load or signal disconnection** You can disconnect loads or signals from channels that are used for multi-pin errors. To disconnect loads or signals, you must switch the channels to failrail 1. You cannot use failrail 2 in this case. Loads or signals can be disconnected only by relays.

**Note**

You can switch multi-pin errors and disconnect loads or signals by using relays that might be carrying a current. Note the related warnings listed in [Safety Precautions for Simulating Electrical Errors with a SCALEXIO System](#) on page 83.

**Monitoring the switching behavior**

With an experiment software such as ControlDesk, you can monitor and trace the switching behavior of the SCALEXIO failure simulation hardware. For further information, refer to [Monitoring the Switching Behavior of the Failure Simulation Hardware \(ControlDesk Electrical Error Simulation via XIL API EESPort !\[\]\(cbe80b694ebd74fcfe136a095b608235\_img.jpg\)\)](#) and [Monitoring the Switching Behavior of Electrical Error Simulation Hardware](#) on page 69.

**Related topics****Basics**

[Hardware for Electrical Error Simulation on SCALEXIO Systems..... 96](#)

## Safety Precautions for Simulating Electrical Errors with a SCALEXIO System

**Introduction**

You must consider some safety precautions to perform electrical error simulation with a SCALEXIO Systems.

**General warning****⚠ WARNING****Risk of unexpected high currents and voltages due to electrical error simulation**

During electrical error simulation, high currents and voltages might be present on board channels and/or connector pins, which is not expected. This can result in death, personal injury, fire, and/or damage to the SCALEXIO system and connected external devices.

**Note**

Especially signal measurement channels (such as channels connected in parallel or interconnected reference lines) can carry high currents.

You must ensure that no voltages or currents outside the specified ranges of the I/O channels can occur during electrical error simulation.

## Switching FRU relays

Using dSPACE XIL API EESPort, switching is done with current load by default.

If FRU relays are switched for electrical error simulation, note the following warnings.

### NOTICE

#### **Risk of increased wear and permanent damage to the relays of the integrated SCALEXIO failure simulation hardware**

To a varying extent, depending on which loads are connected and which currents and voltages are switched, electric arcs and contact erosion can occur in the FRU relays involved. This will eventually destroy the FRU relays. The board or unit on which they are mounted will no longer be usable and will have to be repaired or replaced.

Operating the relays outside the permitted range (i.e., above the maximum switching capacity) can also destroy the relays and will probably damage the board or unit on which they are mounted.

Before using FRU relays for electrical error simulation, you must fulfill the following preconditions:

- To minimize the risk of damage in a multiple error scenario, operate the relays only within the permitted conditions and ranges (i.e., under the maximum switching capacity). For concrete values, refer to [FRU Relays Data Sheet \(SCALEXIO Hardware Installation and Configuration !\[\]\(e6ddc77b791299d975007937cebef274\_img.jpg\)](#)).
- If the connected loads are inductive and you want to simulate electrical errors in their wiring, you must protect the dSPACE hardware from induced high voltages. Refer to [Safety Precautions for Using Inductive Loads \(SCALEXIO Hardware Installation and Configuration !\[\]\(ab52e27d061d76db54e182891376cff5\_img.jpg\)](#)).
- Consider the increased risk of material wear or damage before you set Activation by FRU relay to Allowed in ConfigurationDesk.

Note that defects caused by material wear, misuse or operation outside the permitted ranges are not covered by any warranty, and no liability is accepted by dSPACE for any direct or indirect damage arising from such defects.

### NOTICE

#### **Risk of damage to a connected load**

When FRU relays are switched for electrical error simulation, load rejection can be delayed up to 30 ms due to the switching times of the relays involved. A connected load can be damaged during these 30 ms.

# Hardware for Failure Simulation

## Where to go from here

## Information in this section

Overview of the Failure Simulation Hardware.....	85
DS291 FIU Module.....	86
DS293 FIU Module, DS282 Load Module and DS289MK RSim Module.....	87
DS749 FIU Module.....	90
DS789 Sensor FIU Module.....	90
DS791 Actuator FIU Module.....	92
DS793 Sensor FIU.....	93
DS5355/DS5390 High Current FIU System.....	95
Hardware for Electrical Error Simulation on SCALEXIO Systems.....	96
To perform electrical error simulation, your SCALEXIO system must provide the required failure simulation hardware and be operated by a SCALEXIO Processing Unit.	

## Information in other sections

[DS1450 Bus FIU Board \(PHS Bus System Hardware Reference !\[\]\(0aff635c4179ba9e710b00f4b01d3b20\_img.jpg\)](#))

## Overview of the Failure Simulation Hardware

### Introduction

The EESPort controls the hardware for failure simulation. The following tables show the supported failure simulation systems and their control interfaces according to the simulators they can be installed in.

#### Failure Simulation Hardware for dSPACE Simulator Full-Size

Failure Simulation Hardware	Control Interface
n x DS282 Load Module	CAN
1 x DS293 FIU Module	CAN
1 x DS289MK RSim Module	CAN
n x DS291 FIU Module	RS232
n x DS5355 High Current FIU Controller Card with DS5390 High Current FIU	RS232

**Failure Simulation Hardware  
for dSPACE Simulator Mid-  
Size based on DS2210**

Failure Simulation Hardware	Control Interface
n x DS749 FIU Module	RS232
n x DS789 Sensor FIU Module	RS232
n x DS1450 Bus FIU Board <sup>1)</sup>	RS232

<sup>1)</sup> For more information on the DS1450, refer to [DS1450 Bus FIU Board \(PHS Bus System Hardware Reference !\[\]\(eafc244b53721dd1ec133f0772f70fc7\_img.jpg\)](#)).

**Failure Simulation Hardware  
for dSPACE Simulator Mid-  
Size based on DS2211**

Failure Simulation Hardware	Control Interface
n x DS791 Actuator FIU Module	RS232
n x DS793 Sensor FIU Module	RS232
n x DS1450 Bus FIU Board <sup>1)</sup>	RS232

<sup>1)</sup> For more information on the DS1450, refer to [DS1450 Bus FIU Board \(PHS Bus System Hardware Reference !\[\]\(5a132f13505a6571904d622757b7a8f0\_img.jpg\)](#)).

**Failure Simulation Hardware  
with dSPACE HIL systems  
based on expansion boxes**

Failure Simulation Hardware	Control Interface
n x DS1450 Bus FIU Board <sup>1)</sup>	RS232

<sup>1)</sup> For more information on the DS1450, refer to [DS1450 Bus FIU Board \(PHS Bus System Hardware Reference !\[\]\(73002692dd5e7a64e60946be3158e719\_img.jpg\)](#)).

**Failure Simulation Hardware  
of SCALEXIO Systems**

Failure Simulation Hardware	Control Interface
Internal central failure insertion unit and failure routing units	Ethernet

**Connecting to the RS232  
interface**


To connect the failure simulation hardware to the host PC via RS232 interface, you have to use a standard cable.

Do not use a crossover cable.

## DS291 FIU Module

**Introduction**

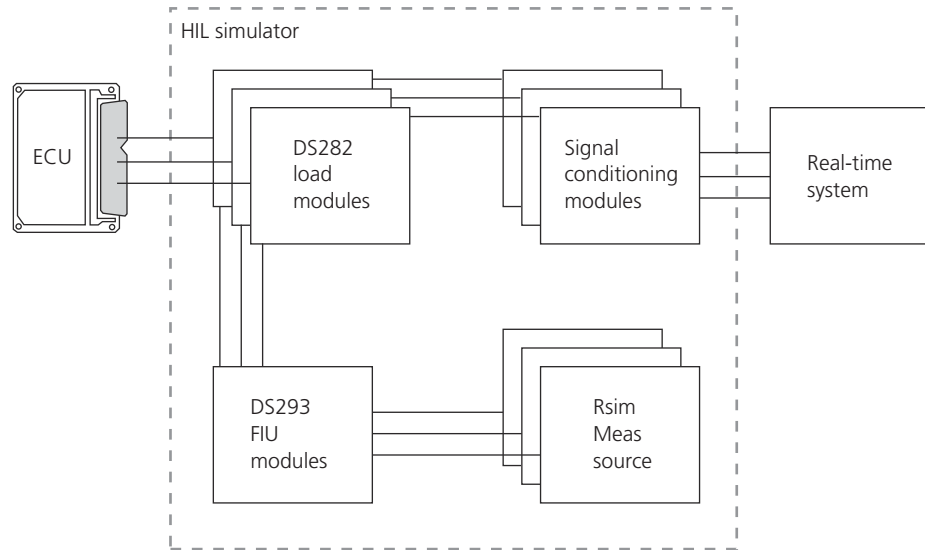
A DS291 FIU Module is a board which can be used for failure simulation.

<b>Pin failures simulated</b>	<p>The pin failures which can be simulated are:</p> <ul style="list-style-type: none"> <li>▪ Cable break</li> <li>▪ Short circuit to ground or minus terminal of the battery voltage (KL31) with or without load (an ECU input is disconnected from a dSPACE output channel)</li> <li>▪ Short circuit to plus terminal of the battery voltage (KL30) with or without load (an ECU input is disconnected from a dSPACE output channel)</li> <li>▪ Short circuit to another ECU pin with or without load</li> </ul>
<b>Features</b>	<p>The DS291 has the following features:</p> <ul style="list-style-type: none"> <li>▪ Controlled via RS232 interface</li> <li>▪ Several can be installed in dSPACE Simulator Full-Size</li> <li>▪ Each board contains 10 channels.</li> </ul>
<b>Description</b>	For more information on the board, refer to the module's hardware documentation.
<b>Related topics</b>	<p><b>References</b></p> <p><a href="#">Failure Classes for the DS291 FIU Module (dSPACE XIL API Reference </a>)</p>

## DS293 FIU Module, DS282 Load Module and DS289MK RSim Module

<b>Introduction</b>	A DS282 Load Module, a DS293 FIU Module and a DS289MK RSim Module form a system for failure simulation.
<b>Pin failures simulated</b>	<p>The pin failures which can be simulated are:</p> <ul style="list-style-type: none"> <li>▪ Open circuit with or without additional hardware in series (RSim)</li> <li>▪ Short circuit to another ECU pin directly or via additional hardware elements (RSim)</li> <li>▪ Short circuit to 5 reference points (potential 0 ... 4) directly or via additional hardware elements (RSim)</li> </ul>

The following illustration shows an overview of the boards:



## Features

All modules have the following features:

- Controlled via CAN interface
- Several DS282 Load Modules can be connected to a DS293 that performs the failures.
- Only one DS293 can be used in dSPACE Simulator Full-Size.

## DS282 features

A DS282 has the following features (the brackets contain the names of the elements in the schematic below):

- 10 channels
- Connects ECU input and output pins to the dSPACE hardware (ECU output pins optionally via loads)
- 3 relays to route the signal to the DS293 (RELn\_0, RELn\_1, RELn\_3 in schematic below) via 3 rails
- 1 relay to perform an open circuit (RELn\_2)

## DS293 features

A DS293 has the following features:

- 3 rails to connect DS282 boards (RAIL0 ... RAIL2 in schematic below)
- 5 connectors to connect reference voltages (Pot 0 ... Pot 4), for example, terminal of battery voltage, ECU sensor grounds, ECU sensor supplies. The setup depends on the project.
- 1 connector for the DS289MK (Rsim)

## DS289MK features

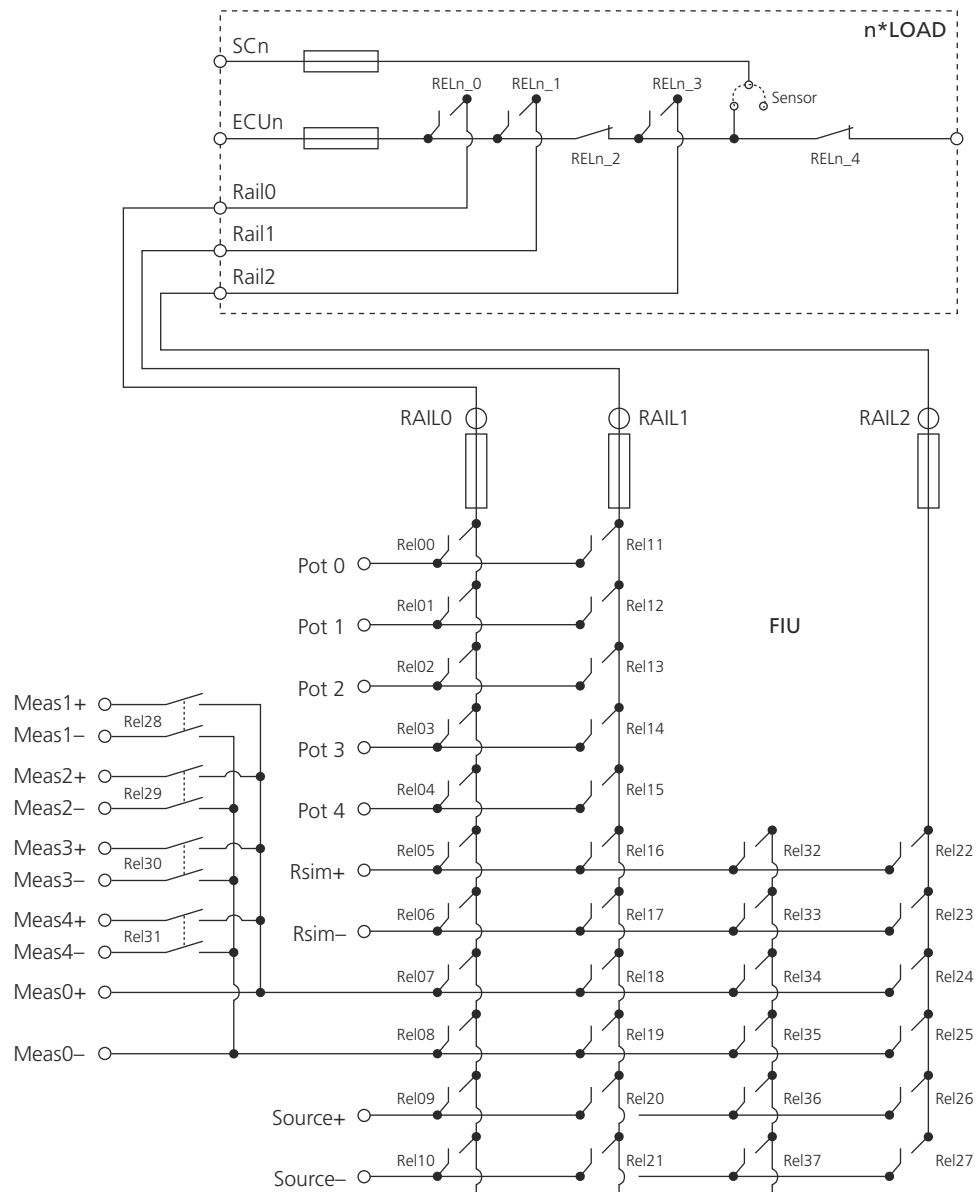
A DS289MK has the following features:

- Simulates a resistance in the range 1 ... 131071  $\Omega$  in steps of 1  $\Omega$



**Description**

When a load module and a DS293 are connected, there are a lot more options for failure simulation. Failure simulation can be done not only via the DS293, but also via the load modules. The following illustration shows a simplified schematic of a DS293 with one connected load module. ECU<sub>n</sub> is the connector for the ECU pin, SC<sub>n</sub> is the connector for the signal conditioning. For more information on the boards, refer to the modules' hardware documentation.



**Related topics****References**[Failure Classes for the DS293 FIU Module \(dSPACE XIL API Reference !\[\]\(99f58673407353e96a019fbca558fd72\_img.jpg\)\)](#)

## DS749 FIU Module

**Introduction**

A DS749 FIU Module is a board which can be used for failure simulation.

**Pin failures simulated**

The pin failures which can be simulated are:

- Cable break
- Short circuit to ground
- Short circuit to battery voltage

**Features**

The DS749 has the following features:

- Controlled via RS232 interface
- Several can be installed in dSPACE Simulator Mid-Size
- Each board contains 10 channels.

**Description**

For more information on the board, refer to the module's hardware documentation.

**Related topics****References**[Failure Classes for the DS749 FIU Module \(dSPACE XIL API Reference !\[\]\(eabd9f9ababee93effadc3b380fe65fd\_img.jpg\)\)](#)

## DS789 Sensor FIU Module

**Introduction**

A DS789 Sensor FIU Module is a board which can be used for simulating failures on ECU inputs.

**Pin failures simulated**

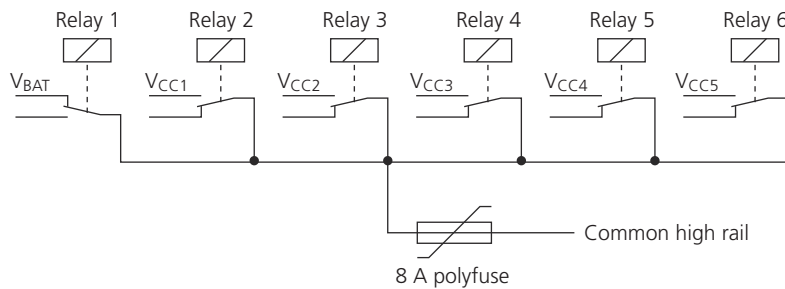
The pin failures which can be simulated are:

- Cable break
- Short circuit to ground
- Short circuit to a common high rail

The common high rail can be connected to the battery voltage (default) or 5 other potentials (see below).

**Common high rail**

The common high rail is typically connected to sensor supply voltages of the ECU for simulating short circuits from a sensor to its supply voltage.



By default, the  $V_{CC}$  potentials are wired to ADC pins as described in the table below.

Vcc Potential	ADC Pin
Vcc1	ADC8
Vcc2	ADC9
Vcc3	ADC10
Vcc4	ADC11
Vcc5	ADC12

If the wiring is not appropriate, contact dSPACE. The wiring can only be changed by dSPACE.

**Features**

The DS789 has the following features:

- Controlled via RS232 interface
- Directly connected to the ECU 1 connector (connector for ECU inputs) of a dSPACE Simulator Mid-Size (based on DS2210)
- Each module contains 81 channels connected to the following signals:
  - DAC1 ... DAC11, /DAC1 ... /DAC11
  - RES1- ... RES6-, RES1+ ... RES6+
  - WAVE0- ... WAVE3-, WAVE0+ ... WAVE3+
  - CRANK-, CRANK+, DIGCRANK
  - CAM1-, CAM1+, DIGCAM1, CAM2-, CAM2+, DIGCAM2
  - DIGOUT1 ... DIGOUT16

- PWMOUT1 ... PWMOUT6
- CAN1H, CAN1L, CAN2H, CAN2L
- RXD-, RXD+, TXD-, TXD+
- Each FIU channel has an additional serial resistance of max. 6.5  $\Omega$  (resistive outputs have an additional resistance of 13  $\Omega$ ).

**Description**

For more information on the board, refer to the module's hardware documentation.

**Related topics****References**

[Failure Classes for the DS789 Sensor FIU Module \(dSPACE XIL API Reference !\[\]\(c694a3ff3b077d76910920a6a1593ab4\_img.jpg\)\)](#)

## DS791 Actuator FIU Module

**Introduction**

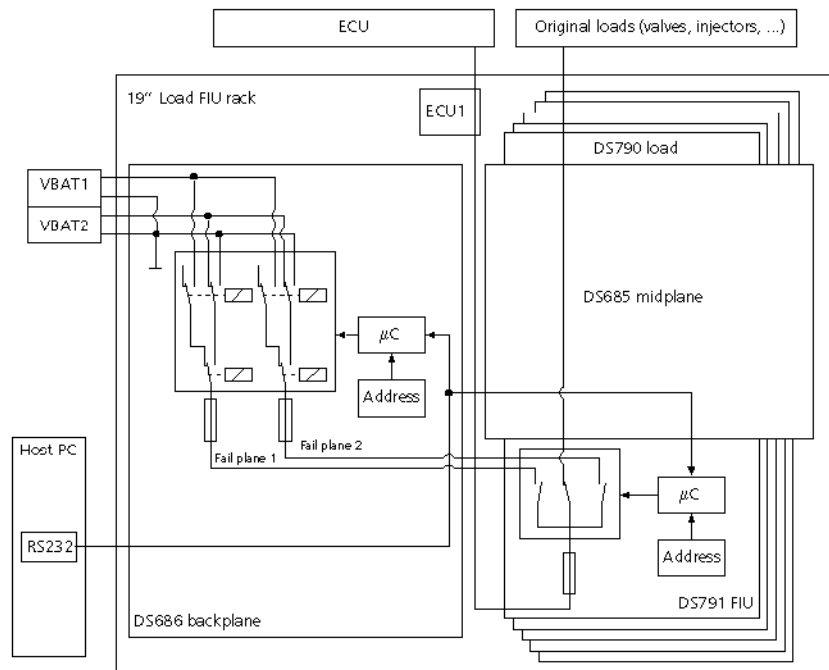
A DS791 Actuator FIU Module is a board which can be used for simulating failures on ECU outputs.

**Pin failures simulated**

The pin failures which can be simulated are:

- Cable break
- Short circuit to two fail planes

Each fail plane can be connected to VBAT1, VBAT2 or ground via the FIU multiplexer of the DS686 Backplane to simulate a failure to the corresponding potential. If a fail plane is not connected to a potential, you can simulate a short circuit between two or more pins.



## Features

The DS791 has the following features:


- Controlled via RS232 interface
- Several can be installed in dSPACE Simulator Mid-Size based on DS2211 (the standard version has 5 cards)
- Each module contains 10 channels.

### Description

For more information on the board, refer to [Failure Simulation for ECU Outputs \(dSPACE Simulator Mid-Size Based on DS2211 Features\)](#).

## Related topics

## References

Failure Classes for the DS791 Actuator FIU Module and the DS793 Sensor FIU Module (dSPACE XIL API Reference )

## DS793 Sensor FIU

## Introduction

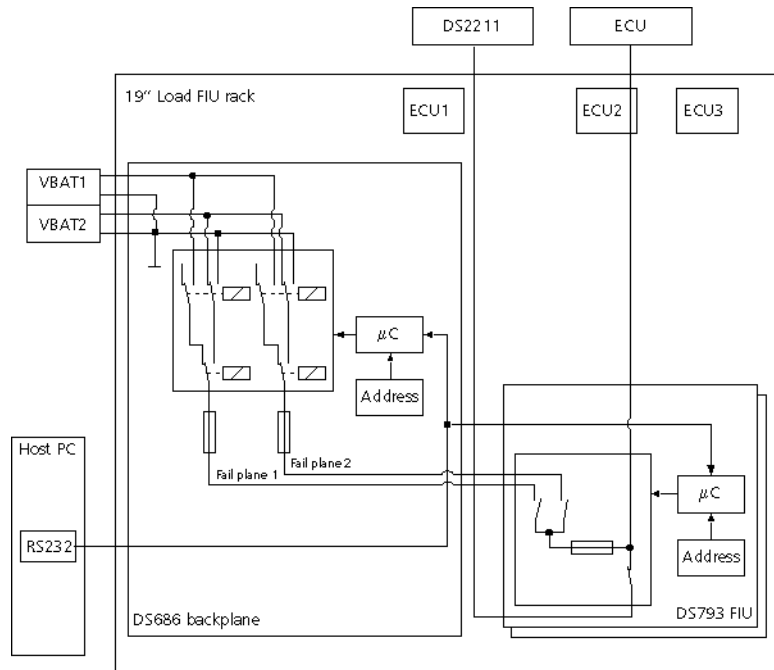
A DS793 Sensor FIU Module is a board which can be used for simulating failures on ECU inputs.

**Pin failures simulated**

The pin failures which can be simulated are:

- Cable break
- Short circuit to two fail planes

Each fail plane can be connected to VBAT1, VBAT2 or ground via the FIU multiplexer of the DS686 Backplane to simulate a failure to the corresponding potential. If a fail plane is not connected to a potential, you can simulate a short circuit between two or more pins.

**Features**

The DS793 has the following features:

- Controlled via RS232 interface
- Can simulate a loose contact or contact bounce (a pin is connected to the fail plane for specified time durations)
- Several can be installed in dSPACE Simulator Mid-Size based on DS2211 (2 cards are required for all ECU inputs of a standard simulator)
- Each DS793 module can contain a maximum of 81 channels. (The DS793 module is a base board for up to nine DS794 modules. Each DS794 module has nine channels.)
- The DS793 and DS791 modules have the same control. If you want to control a DS793, you must specify DS791 as the board type in the signal list.

**Description**

For more information on the board, refer to [DS793 Sensor FIU \(dSPACE Simulator Mid-Size Based on DS2211 Features\)](#).

**Related topics****Basics**

[Simulating Unstable Pin Failures \(Loose Contacts\) with DS793 Modules and SCALEXIO Systems..... 118](#)

**References**

[Failure Classes for the DS791 Actuator FIU Module and the DS793 Sensor FIU Module \(dSPACE XIL API Reference !\[\]\(83f22ed94ec5517769dd76d702c6bfd8\_img.jpg\)\)](#)

## DS5355/DS5390 High Current FIU System

**Introduction**

A FIU system based on a DS5355 High Current FIU Controller Card and a DS5390 High Current FIU board can be used for failure simulation.

**Pin failures simulated**

The pin failures which can be simulated are:

- Cable break
- Short circuit to ground or minus terminal of the battery voltage (KL31) with or without load (an ECU input is disconnected from a dSPACE output channel)
- Short circuit to plus terminal of the battery voltage (KL30) with or without load (an ECU input is disconnected from a dSPACE output channel)
- Short circuit to another ECU pin with or without load

**Features**

The DS5355/DS5390 high current FIU system has the following features:

- Controlled via RS232 interface
- Several can be installed in dSPACE Simulator Full-Size
- Each DS5390 High Current FIU contains 10 channels.

**Description**

For more information on the board, refer to the board's hardware documentation.

**Related topics****References**

[Failure Classes for the DS5355/DS5390 High Current FIU System \(dSPACE XIL API Reference !\[\]\(8aa05b4b06c05d58ddd90cdbf335b307\_img.jpg\)\)](#)

## Hardware for Electrical Error Simulation on SCALEXIO Systems

---

### Introduction

Provides basic information on the failure simulation hardware of SCALEXIO systems.

---

### Electrical error simulation concept of a SCALEXIO system

To simulate failures (electrical errors) in an ECU wiring, the following hardware components are required in a SCALEXIO system:

**Central failure insertion unit** A central failure insertion unit (central FIU), which is located, for example, on an DS2642 FIU & Power Switch Board or a DS2680 I/O Unit. Refer to [Central FIU](#) on page 97.

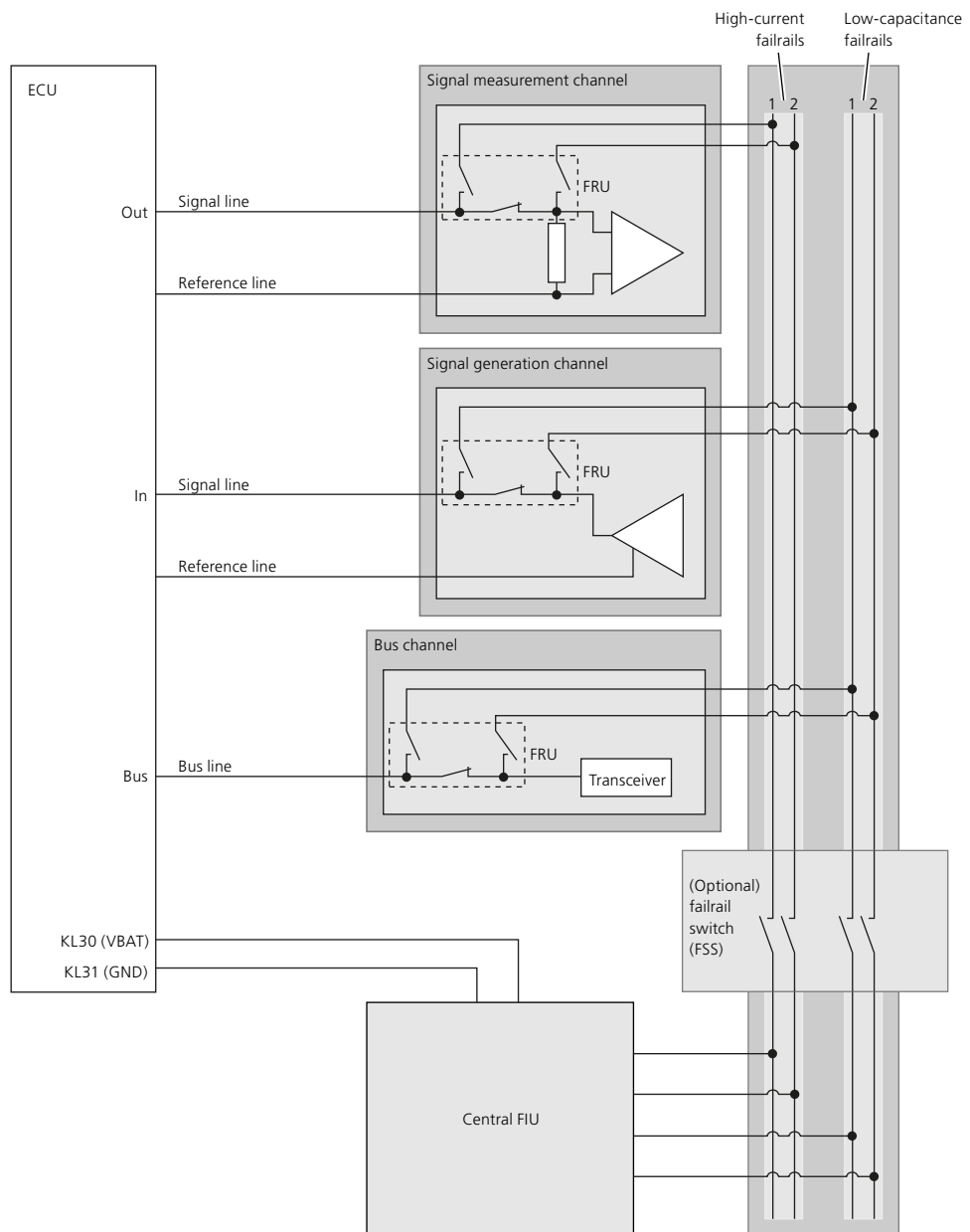
**Failure routing units** Single failure routing units (FRUs), which are located, for example, in the signal or bus lines of the single channels. Refer to [Failure routing units](#) on page 98.

**Failrails** The failrails which connect the central FIU to the single failure routing units (FRUs) via the backplanes of the SCALEXIO slot or I/O units. Refer to [Failrails](#) on page 98.

**Failrail segment switches (optional)** One or more (optional) failrail segment switches which are used for selectively connecting/disconnecting SCALEXIO slot or I/O units to/from the common failrails of the SCALEXIO system. Refer to [Failrail segment switches](#) on page 98.

The following illustration shows the hardware components that are required for electrical error simulation.





**SCALEXIO Processing Unit** The SCALEXIO system must consist of a SCALEXIO rack and be operated by a SCALEXIO Processing Unit. (A SCALEXIO system with SCALEXIO LabBox and a DS6001 Processor Board does not support electrical error simulation.)

### Central FIU

The *central FIU* is the major component of the SCALEXIO failure simulation hardware.

Usually, the central FIU activates a single electrical failure which is preconfigured by a channel's *failure routing unit* (FRU). For failure simulation on a bus channel

and for simulating multiple failures, the central FIU does not activate a failure but only preconfigures it. The failure is activated by the FRU(s) of the channel(s) involved.

The central FIU is connected to the single FRUs via the *failrails* located on the backplane of a SCALEXIO slot unit or I/O unit. The central FIU is located on a DS2642 FIU & Power Switch Board or a DS2680 I/O Unit.

The central FIU has semiconductor switches. In comparison to the relays used for the FRUs, these switches are able to switch very fast without bouncing (pulsed switching). It is therefore possible to simulate loose contacts or defined switch bouncing with the semiconductor switches of the central FIU.

---

### Failrails

The *failrails* connect the single *failure routing units* (FRUs) to the *central FIU*. As signals for signal measurement and signal generation have different requirements, a SCALEXIO system can have different failrails:

- *Low-capacitance failrails* to connect the FRUs of signal generation channels and bus channels to the central FIU. These failrails have a low capacitance related to signal ground (KL 31) to disturb the signals as little as possible.
- *High-current failrails* to connect the FRUs of signal measurement channels to the central FIU. These failrails are able to carry a high current.

---

### Failrail segment switches

The SCALEXIO failrail system can contain optional *failrail segment switches*. These switches are used for selectively connecting/disconnecting slot units and I/O units to/from the common failrails of the SCALEXIO system. Failrail segment switches can also switch the inter-rack connections of the failrail system in a multi-rack SCALEXIO system.

The selective disconnection of large sections of the failrails by using failrail segment switches minimizes system-inherent parasitic effects (e.g., parasitic capacitances of semiconductor switches).

---

### Failure routing units

For SCALEXIO HighFlex I/O boards and MultiCompact I/O units and boards, each signal channel and each bus channel provide one or more *failure routing units* (FRUs) to connect the channel to the failrails. (Standard SCALEXIO I/O boards do not have failure routing units and do not support electrical error simulation.)

An FRU has *FRU relays* for switching. That is why FRUs cannot be used to simulate pulsed switching. Switching an FRU relay can be accompanied by contact chatter.

The FRU relays are used and switched as follows:

- *To simulate an electrical error of a single signal line*, the FRU relays preconfigure the error first. The error is then actually activated by the semiconductor switches of the central FIU.

If parallel channels are used, for example, for current enhancement, they are switched synchronously.

- *To simulate an electrical error of a single bus line*, the semiconductor switches of the central FIU do not activate the error but only preconfigure it. The error is activated by the FRU relays, because they disturb the bus signals as little as possible.
- *To simulate multiple electrical errors*, the first of the errors – except for an error in a bus line – is switched the same way as when switching a single error. This means that the first error is preconfigured by switching the channel's FRU relays first and then actually activated by the semiconductor switches of the central FIU. Depending on the selected error type, the second and all other errors can be activated by the FRU relays without preconfiguring the error.
- *To disconnect the loads or signals* during electrical error simulation (load rejection). When simulating errors:
  - A load can be disconnected from a signal measurement channel.
  - A generated signal can be disconnected from a signal generation or bus channel.


#### Short-circuit detection units

Software-controlled short-circuit detection units are implemented in the connection between each power switch channel and the central FIU. They protect the semiconductors of the central FIU against overcurrent if the fuses implemented in the signal measurement, signal generation, and/or bus channels react too slowly.

If the signal measurement, signal generation, or bus board detects an overcurrent, the semiconductor of the central FIU are opened. The status LEDs of the boards display the error state, and the short-circuit detection units report the overcurrent event to the SCALEXIO Processing Unit.

#### Related topics

##### Basics

Basics on Electrical Error Simulation with SCALEXIO Systems (SCALEXIO Hardware Installation and Configuration )  
 Electrical Error Simulation with the Integrated SCALEXIO FIU..... 78

## Defining Failure Classes with Signal Files

#### Introduction

For non-SCALEXIO systems, you define the failure classes allowed for failure simulation in the simulator's signal file.

**Where to go from here****Information in this section**

<a href="#">Structure of Signal Files.....</a>	<a href="#">100</a>
The signal file contains the wiring information of a non-SCALEXIO system and is part of its standard documentation.	
<a href="#">Comment Area.....</a>	<a href="#">101</a>
The comment area is the part of a signal file which contains comments.	
<a href="#">Format Area.....</a>	<a href="#">102</a>
The format area is the part of a signal file which contains attributes for it.	
<a href="#">Signallist Area.....</a>	<a href="#">103</a>
The signallist area is the part of a signal file which contains the information on the signals and their wiring.	
<a href="#">How to Configure Signal Files.....</a>	<a href="#">106</a>
Before you start using a failure simulation system, you can adapt the signal file to your needs.	
<a href="#">Evaluating Signal Files.....</a>	<a href="#">107</a>
When a signal file is read, its contents are automatically evaluated to set the pins in the failure simulation system. This knowledge is important if some pins are missing in the failure simulation system after a signal file is read.	

## Structure of Signal Files

**Introduction**

The signal file contains the wiring information of a non-SCALEXIO system and is part of its standard documentation.

Normally, dSPACE generates this file when designing the simulator. It is provided in Excel format and must be converted to a CSV file.

**Areas of the signal file**

The signal file is divided into three areas:

- /Comment ... /CommentEnd, see [Comment Area](#) on page 101
- /Format ... /FormatEnd, see [Format Area](#) on page 102
- /Signallist ... /SignallistEnd, see [Signallist Area](#) on page 103

**Example**

The following illustration shows an example of a signal file:

/Comment																						
/CommentEnd																						
/Format																						
Version	1.0																					
BoardType	DS0791																					
Potential0	VBAT1																					
Potential1	VBAT2																					
Potential2	GND																					
FIUMUXAddress	0x0B																					
AdditionalFiles	none																					
FaultClassFile	none																					
/FormatEnd																						
/SignalList																						
	ECU Description						Simulator Description															
IO_Index	ECU_Name()	PIN_Number()	PIN_Name()	Signal_Description()	Faults_ECU()	ASPACE_IO_channel	ASPACE_Board	Loadboard_Channel	Load_Resistance	Maximal Load Power	Load connected to	FIU_Module_ID	FIU_Module_Channel	FIU_Type	GND_Jumper_State	Disconnect_Jumper	HYP_Connector	HYP_Connector Pin	CARB_Connector Pin			
1	#	#	#	#	-	CRANK+	DS2211_1	-	-	-	-	-	6	DS793	-	-	-	HYP2	D13	-		
2	#	#	#	#	-	CRANK-	DS2211_1	-	-	-	-	-	5	DS793	-	-	-	HYP2	E13	-		
3	#	#	#	#	-	CRANK_DIG	DS2211_1	-	-	-	-	-	5	DS793	-	-	-	HYP2	C13	-		
4	#	#	#	#	-	CAM1+	DS2211_1	-	-	-	-	-	8	DS793	-	-	-	HYP2	A14	-		
5	#	#	#	#	-	CAM1-	DS2211_1	-	-	-	-	-	7	DS793	-	-	-	HYP2	B14	-		
6	#	#	#	#	-	CAM2+	DS2211_1	-	-	-	-	-	10	DS793	-	-	-	HYP2	D14	-		
7	#	#	#	#	-	CAM2-	DS2211_1	-	-	-	-	-	9	DS793	-	-	-	HYP2	E14	-		
8	#	#	#	#	-	CAM1_DIG	DS2211_1	-	-	-	-	-	3	DS793	-	-	-	HYP2	C14	-		
9	#	#	#	#	-	CAM2_DIG	DS2211_1	-	-	-	-	-	4	DS793	-	-	-	HYP2	A15	-		
10	#	#	#	#	-	DSP0+	DS2211_1	-	-	-	-	-	8	DS793	-	-	-	HYP2	B15	-		
11	#	#	#	#	-	DSP0-	DS2211_1	-	-	-	-	-	7	DS793	-	-	-	HYP2	C15	-		
12	#	#	#	#	-	DSP1+	DS2211_1	-	-	-	-	-	10	DS793	-	-	-	HYP2	B16	-		
13	#	#	#	#	-	DSP1-	DS2211_1	-	-	-	-	-	9	DS793	-	-	-	HYP2	C16	-		
14	#	#	#	#	-	DSP2+	DS2211_1	-	-	-	-	-	2	DS793	-	-	-	HYP2	D16	-		
15	#	#	#	#	-	DSP2-	DS2211_1	-	-	-	-	-	1	DS793	-	-	-	HYP2	E16	-		
16	#	#	#	#	-	DSP3+	DS2211_1	-	-	-	-	-	4	DS793	-	-	-	HYP2	A17	-		
17	#	#	#	#	-	DSP3-	DS2211_1	-	-	-	-	-	3	DS793	-	-	-	HYP2	B17	-		
18	#	#	#	#	-	DSP4	DS2211_1	-	-	-	-	-	9	DS793	-	-	-	HYP3	B15	-		
19	#	#	#	#	-	DSP5	DS2211_1	-	-	-	-	-	10	DS793	-	-	-	HYP3	C15	-		
20	#	#	#	#	-	DSP6	DS2211_1	-	-	-	-	-	1	DS793	-	-	-	HYP3	B16	-		
21	#	#	#	#	-	DSP7	DS2211_1	-	-	-	-	-	2	DS793	-	-	-	HYP3	C16	-		
22	#	#	#	#	-	CRANK+	DS2211_2	-	-	-	-	-	6	DS793	-	-	-	HYP5	D13	-		
23	#	#	#	#	-	CRANK-	DS2211_2	-	-	-	-	-	5	DS793	-	-	-	HYP5	E13	-		
/SignalListEnd																						

## Comment Area

**Description**

This area contains comments for the signal file. Entries in this area are not evaluated.

**Related topics**

Basics

Structure of Signal Files..... 100

## Format Area

### Description

This area contains attributes for the signal file. Each row must contain one attribute name in the first column and its value in the next column. An empty row is not allowed. The following table shows the attributes and their descriptions:

Attribute	Type	Used by FIU	Description
Version	String	All	Version of the signal file
BoardType	String	All	Type of the failure insertion unit. The valid types are DS291, DS293, DS749, DS789, DS791, DS1450, DS5355/DS5390. For a DS793, specify the board type "DS791".
FIUMUXAddress	Hexadecimal	DS789, DS791, DS793	Address for the FIU multiplexer in the range 0x1 ... 0x7F (for DS791 and DS793) and 0x1 ... 0x77 (for DS789)
Potential0 ... Potential2	String	DS791, DS793	Name for potentials 0 ... 2. It is to be used for the potential mapping in the EESPort configuration file. In a standard signal file for dSPACE Simulator Mid-Size, the following potentials are set: <ul style="list-style-type: none"> <li>VBAT1 (Potential0)</li> <li>VBAT2 (Potential1)</li> <li>GND (Potential2)</li> </ul>
Potential0 ... Potential4	String	DS293	Name for potentials 3 ... 4. It is to be used for the potential mapping in the EESPort configuration file.
Potential0 ... Potential5	String	DS789	Name for potential 5. It is to be used for the potential mapping in the EESPort configuration file.
OffsetCANAddress	Hexadecimal	DS293	Offset address for modules DS294 and DS295, for example, 0x1
OffsetLoadAddress	Hexadecimal	DS293	Offset address for modules DS282 and DS293, for example, 0x1
OffsetPowerSwitchAddress	Hexadecimal	DS293	Offset address for module DS285, for example, 0x1
FIULoadAddress	String	DS293	Address of the DS293 with the syntax: Cabinet_Rack_Slot Cabinet, Rack, Slot are integers separated by underscores. The valid ranges are: Cabinet: 0 ... 15 Rack: 0 ... 15 Slot: 0 ... 15
RSIMLoadAddress	String	DS293	Address of the DS289 with the syntax: Cabinet_Rack_Slot (see description above)
FIUCANAddress	String	DS293	Address of the DS294 with the syntax: Cabinet_Rack_Slot (see description above)
RSIMCANAddress	String	DS293	Address of the DS289 with the syntax: Cabinet_Rack_Slot (see description above)

### Note

Do not use a semicolon or special characters in any cell, including empty cells, as the table is exported as a CSV file with semicolons as separators.

**Related topics****Basics**

[Structure of Signal Files.....](#) 100

## Signallist Area

**Description**

This area contains the information on the signals and their wiring. Rows 1 to 5 contain header information for the columns. Rows 6 and upward contain information on the signals. See the following table:

Row	Description
1	dSPACE-specific information <sup>1)</sup>
2	Header <sup>2)</sup> of the columns
3	Empty
4	Empty
5	dSPACE-specific header of the column <sup>1)</sup>
6, 7, ...	Each row contains the wiring information <sup>3)</sup> of one signal

<sup>1)</sup> Read-only

<sup>2)</sup> Lets you define the specific column name. Refer to [Column headers](#) on page 103.

<sup>3)</sup> For information on the entries, refer to [Entries of a Row](#) on page 104.

**Note**

Do not use a semicolon or special characters in any cell, including empty cells, as the table is exported as a CSV file with semicolons as separators.

**Column headers**

In the second row, you can define your own column headers. If you change the header, you must observe the following rules:

- The header must not contain a semicolon or be empty.
- The header names must be unique.
- The column headers ECU\_Name(index), PIN\_Name(index), and Faults\_ECU(index) contain an index to assign the entries of the pin, ECU, and failure. For example, the entry in row PIN\_Name(1) is a pin on an ECU specified in row ECU\_Name(1) that can simulate the failures specified in row Faults\_ECU(1). The index number is a positive number and starts with 1.

**Entries of a Row**

The following table lists all the columns that describe the wiring of a signal. For an example, refer to [Structure of Signal Files](#) on page 100.

Name	Format	Type	Used by FIU	Description
ECU_Name(index)	Name	String	All	Name of the ECU. An empty string is not allowed.
PIN_Name(index)	Name	String	All	Name of the pin. An empty string is not allowed. Each name must be unique within an ECU, except the name "-". If the name is "-", this pin is disabled for failure simulation.
Load_Module_Id	Cabinet_Rack_Slot	Integers (separated by an underscore)	DS293	Address of the load module for the pin. The valid ranges are: Cabinet: 0 ... 15 Rack: 0 ... 15 Slot: 0 ... 15
Load_Module_Channel	Channel	Integer	DS293	Channel number for the pin. Several numbers are separated by commas. The number of channels is defined in the Load_channels_required column.
FIU_Module_ID	ModuleID	Hexadecimal(s)	DS291, DS749, DS789, DS791, DS793, DS1450 <sup>1)</sup> , DS5355/ DS5390	ID of the address of the FIU module. Several numbers are separated by commas. For details, see the <i>FIU_Module_ID</i> and <i>FIU_Module_Channel</i> section below.
FIU_Module_Channel	ModuleChannel	Integer(s)	DS291, DS749, DS789, DS791, DS793, DS1450 <sup>1)</sup> , DS5355/ DS5390	Channel number of the address of the FIU module. Several numbers are separated by commas. For details, see the <i>FIU_Module_ID</i> and <i>FIU_Module_Channel</i> section below.
FIU_Default_Termination_R1	-	String	DS1450 <sup>1)</sup>	Default termination resistance for the bus end next to the real-time system. The valid bus termination resistance values are: inf (no termination), 23 Ω, 24 Ω, 26 Ω, 28 Ω, 30 Ω, 36 Ω, 40 Ω, 45 Ω, 60 Ω, 75 Ω, 90 Ω, 120 Ω, 180 Ω, 360 Ω.
FIU_Default_Termination_R2	-	String	DS1450 <sup>1)</sup>	Default termination resistance for the bus end next to the ECU. The valid bus termination resistance values are: inf (no termination), 23 Ω, 24 Ω, 26 Ω, 28 Ω, 30 Ω, 36 Ω, 40 Ω, 45 Ω, 60 Ω, 75 Ω, 90 Ω, 120 Ω, 180 Ω, 360 Ω.
FIU_Type	Name	String	DS291, DS791,	FIU type to which the pin is connected.



Name	Format	Type	Used by FIU	Description
Faults_ECU(index)	FaultsECU	Integer	DS793, DS1450 <sup>1), 2)</sup> DS5390 <sup>3)</sup>  All	Defines the pin failures which can be simulated. Several pin failures are separated by commas. For a description of the pin failures, refer to <a href="#">Failure Classes (dSPACE XIL API Reference [1])</a> .
Load_channels_required	LoadChannelRequired	Integer	DS293	Number of load channels that are required for failure simulation. The number depends on the max. current: 0: No failure simulation or sensor FIU 1: Current of 0 ... 8 A 2: Current of 8 ... 14 A 3: Current of 14 ... 20 A 4: Current of 20 ... 24 A

<sup>1)</sup> The signal list must be manually adapted.

<sup>2)</sup> The channels must be separately configured with DS1450High or DS1450Low as FIU\_Type. See the port configurations in the Demos folder and the migration aspects below.

<sup>3)</sup> Usually, the DS5355/DS5390 is used together with a DS291 in a system. However, the channels must be separately configured with DS291 or DS5390 as FIU\_Type.

**Migration aspects when using DS1450 Bus FIU Board** As of XIL API .NET 2015-A, the signal list for a simulator with a DS1450 Bus FIU Board has changed.

Before, there was one entry per channel with *DS1450* as the FIU\_Type and different failure classes for specifying the high and low configurations of the switches S1, S2 and S3.

Now, each channel is represented by two entries. The configuration of the switches S1, S2, and S3, is done by the FIU\_Types *DS1450High* and *DS1450Low*. The failure classes no longer provide the high or low setting. This allows you a more flexible configuration of the switches. For details, refer to [Failure Classes for the DS1450 Bus FIU Board \(dSPACE XIL API Reference \[1\]\)](#).

#### FIU\_Module\_ID and FIU\_Module\_Channel

You can switch several channels at the same time. These channels are specified with the FIU\_Module\_ID and FIU\_Module\_Channel parameters. Note the following:

- To switch several channels on the same module, specify the module address in FIU\_Module\_ID and the list of the channels in FIU\_Module\_Channel.
- To switch several channels on different modules, specify the list of module addresses in FIU\_Module\_ID and the list of the channels in FIU\_Module\_Channel. You must specify the corresponding module address for each channel. Thus, both lists must have the same number of elements.

**Note**

The address 00 for FIU\_Module\_ID is not supported by modules connected via RS232 interface.

The following table shows some examples of the FIU\_Module\_ID and FIU\_Module\_Channel parameters:

FIU_Module_ID	FIU_Module_Channel	Description
2A,2A,3B	1,2,3	Three channels are switched at the same time: <ul style="list-style-type: none"> <li>Channels 1 and 2 of the module with address 2A</li> <li>Channel 3 of the module with address 3B</li> </ul>
3B	1,2,3,4	Channels 1 ... 4 of the module with address 3B are switched at the same time.
2A,3B	1,2,3	Not allowed because the number of elements for the two parameters are not equal.

**Related topics****Basics**

[Structure of Signal Files..... 100](#)

## How to Configure Signal Files

**Objective**

Before you start using a failure simulation system, you can adapt the signal file to your needs.

**Signal file**

A signal file describes the hardware of the failure simulation system. All the pins to be failure-simulated must be defined in this file. A signal file for your simulator is generated and supplied by dSPACE.

For more information, refer to [Structure of Signal Files](#) on page 100.

**Restrictions**

The allowed failure classes for an ECU are defined in the signal file. If you change these settings, it is at your own risk.

<b>Preconditions</b>	<p>A signal file in Microsoft Excel format must exist.</p> <p>Signal files delivered with older simulators cannot be used with the XIL API EESPort. dSPACE can upgrade the signal files so that you can use the component.</p>
<b>Method</b>	<p><b>To configure a signal file</b></p> <ol style="list-style-type: none"> <li>1 In Microsoft Excel, open the signal file which was delivered with your simulator.</li> <li>2 Change the pin table as required. Be careful when editing the table, as the failure simulation does not work if entries in the table are wrong. Do not change cells that are described as read-only. For a description of the table format, refer to <a href="#">Structure of Signal Files</a> on page 100.</li> <li>3 Export the table as a CSV file (comma separated values). Ensure that the separator in the file is a semicolon. You can select the separator under <b>System settings</b>.</li> </ol>
<b>Result</b>	The signal file is created.

## Evaluating Signal Files

<b>Introduction</b>	When a signal file is read, its contents are automatically evaluated to set the pins in the failure simulation system. This knowledge is important if some pins are missing in the failure simulation system after a signal file is read.
<b>Evaluation points</b>	<p>The following points are automatically evaluated:</p> <ul style="list-style-type: none"> <li>▪ The <b>PIN_Name</b> together with the <b>ECU_Name</b> must be unique for all the signal files of a failure simulation system (including additional signal files).</li> <li>▪ The <b>Load_channels_required</b> row must contain a positive value. A value 0 or lower means the pin is not available for failure simulation.</li> <li>▪ The <b>Load_Module_Id</b> must contain one ID.</li> <li>▪ The <b>Load_Module_Channel</b> must contain the number of values, which is specified in the <b>Load_channels_required</b> row.</li> <li>▪ The <b>Pin_Name(Index)</b> row must contain a string. If the string is "-", the pin is not available for failure simulation.</li> <li>▪ The <b>ECU_Name(Index)</b> row must contain a string. If the string is "-", the ECU is not available for failure simulation.</li> <li>▪ If the <b>Faults_ECU(Index)</b> is equal to "-", load relays are disabled.</li> </ul>

- The combination of `FIU_MODULE_ID` and `FIU_MODULE_CHANNEL` must be unique for all signal files of a failure simulation system, including additional signal files.

For more information on signal files, refer to [Structure of Signal Files](#) on page 100.

## Creating Error Configurations

### Introduction

Provides information on creating an error configuration.

### Where to go from here

### Information in this section

#### [Basic Information on Configuring Errors..... 108](#)

Provides basic information on configuring errors.

#### [Specifying Errors..... 111](#)

For simulating pin failures, you have to create an error in the error configuration.

#### [Activating Electrical Error Simulation..... 114](#)

You can activate errors by triggering an error set.

#### [Updating Electrical Error Simulation..... 116](#)

You can update an error configuration during run time.

#### [Deactivating Electrical Error..... 117](#)

For proper handling of an error configuration, you have to deactivate electrical error simulation.

## Basic Information on Configuring Errors

### Introduction

Provides basic information on configuring errors.

### Creating an error configuration

Failure pattern that you want to specify for your failure simulation are to be specified in an error configuration. An error configuration consists of error sets and errors. An error set is comparable with a failure pattern and an error is comparable to a pin failure.

An error configuration can be specified directly in your script or by loading an error configuration file from the file system.

**ErrorConfiguration** The basic object required for any further handling of an error configuration is the **ErrorConfiguration** object.

An error configuration is an object created by the **EESPort** factory class.

```
var MyErrorConfiguration =
TB.EESPortFactory.CreateErrorConfiguration("My error configuration");
```

**ErrorSet** An error set is used to group errors (pin failures). The order in which the errors are added to the error set is used for the activation of the errors. However, the errors are activated not until the error set is triggered. For more information, refer to [Activating Electrical Error Simulation](#) on page 114.

With defining an empty error set, you can specify a time interval with no error simulating. A specified error can be assigned to more than one error set.

An error set is an object created by the **ErrorConfiguration** object. The **create** method contains the trigger type, the error set has to react to.

#### Note

dSPACE XIL API .NET EESPort implementation supports only manual and software triggers.

To implement a manual trigger, you only have to configure it as the trigger type.

```
var MyErrorSet1 = MyErrorConfiguration.CreateErrorSet("ErrorSet1", TriggerType.eMANUAL)
```

To implement a software trigger, you have to configure it as the trigger type and you have to specify the condition or the duration for the trigger.

```
var MyErrorSet1 = MyErrorConfiguration.CreateErrorSet("ErrorSet1", TriggerType.eSOFTWARE)
# Specify a ConditionWatcher object or a DurationWatcher object as 'MyTrigger'
vendorSpecificErrorSet = IDSErrorSet(MyErrorSet1)
vendorSpecificErrorSet.SetSoftwareTrigger(MyTrigger)
```

The following code is an example of specifying the software trigger with a **ConditionWatcher** object. The condition consists of variable names that are combined by using the ASAM General Expression Syntax (GES).

```
defines = Dictionary[str,str]()
defines.Add('TurnSignalLever', 'Model Root/TurnSignalLever[-1..1]/Value')
defines.Add('BatteryVoltage', 'Model Root/BatteryVoltage[V]/Value')
MyTrigger = testbench.WatcherFactory.CreateConditionWatcher("TurnSignalLever == -1 && BatteryVoltage < 10", defines)
MyTrigger.TimeOut = 30
```

The following code is an example of specifying the software trigger with a **DurationWatcher** object.

```
MyTrigger =
testbench.WatcherFactory.CreateDurationWatcherByTimeSpan(5)
```

The instantiated error set must be added to the error configuration.

```
MyErrorConfiguration.AddErrorSet(MyErrorSet1);
```

**Error** An error is specified by:

- Error category

The error category defines how a signal should be disturbed.

Which errors you can create for a signal depends on the connected FIU hardware.

The following table gives you a short overview on the basic error categories mentioned in the ASAM standard and the related terminology used by dSPACE for the failure classes. For more detailed information, refer to [Failure Classes \(dSPACE XIL API Reference !\[\]\(d0a1791f26d167e866e44ebbf83efebe\_img.jpg\)](#)).

Error Category (ASAM)	Failure Class (dSPACE)
ErrorPin2Pin	Short circuit to another ECU pin
InterruptError	Cable break <sup>1)</sup>
ErrorToGround	Short circuit to GND
ErrorToUbat	Short circuit to Ubat
ErrorToPotential	Short circuit to Potential
InterruptAtPosition	Cable break <sup>2)</sup>
InterchangedPins	Not supported

<sup>1)</sup> Open circuit

<sup>2)</sup> Open circuit on the high or the low bus line. Only supported by the DS1450 Bus FIU Board.

- Error type

The error type defines the disturbance itself. The concrete circuit of each error type can vary between short-circuit errors and interrupt errors.

Error Type (ASAM)	Description
Simple	The error is set statically.
Dynamic	The error is set dynamically. This means that the error is set for a specified duration.
Resistor	The error provides an additional resistor to be switched. The resistor is specified in $\Omega$ .
Dynamic Resistor	The error is a combination of a dynamic error with a switchable resistor.
Loose Contact	The error is set dynamically by a PWM signal specified by frequency and duty cycle. This error type is also known as <i>pulsed switching</i> .
Loose Contact Resistor	The error is a combination of a loose contact error with a switchable resistor.

- Load type

Except for interrupt errors, you can specify whether to disconnect the load from the simulator platform when a signal is disturbed.

For information on the supported EESPort errors, as a combination of the error category and the error type, refer to [Overview of EESPort Errors Supported by dSPACE FIU Hardware \(dSPACE XIL API Reference !\[\]\(d5d7044e5caf6907399af2dced8d6ff8\_img.jpg\)](#)).

The creation of an error is based on the ErrorFactory class.

```
var MyErrorFactory = MyErrorConfiguration.GetErrorFactory();
```

The instantiated error factory can be used to create errors and add them to the error set. The following example shows how to create a simple short circuit to ground on Signal 1 with disconnecting the load. By using the `ToBaseError` method, the created error is returned as a `BaseError` object, which you can use for reading the settings of the error.

```
MyErrorSet1.AddError(MyErrorFactory.CreateErrorToGround("Signal 1", LoadType.eWITHOUT_LOAD).AsSimple().ToBaseError());
```

For more information on the various errors, refer to [Specifying Errors](#) on page 111.

## Handling an error configuration file

An error configuration can be saved to a file and loaded in other XIL API applications.

Writing to a file and reading from a file is supported by the common XIL API `DocumentHandling` package. This provides writer and reader objects for the different file formats used with XIL API ports.

### Note

If you have created an error configuration file using the dSPACE XIL API .NET implementation, and this file contains the configuration of software-triggered errors, it is no longer compatible to the ASAM AE XIL standard. You can use this error configuration file only with the dSPACE EESPort.

To make your electrical error simulation tests compatible to the ASAM standard, use error configurations, which you create during run time of your application.

Before you can create a `FileWriter` object to provide access to the methods and properties to write an error configuration to a file, the Testbench and the EESPort must be created.

```
private const string MyFile = "D:\\Work\\MyErrorConfiguration.xml";
var MyErrorConfigurationFileWriter =
MyErrorConfiguration.CreateEESConfigurationFileWriter(MyFile);
```

The previously implemented error configuration in your code can then be saved to the file writer.

```
MyErrorConfiguration.Save(MyErrorConfigurationFileWriter);
```

An existing error configuration file can be loaded to your XIL API application by loading the file via a `FileReader` object.

```
var MyErrorConfiguration2 = TB.EESPortFactory.CreateErrorConfiguration("New error configuration");
var MyErrorConfigurationFileReader = MyErrorConfiguration2.CreateEESConfigurationFileReader(MyFile);
MyErrorConfiguration2.Load(MyErrorConfigurationFileReader);
```

## Examples

For an example how to specify an error configuration with the various variants of errors, refer to the `ErrorConfiguration.cs` sample project.

# Specifying Errors

## Introduction

For simulating pin failures, you have to create an error in the error configuration.

**Available attributes for errors**

The main property of an error is its error category. For each error category, the API provides specific create methods, which you can enhance by further parameters and properties. Most of these attributes are available for any error category. For limitations, refer to the detailed overview at the bottom of this topic.

**Note**

For each ECU pin, you have to configure which failure class is allowed to be simulated. Only the allowed pin failures can be applied to the failure simulation system.

- For systems with discrete FIU, the failure classes allowed for failure simulation are specified in the simulator's *signal file*. For more information, refer to [Defining Failure Classes with Signal Files](#) on page 99.
- For SCALEXIO systems with integrated SCALEXIO FIU, you specify the failures allowed for the ECU signal and the load rejection of signal measurement channels in ConfigurationDesk. For more information, refer to [Specifying Allowed Failure Classes for ECU Pins \(SCALEXIO – Hardware and Software Overview\)](#).

**Simple errors** If you want to create a simple error, you have to use the **Simple** error builder. Simple error means, that the error is switched once. The pin state will be changed not until you explicitly specified it.

**Dynamic errors** If you want to create an error for a specified duration, you can use the **Dynamic** error builder. If the error is to be repeated many times, you can additionally specify the frequency for the activation.

Simulating dynamic errors is limited by the latencies of the used simulation system. Refer to [Latencies when Performing Electrical Error Simulation](#) on page 68.

**Resistor errors** If the signal line provides a switchable resistor, you can control it by using the **Resistor** error builder.

**Loose contact failures** DS793 modules and SCALEXIO systems use semiconductor devices for switching the errors. With these devices very fast switching between the failure potential and the normal state is possible. This allows loose contacts or switch bouncing to be simulated. Such an error is realized by using the **LooseContact** error builder, refer to [Simulating Loose Contacts or Switch Bouncing](#) on page 118.

**Load rejection** Except for interrupt errors, you can specify whether to disconnect the load from the simulator platform when a signal is disturbed. You do this by specifying **WITHOUT\_LOAD** as the load type. The **WITH\_LOAD** load type is used to keep the load connected.

**Multi-pin errors**

If you want to enhance the number of signals to be simultaneously switched by activating an error, you can specify the pin failures also as multi-pin errors.



Interrupt errors cannot be specified as multi-pin errors.

#### Note

##### Using SCALEXIO system

Multi-pin errors must be specified with at least three signals. A multi-pin error specified with two signals will be automatically realized as a regular *Short to Pin* error. Using mixed error categories is not supported.

If you want to use multi-pin errors and *Short to Pin* errors at the same time, you have to explicitly specify the fail rails for the multi-pin errors to be used.

#### Details on the available errors

The following code examples show you the usual combinations of the error attributes. These extracts must be included in a correctly instantiated error factory, refer to [Basic Information on Configuring Errors](#) on page 108.

**Cable break** A cable break is realized by an `InterruptError` error.

```
// Static error
CreateInterruptError("Signal1").AsSimple().ToBaseError()

// Dynamic error with a duration of 50 ms (50,000 µs)
CreateInterruptError("Signal1").AsDynamic(50000).ToBaseError()

// Dynamic error with a duration of 50 ms with a frequency of 1 Hz and a duty cycle of 50%
CreateInterruptError("Signal1").AsDynamic(50000).WithFrequency(1, 50).ToBaseError()
```

**Short circuit to potential** A short circuit can be realized to different potentials.

- A short circuit to ground (GND) is realized by an `ErrorToGround` error.
- A short circuit to +Ubat is realized by an `ErrorToUbat` error.
- A short circuit to a specified potential is realized by an `ErrorToPotential` error.

These errors can be used in the same way. The code examples are shown for the `ErrorToGround` error.

```
// Static error without load rejection
CreateErrorToGround("Signal1", LoadType.eWITH_LOAD).AsSimple().ToBaseError()

// Static error with load rejection
CreateErrorToGround("Signal1", LoadType.eWITHOUT_LOAD).AsSimple().ToBaseError()

// Static error without load rejection and a 50 Ω resistor
CreateErrorToGround("Signal1", LoadType.eWITH_LOAD).AsSimple().WithResistor(50).ToBaseError()

// Dynamic error without load rejection and a duration of 50 ms (50,000 μs)
CreateErrorToGround("Signal1", LoadType.eWITH_LOAD).AsDynamic(50000).ToBaseError()

// Dynamic error without load rejection, a duration of 50 ms, a frequency of 1 Hz and a duty cycle of 50%
CreateErrorToGround("Signal1", LoadType.eWITH_LOAD).AsDynamic(50000).WithFrequency(1, 50).ToBaseError()

// Static multi-pin error without load rejection
CreateErrorMultiToGround(["Signal1", "Signal2"], [LoadType.eWITH_LOAD, LoadType.eWITH_LOAD]).AsSimple().ToBaseError()
```

**Short circuit to other ECU signal** A short circuit to another ECU signal is realized by an `ErrorPin2Pin` error.

```
// Static error without load rejection
CreateErrorPin2Pin("Signal1", "Signal2", LoadType.eWITH_LOAD, LoadType.eWITH_LOAD).AsSimple().ToBaseError()

// Static error with load rejection
CreateErrorPin2Pin("Signal1", "Signal2", LoadType.eWITHOUT_LOAD, LoadType.eWITHOUT_LOAD).AsSimple().ToBaseError()

// Static error without load rejection and a 50 Ω resistor
CreateErrorPin2Pin("Signal1", "Signal2", LoadType.eWITH_LOAD, LoadType.eWITH_LOAD).AsSimple().WithResistor(50).ToBaseError()

// Dynamic error without load rejection and a duration of 50 ms (50,000 μs)
CreateErrorPin2Pin("Signal1", "Signal2", LoadType.eWITH_LOAD, LoadType.eWITH_LOAD).AsDynamic(50000).ToBaseError()

// Dynamic error without load rejection, a duration of 50 ms, a frequency of 1 Hz and a duty cycle of 50%
CreateErrorPin2Pin("Signal1", "Signal2", LoadType.eWITH_LOAD, LoadType.eWITH_LOAD).AsDynamic(50000).WithFrequency(1, 50).ToBaseError()

// Static multi-pin error without load rejection
CreateErrorMultiPin2Pin(["Signal1", "Signal2", "Signal3"], [LoadType.eWITH_LOAD, LoadType.eWITH_LOAD, LoadType.eWITH_LOAD]).AsSimple().ToBaseError()
```

## Activating Electrical Error Simulation

### Introduction

You can activate errors by triggering an error set.

### Basics

Depending on the failure class supported by the FIU hardware, you can activate errors for one or more pins at the same time, for example, by using the `CreateErrorToGround` method for one pin or the `CreateErrorMultiToGround` for multi pins of the instantiated error factory. An

error must be contained in an error set. Otherwise it cannot be activated. All created error sets and errors refers to an error configuration.

You can start the execution of an error set and switching to the next error set by using different triggers:

- Manual trigger  
The trigger is explicitly specified in the XIL API application.
- Hardware trigger  
The FIU hardware reacts on a trigger input line.
- Software trigger  
The FIU hardware reacts on a trigger signal defined in software, for example, defined in the model or in other software connected to the FIU hardware.

#### Note

dSPACE XIL API .NET EESPort implementation supports only manual and software triggers.

### Starting an error set via manual trigger

After you have loaded an error configuration file or specified an error configuration in your application, you can add it to your EESPort, download it to the electrical error simulation system and activate it. Then you can trigger the errors.

You can use only one error configuration at the same time.

```
// Connect an error configuration to the EESPort
MyEESPort.SetErrorConfiguration(MyErrorConfiguration);

// Download the error configuration to the EESPort
MyEESPort.Download();

// Start the EESPort
MyEESPort.Activate();

// Activate the first error set specified in the error configuration
MyEESPort.Trigger();

// Wait for executing the triggered error set, for example by calling Sleep
// Activate the second error set specified in the error configuration
MyEESPort.Trigger();
...
```

When you have activated errors, you should deactivate them afterwards, refer to [Deactivating Electrical Error](#) on page 117.

### Starting an error set via software trigger

After you have loaded an error configuration file or specified an error configuration in your application, you can add it to your EESPort, download it to the electrical error simulation system and activate it. The errors are automatically triggered if the conditions are fulfilled that you specified in the error configuration. The condition can be specified via a ConditionWatcher or a DurationWatcher. The **WaitForTrigger** method waits for the error set to be

activated for the specified time. If the error set is not activated within the specified time, an exception is thrown. The `WaitForTrigger` method can be used to synchronize the start of the test application with the execution of the error configuration.

You can use only one error configuration at the same time.

```
// Connect an error configuration to the EESPort
MyEESPort.SetErrorConfiguration(MyErrorConfiguration);

// Download the error configuration to the EESPort
MyEESPort.Download();

// Start the EESPort
MyEESPort.Activate();

// The error sets are automatically triggered by the specified conditions
// Wait for trigger for 5 seconds
MyEESPort.WaitForTrigger(5000)
...
```

When you have activated errors, you should deactivate them afterwards, refer to [Deactivating Electrical Error](#) on page 117.

For information on updating a downloaded error configuration, refer to [Updating Electrical Error Simulation](#) on page 116.

## Updating Electrical Error Simulation

### Introduction

You can update an error configuration during run time.

### Basics

Usually, you are working with a static error configuration, i.e., you create it, download it to the hardware, and execute all its contained error sets. With the **Update** method of an EESPort instance, you can dynamically append the contents of an error configuration during run time.

You can update only the previously downloaded error configuration. You cannot use the **Update** method to download a new error configuration. You can only append the error configuration with error sets. You are not allowed to delete or change already configured error sets in the currently downloaded error configuration.

The update has no effect on the already downloaded error configuration. Its error sets are executed as specified. There is only one use case in which the execution of an error set will be interrupted by an updated error configuration.

For this exception, the following conditions must be fulfilled:

- The currently active error set must be the last error set in the already downloaded error configuration.

- The first new error set in the updated error configuration is to be triggered by software.
- The specified condition for the software trigger is fulfilled.

In this case, the new error set is directly activated and interrupts the execution of the previous error set.

### Updating an error configuration

An already downloaded error configuration can be updated during run time. For information on the initial download of an error configuration, refer to [Activating Electrical Error Simulation](#) on page 114.

```
// Create new error set
var MyNewErrorSet = MyErrorConfiguration.CreateErrorSet("ErrorSetUpdate",
TriggerType.eMANUAL);

// Create a new error (static error without load rejection)
MyNewErrorSet.AddError(MyErrorFactory.CreateErrorToGround("Signal 1",
LoadType.eWITHOUT_LOAD).AsSimple().ToBaseError()));

// Save the modified error configuration file
MyErrorConfiguration.Save(MyErrorConfigurationFileWriter);

// Update the EESPort with the modified error configuration
MyEESPort.Update();

// Activate the first new error set specified in the modified error configuration
MyEESPort.Trigger();

// Wait for executing the triggered error set, for example by calling Sleep
// Activate the second new error set specified in the modified error configuration
MyEESPort.Trigger();
...
```

When you have activated errors, you should deactivate them afterwards, refer to [Deactivating Electrical Error](#) on page 117.

## Deactivating Electrical Error

### Introduction

For proper handling of an error configuration, you have to deactivate the electrical error simulation.

### Terminating errors

The execution of errors is stopped, if there is no further trigger to start an error set.

You can explicitly terminate errors by deactivating the EESPort.

```
MyEESPort.Deactivate();
```

**Note**

If you try to trigger an error set that is not available, the **Dispose** method of the EESPort is executed.

**Unload the error configuration**

After deactivating the EESPort you are able to activate it again with the same error configuration.

If you want to load another error configuration or terminate the entire test application, you have to unload the error configuration.

```
MyEESPort.Unload();
```

**Terminating the test application**

After the error configuration has been unloaded, you can disconnect the EESPort from your FIU hardware and free the instantiated resource from the EESPort factory.

```
MyEESPort.Disconnect();
...
MyEESPort.Dispose();
...
```

## Simulating Unstable Pin Failures (Loose Contacts) with DS793 Modules and SCALEXIO Systems

**Introduction**

With a DS793 Sensor FIU or in a SCALEXIO system, you can simulate failures that are not permanent (pulsed switching), i.e., loose contacts or switch bouncing.

## Simulating Loose Contacts or Switch Bouncing

**Introduction**

DS793 modules and systems with integrated SCALEXIO FIU use semiconductor devices for switching the pin failures. The use of semiconductor switches makes pulsed switching possible. Other FIU modules switch via relays. Relays are mechanical switches that themselves bounced, so specified switch bouncing is not possible.

**LooseContactError**

Pulsed switching means that the channels are switched alternately between the failure potential and the normal state very fast. This allows loose contacts or

switch bouncing to be simulated. The switching sequence is specified by the frequency and its duty cycle, and the duration of the error. These attributes can be specified by using the **Dynamic** error builder for the **LooseContactError** error.

- The frequency specifies the number of switches per second.
- The duty cycle specifies the percentage of the activation time of the error. In the rest of the time (100% - DutyCycle), the error is deactivated.
- The duration specifies the time of the entire switching sequence.

#### Note

DS791 and DS793 FIU modules have the same failure classes, but only a DS793 can simulate loose contacts or switch bouncing. The failure simulation of a DS791 is always static.

#### Preconditions for using DS793 modules

The following preconditions apply to simulating pulsed switching with DS793 modules:

- An error set can contain dynamic errors with different frequencies, duty cycles and durations.
- The signal file must contain the *FIU\_Type* column in the Signallist area, and the DS793 must be specified for the ECU pin. If this column is missing, you can extend the signal file yourself. For details, refer to [Signallist Area](#) on page 103

#### Switching limitations

The following limitations must be observed for switching.

##### Limitations for DS793 modules

- The duty cycle must be in the range 0.025 ... 99.975 %.
- The frequency must be in the range 6.105006 ... 25,000.0 Hz.

##### Limitations for systems with integrated SCALEXIO FIU

- The duty cycle must be in the range 0.0000344827 ... 99.9999655173 %.
- The frequency must be in the range 0.001724 ... 5000.0 Hz.





# Working with XIL API .NET on Linux

Where to go from here

Information in this section

Getting Started with XIL API .NET on Linux.....	122
Reference Information on XIL API .NET on Linux.....	133
Limitations and Migration Aspects When Working with XIL API .NET on Linux.....	136

# Getting Started with XIL API .NET on Linux

**Introduction** Provides basic information on the dSPACE XIL API .NET on Linux, how to install it, and how to run the included demo applications.

## Where to go from here

## Information in this section

[Introduction to the dSPACE XIL API .NET on Linux.....](#) 122

The dSPACE XIL API .NET on Linux provides a standardized interface for controlling and accessing simulation applications.

[How to Install the dSPACE XIL API .NET on Linux.....](#) 123

To install the dSPACE XIL API .NET on Linux on Ubuntu Linux 18.04 LTS.

[How to Register a Platform.....](#) 125

To make a local or remote platform known to the XIL API .NET server, you must register it once.

[How to Load and Start a Simulation Application.....](#) 127

To load a simulation application to a dSPACE platform and start its execution.

[How to Execute the Python Demo Scripts.....](#) 128

To understand how to access the XIL API .NET server in Python, you can inspect and execute the provided demo scripts.

[How to Build and Run the C# Demo Applications.....](#) 131

To understand how to access the XIL API .NET server in C#, you can inspect, build, and run the provided demo applications.

## Introduction to the dSPACE XIL API .NET on Linux

**Overview** The dSPACE XIL API .NET on Linux provides an interface for controlling and accessing simulation applications according to the ASAM AE XIL API standard.

## Basics

The dSPACE XIL API .NET implementation is based on the ASAM AE XIL 2.1.0 standard.

The main approach of this standard is the decoupling of test automation software and test hardware to allow for the reuse of test cases on different hardware systems.

To understand the concepts of the ASAM AE XIL standard, it is recommended to read the ASAM user documentation before using the dSPACE-specific API implementation. The documents of the ASAM AE XIL standard are not contained

in the dSPACE XIL API .NET on Linux installation. You find the documentation in the dSPACE XIL API .NET on Windows installation or you can request it from dSPACE Support.

**Basic work steps** To work with the included demo programs, perform the following steps:

- Make the platform known to the system. Refer to [How to Register a Platform](#) on page 125.
- Load and start the simulation application to the platform. [How to Load and Start a Simulation Application](#) on page 127.
- Access the running simulation application from your C# or Python sources via the API. Refer to [How to Execute the Python Demo Scripts](#) on page 128 and [How to Build and Run the C# Demo Applications](#) on page 131.

---

#### Compatibility

**Linux distribution** dSPACE recommends using Ubuntu Linux 18.04 LTS 64-bit. If you want to use a different Linux distribution, contact dSPACE Support.

---

#### Migrating applications that use XIL API .NET on different operating systems

If you want to migrate an application that works with an XIL API .NET on Windows server to work with dSPACE XIL API .NET on Linux, you have to consider some differences between the two APIs. Refer to [Aspects of Migrating From Windows to Linux](#) on page 137.

---

#### Limitations

There are some limitations to note when using the dSPACE XIL API .NET on Linux implementation. Refer to [Limitations for Working with XIL API .NET on Linux](#) on page 136.

---

#### Demos

Demo applications for common use cases are provided with the product. Refer to [How to Build and Run the C# Demo Applications](#) on page 131.

## How to Install the dSPACE XIL API .NET on Linux

---

#### Objective

To install dSPACE XIL API .NET on Linux.

---

#### Preconditions

The following preconditions must be met:

- You must have Ubuntu Linux 18.04 LTS (Bionic Beaver).
- You do not have to work as the Linux superuser (**root**), but during installation, you are prompted to enter the superuser password.
- Microsoft .NET Core 3.1 Runtime must be installed on the Linux system.

If it is not installed, proceed as follows:

- If the Linux system has Internet access and you have clearance from your local system administrator, execute the `configure-ms-package-server.sh` script.

The script carries out the instructions on the official Microsoft .NET Core installation website:

<https://docs.microsoft.com/en-us/dotnet/core/install/linux-ubuntu>

- If you cannot access the Internet, you can, for example, use a local package mirror to get the Microsoft .NET Core 3.1 packages. Follow the instructions on the Microsoft .NET Core installation website to perform the installation.

## Method

### To install dSPACE XIL API .NET on Linux

- 1 Open a web browser and navigate to [https://www.dspace.com/go/ProductsForLinux](https://www.dsspace.com/go/ProductsForLinux). There you find a link to the specific item for XIL API .NET for Linux.
- 2 Logon with your mydSPACE account to get access to the download package.
- 3 In a Terminal window, create a new working directory with write privileges and download the `dSPACE-xilapi.net-4.0.tar.gz` archive to it.

- 4 Change to the new directory and extract the GZ archive:

```
tar -xzf dSPACE-xilapi.net-4.0.tar.gz
```

- 5 If you work with floating network license servers, uncomment the following line in the `./install.sh` installation script, i.e., remove the leading hash (#) in:

```
# enable_license_server 127.0.0.1
```

Replace the local host address (127.0.0.1) with the IP address or the FQDN of your license server. If you work with multiple license servers, add a corresponding line for each server to the script.

- 6 Execute the installation script:

```
./install-xilapi.sh
```

During the execution, you are prompted to enter the root password.

The required Debian packages with all of their dependencies are automatically installed on the Linux system.

- 7 Verify that the CodeMeter server process is running by executing the following command:

```
systemctl status codemeter.service
```

If the process is running, you will see the following output:

```
● codemeter.service - CodeMeter RunTime Server
   Loaded: loaded (/lib/systemd/system/codemeter.service; enabled; vendor preset
   Active: active (running) since Thu 2020-06-04 07:23:32 PDT; 38min ago
     Main PID: 740 (CodeMeterLin)
        Tasks: 11 (limit: 2266)
      CGroup: /system.slice/codemeter.service
              └─740 /usr/sbin/CodeMeterLin -f

Jun 04 07:23:32 ubuntu systemd[1]: Started CodeMeter RunTime Server.
```

If the process is not running, this will be displayed as inactive in the output in white letters. In this case, start the process as a superuser with the following command:

```
sudo systemctl start codemeter.service
```

Per default, CodeMeter Runtime uses a hardware token. This token must be connected if you use this method to get licenses.

If you want to use floating network licenses and you did not use the installation script to enable license servers, use the following command in the Terminal window to add a license server to the server list:

```
cmu --add-server <ServerIP or FQDN>
```

## Result

You installed dSPACE XIL API .NET on Linux on your Linux system.

The installation is located under `/opt/dspace/xilapi.net4.0`, which is referenced later in this document as `<InstallationPath>`.

The following table provides an overview of the relevant folder content.

Subdirectory	Contents
<code>/opt/dspace/share/&lt;GUID&gt;/log</code>	Folder for saving simulation log files.
<code>&lt;InstallationPath&gt;/Testbench/Main/bin</code>	Folders that contain the XILAPI.NET executables and libraries.
<code>&lt;InstallationPath&gt;/MAPort/Main/bin</code>	
<code>&lt;InstallationPath&gt;/demos</code>	Folders that contain demo applications for common use cases of working with the XIL API .NET API.

## Next step

You can now register the platforms that you want to access via the XIL API.

# How to Register a Platform

## Objective

To make a platform known to the XIL API .NET server, you must register it once.

## Supported dSPACE platforms

dSPACE XIL API .NET on Linux supports the following platforms:

- VEOS
- SCALEXIO (SCALEXIO system with SCALEXIO Processing Unit or DS6001 Processor Board)

## Preconditions

- The dSPACE platform must be connected and switched on.
- If you want to work with VEOS, the VEOS Simulator must be installed.
- The IP address of the platform is required as input data.

If you want to register a SCALEXIO or DS6001 MP system, refer to the note at the end of this topic.

If you want to register remote VEOS platforms, you must meet the following additional preconditions:

- The VEOS installation on the local system and on the remote system must be of the same dSPACE Release.
- You must enable remote access *before* you start the VEOS Simulator.

```
/opt/dspace/veos5.1/bin/veoskernel config --enable-remote-access
```

#### Tip

- If you do not know the IP address of the Linux machine, execute `hostname -I` in a Terminal window on the Linux machine.
- If you run Linux on a virtual machine, make sure that the network connection of the virtual machine is set to NAT.

## Method

### To register a platform

- 1 In a Terminal window, enter:

```
<InstallationPath>/MAPort/Main/bin/CmdLoader -start -rnc  
<IpAddress> -t <PlatformType>
```

## Result

You registered the platform.

The registration data is stored in the recent platform configuration.

**Examples** To register a VEOS platform with the IP address 192.168.10.1, use the following command.

```
CmdLoader -start -rnc 192.168.10.1 -t VEOS
```

To register a SCALEXIO platform, such as a SCALEXIO Processing Unit or a DS6001 Processor Board, with the IP address 192.168.10.1, use the following command.

```
CmdLoader -start -rnc 192.168.10.1 -t SCALEXIO
```

You can find examples on using the `CmdLoader` in `<InstallationPath>/demos/MAPort/CmdLoader`.

#### Note

##### Using a SCALEXIO MP or DS6001 MP system

With the `CmdLoader` on Linux, you cannot register a multiprocessor platform.

To work with such a platform, you must provide the required `RecentHardware.xml` file on your Linux system. You can do this in the following way.

- Register the SCALEXIO MP or DS6001 MP system on a Windows platform, e.g., in ControlDesk.
- Copy the `RecentHardware.xml` file from your Windows system in `C:\ProgramData\dSPACE\PlatformManagement` to `{XDG_DATA_HOME}/dspace/platformmanagement` on your Linux system.

#### Next step

You can now download a simulation application to the registered platform.

#### Related topics

##### Basics

[Introduction to the dSPACE XIL API .NET on Linux.....](#) 122

##### HowTos

[How to Install the dSPACE XIL API .NET on Linux.....](#) 123

##### References

[CmdLoader.....](#) 133

## How to Load and Start a Simulation Application

#### Objective

To load a simulation application to a dSPACE platform and start its execution.

#### Preconditions

The following preconditions must be met:

- The dSPACE platform must be registered. Refer to [How to Register a Platform](#) on page 125.

- If you want to use VEOS as platform for offline simulation, the VEOS Simulator must be running.
- The path to the simulation application is required as input data. Refer to [How to Install the dSPACE XIL API .NET on Linux](#) on page 123.  
For example, the application path to the platform-specific turnlamp demo simulation application is  
<InstallationPath>/demos/SampleExperiments/TurnSignal\_<Platform>/BuildResult/turnlamp.sdf

Method	<b>To load and start a simulation application</b>  1 In a Terminal window, enter: <div>&lt;InstallationPath&gt;/MAPort/Main/bin/CmdLoader &lt;ApplicationPath&gt; -p &lt;PlatformName&gt;</div>
Result	You loaded the simulation application to the dSPACE platform and started it.
Next step	You can now access the running simulation application via the XIL API .NET server.  For examples of controlling a simulation application via the XIL API, refer to <a href="#">How to Execute the Python Demo Scripts</a> on page 128 or to <a href="#">How to Build and Run the C# Demo Applications</a> on page 131.
Related topics	<div>Basics</div> <div>Introduction to the dSPACE XIL API .NET on Linux..... 122</div> <div>HowTos</div> <div>How to Install the dSPACE XIL API .NET on Linux..... 123</div> <div>References</div> <div>CmdLoader..... 133</div>

## How to Execute the Python Demo Scripts

Objective	To understand how to access a running simulation application via the XIL API .NET server in Python, you can inspect and execute the provided demo scripts.
-----------	------------------------------------------------------------------------------------------------------------------------------------------------------------



**Preconditions**

The following preconditions must be met:

- The target platform to run the simulation application must be registered on the XIL API .NET server. Refer to [How to Register a Platform](#) on page 125.
- Python 3.9 must be installed to execute the demo scripts.

**Method****To execute the Python demo scripts**

- 1 Create your own working directory to keep your demo scripts independent from the installed scripts. In a Terminal window on the XIL API .NET server, enter:

```
mkdir <WorkingDir> && chmod u+w <WorkingDir>
```

- 2 Copy the demo scripts to the working directory:

```
cp -r <InstallationPath>/demos <WorkingDir>
```

- 3 Change to the demo script directory:

```
cd <WorkingDir>/demos/MAPort/Python
```

The following Python demo scripts are available:

Demo Name	Description
1_ReadWrite	This demo shows you how to read and write variables on the simulation platform.
2_BasicCapturing	This demo shows you how to capture two model variables started by a trigger. The capture result is stored in the internal memory.
3_DurationWatcherCapturing	This demo shows you how to control the capturing via a duration watcher.
4_ConditionWatcherCapturing	This demo shows you how to control the capturing via a condition watcher.
5_CapturingToMDFFile	This demo shows you how to save the capture result to an MDF file.
6_FetchingDataWhileCapturing	This demo shows you how to get a current capture result.
10_ValueContainer	This demo shows you how to use the value container classes.
11_CDFXHandling	This demo shows you how to write multiple variable values to the platform by using the <b>DownloadParameterSets</b> method. The CDFX file to be loaded can be created in ControlDesk.
13_CheckVariableNames	This demo shows you how to use the <b>CheckVariableNames</b> method to check for the existence of variables in the loaded real-time application.
15_TriggerClientEvents_Memory	This demo shows you how to trigger client events while a capture is running. In this demo script, the capture data is stored in memory.
16_TriggerClientEvents_MDFFile	This demo shows you how to trigger client events while a capture is running. In this demo script, the capture data is stored in an MDF file.

- 4 Depending on the application that you want to access, adjust the following variables in the demo scripts via a text editor:

- For single-platform VEOS:

```
IsMPSSystem = False
MAPortConfigFile = \
    r"../Common/PortConfigurations/MAPortConfigVEOS.xml"
```

- For multicore VEOS:

```
IsMPSystem = True
MAPortConfigFile = \
    r"../Common/PortConfigurations/MAPortConfigVEOSMC.xml"
```

- For a SCALEXIO Processing Unit:

```
IsMPSystem = False
MAPortConfigFile = \
    r"../Common/PortConfigurations/MAPortConfigSCALEXIO.xml"
```

- For a DS6001 Processor Board:

```
IsMPSystem = False
MAPortConfigFile = \
    r"../Common/PortConfigurations/MAPortConfigDS6001.xml"
```

If you want to use the SCALEXIO Processing Unit or the DS6001 Processor Board as an MC or MP system, you must set the `IsMPSystem` option to `True` and specify the related MAPort configuration file.

- 5 You can now run the demo scripts one after another by calling the Python interpreter:

```
python3.9 ./<DemoName>
```

For example, enter `python3.9 ./1_ReadWrite.py` to run the first demo.

#### Tip

- Alternatively, you can execute the demo scripts by entering:  
`./<DemoName>`
- You can open each demo script in a text editor to study how it accesses the running simulation application via the dSPACE XIL API .NET server.

## Result

You executed the Python demo scripts that access the running simulation applications via the dSPACE XIL API .NET server.

## Related topics

### Basics

[Introduction to the dSPACE XIL API .NET on Linux..... 122](#)

### HowTos

[How to Build and Run the C# Demo Applications..... 131](#)

## How to Build and Run the C# Demo Applications

### Objective

To understand how to access a running simulation application via the XIL API .NET server in C#, you can inspect, build, and run the provided demo applications.

### Preconditions

The following preconditions must be met:

- The target platform to run the simulation application must be registered. Refer to [How to Register a Platform](#) on page 125.
- The .NET Core SDK on Linux must be installed to build the demo applications. Refer to <https://docs.microsoft.com/en-us/dotnet/core/install/linux>.

### Method

#### To build and run the C# demo applications

- 1 If you already created a working directory with the demo scripts, skip steps 1 and 2.

Create your own working directory to keep your demo scripts independent from the installed scripts. In a Terminal window, enter:

```
mkdir <WorkingDir>
```

- 2 Copy the demo scripts to the working directory:

```
cp -r <InstallationPath>/demos <WorkingDir>
```

- 3 Change to the demo directory:

```
cd <WorkingDir>/demos
```

- 4 Using the .NET Core SDK on Linux, build the executable demo applications:

```
dotnet msbuild /Restore /t:Rebuild /v:n  
/p:Configuration=Release /p:Platform=x64 MAPortDemos.sln
```

The following C# demo applications are available:

Demo Name	Description
1_ReadWrite	This demo shows you how to read and write variables on the simulation platform.
2_BasicCapturing	This demo shows you how to capture two model variables started by a trigger. The capture result is stored in the internal memory.
3_DurationWatcherCapturing	This demo shows you how to control the capturing via a duration watcher.
4_ConditionWatcherCapturing	This demo shows you how to control the capturing via a condition watcher.
5_CapturingToMDFFile	This demo shows you how to save the capture result to an MDF file.
6_FetchingDataWhileCapturing	This demo shows you how to get a current capture result.
10_ValueContainer	This demo shows you how to use the value container classes.
11_CDFXHandling	This demo shows you how to write multiple variable values to the platform by using the <b>DownloadParameterSets</b> method. The CDFX file to be loaded can be created in ControlDesk.
13_CheckVariableNames	This demo shows you how to use the <b>CheckVariableNames</b> method to check for the existence of variables in the loaded real-time application.

Demo Name	Description
15_TriggerClientEvents_Memory	This demo shows you how to trigger client events while a capture is running. In this demo script, the capture data is stored in memory.
16_TriggerClientEvents_MDFFile	This demo shows you how to trigger client events while a capture is running. In this demo script, the capture data is stored in an MDF file.

**5** You can now run the demo applications one after another:

```
./bin/Release/<DemoApplicationName>
```

For example, enter `./bin/Release/1_ReadWrite` to run the first demo.

---

## Result

You built and ran C# demo applications that accesses a running simulation application via the dSPACE XIL API .NET server.

---

## Related topics

### Basics

[Introduction to the dSPACE XIL API .NET on Linux..... 122](#)

### HowTos

[How to Execute the Python Demo Scripts..... 128](#)

# Reference Information on XIL API .NET on Linux

## Introduction

Provides reference information on the dSPACE XIL API .NET on Linux.

## CmdLoader

### Syntax

```
/opt/dspace/xilapi.net4.0/MAPort/Main/bin/CmdLoader
<{Application [-suppress_start]}|-start|-stop|-pause|-unload>
<-p PlatformName | {-rnc NetClient -t PlatformType}>
[-go Seconds] [-af AccelerationFactor]
[-ra]
[-ol OutputLevel] [-q]
[-?]
```

With the following meta-syntax:

Symbols	Meaning
< >	These brackets enclose mandatory parameters.
[ ]	These brackets enclose optional parameters.
{ }	These brackets enclose parameter combinations.

### Description

This command lets you download and control the execution of a simulation application (RTA and OSA) on a dSPACE platform.

You can use it from the Linux command line interface, i.e., manually in a Terminal window or scripted in a shell script or makefile.

**Options and parameters** The following options and parameters are provided:

Name	Meaning
<Application>	The name of the SDF file that specifies the RTA or OSA file to be downloaded to a registered platform.
-suppress_start	Prevents the automatic start of the simulation after it was downloaded.
-start	Starts the currently loaded simulation.
-stop	Stops the currently loaded simulation.
-pause	Pauses the currently loaded simulation.
-unload	Clears the currently loaded RTA or OSA file from the simulation platform.
-p <PlatformName>	Specifies the name of the dSPACE platform to be accessed. The recent hardware configuration is scanned for this platform name. If the related platform is not registered, it is registered before the application file that is specified in the <Application> parameter is loaded to the specified platform.

Name	Meaning
-rnc <NetClient>	Specifies the name of the net client or the IP address to be used for remote access.
-t <PlatformType>	Specifies the type of simulation platform to be accessed. Current only <b>VEOS</b> and <b>SCALEXIO</b> are available as platform types.
-go <Seconds>	Specifies the number of seconds during which the standard output can display error messages returned by the error check mechanism of the platform. The maximum time value is 120 s. The internal logging that is used for requests to dSPACE Support is not affected.
-af <AccelerationFactor>	Specifies the factor by which to accelerate or decelerate the execution of the simulation in relation to real time.
-ra	Registers all platforms that are configured in the following file: <code>~/ .local/share/dspace/platformmanagement/RecentHardware.xml</code> This file is commonly used to share platform configurations among multiple systems.
-ol <OutputLevel>	Specifies how detailed the logging information is that is printed to <b>stdout</b> . OutputLevel is an integer from 0 to 5, where 0 relates to no logging and 5 to a verbose logging. The internal logging that is used for requests to dSPACE Support is not affected.
-q	Runs the command in quiet mode, i.e., with no logging information.
-?	Displays help information on this command. The internal logging that is used for requests to dSPACE Support is not affected.

### Return values

The **CmdLoader** command returns the following values:

Returned Value	Meaning
0	Successful command execution. No error occurred.
1	An error occurred during the loading of an application.
2	The platform detected an error during the execution of an application.

### Examples

The following examples show how to use the **CmdLoader** command:

Syntax	Action
<b>CmdLoader -ra</b>	Registers the unregistered platforms in the recent hardware configuration.
<b>CmdLoader -p VEOS veos_demo.sdf</b>	Downloads the OSA that is specified in the SDF file to a platform named VEOS and starts it.

You can find further examples on using the **CmdLoader** in  
`<InstallationPath>/demos/MAPort/CmdLoader`.

---

**Related topics****Basics**

[Introduction to the dSPACE XIL API .NET on Linux.....](#) 122

**HowTos**

[How to Load and Start a Simulation Application.....](#) 127

[How to Register a Platform.....](#) 125

# Limitations and Migration Aspects When Working with XIL API .NET on Linux

## Where to go from here

## Information in this section

[Limitations for Working with XIL API .NET on Linux.....](#) 136

Provides information on the limitations for working with XIL API .NET on Linux.

[Aspects of Migrating From Windows to Linux.....](#) 137

Provides information on differences of the XIL APIs concerning the operation systems.

## Limitations for Working with XIL API .NET on Linux

### General limitations

**Restricted platform type** dSPACE XIL API .NET on Linux supports the following platforms:

- VEOS
- SCALEXIO (SCALEXIO system with SCALEXIO Processing Unit or DS6001 Processor Board)

**No XIL API Framework support** dSPACE XIL API .NET on Linux supports only access to an XIL API Testbench.

**Restricted XIL API Testbench ports** dSPACE XIL API .NET on Linux does not support access to the following port types:

- ECU
- Network
- Diagnostics
- Electrical error simulation (EES)

**Restricted XIL API Testbench objects** dSPACE XIL API .NET on Linux does not support access to objects of the following types:

- Signal
- SignalGenerator



- Symbol
- TargetScript

**No MATLAB support** MATLAB is not supported as a client.

#### Limitations when using MAPort

**No stimulus support** The dSPACE XIL API .NET MAPort implementation on Linux does not support stimuli.

## Aspects of Migrating From Windows to Linux

#### Differences in accessing an XIL API .NET server depending on the operating system

If you want to migrate an application for an XIL API .NET on Windows server to work with dSPACE XIL API .NET on Linux, you have to consider some differences between the two APIs.

**Differences in addressing the XIL API .NET server** How the interfaces of the XIL API .NET server are addressed depends on the operating system:

- In Windows, you can access the XIL API .NET server via the `clr` Python module. This way, you can address the C# .NET interfaces that are provided according to the ASAM XIL API standard. The access to the XIL API .NET server is initialized as follows:

```
import clr
import sys, os
clr.AddReference("ASAM.XIL.Implementation.TestbenchFactory,
    Version=2.1.0.0,
    Culture=neutral,
    PublicKeyToken=fc9d6585b27d387")
clr.AddReference("ASAM.XIL.Interfaces, Version=2.1.0.0, Culture=neutral,
    PublicKeyToken=bf471dffa114ae984")
from ASAM.XIL.Implementation.TestbenchFactory.Testbench import TestbenchFactory
from ASAM.XIL.Interfaces.Testbench.Common.Error import TestbenchPortException
from ASAM.XIL.Interfaces.Testbench.MAPort.Enum import MAPortState
```

- On Linux, you can access the XIL API .NET server via the Python client natively. The access to the XIL API server is initialized as follows:

```
import sys, os
from ASAM.XIL.Implementation.TestbenchFactory.Testbench import TestbenchFactory
from ASAM.XIL.Interfaces.Testbench.Common.Error.TestbenchPortException import
TestbenchPortException
from ASAM.XIL.Interfaces.Testbench.MAPort.Enum import MAPortState
```

**Different handling of Python data types** If you work in Windows, the list, set, tuple, and dictionary Python data types must be converted to the corresponding .NET data types, such as `Array[int]` or `Dictionary[str, int]` and vice versa, when you use the XIL API interfaces. If you work in Linux, a

conversion is not required. As a consequence, you must use the Python data type-specific functions.

- Example of handling Python data types in Windows:

```
# List
DemoVariablesList = [BatteryVoltage, TurnSignalLever]
DemoVariablesList.extend([TurnSignalFrontLeft, TurnSignalFrontRight])
#The Python list has to be converted to an .net Array DemoCapture.
Variables = Array[str](DemoVariablesList)
...
#Dictionary
DemoDefines = Dictionary[str, str]()
DemoDefines.Add('Lever', TurnSignalLever)
```

For example, to get the number of elements of a list, use the **Count** property.

- Example of handling Python data types in Linux:

```
#List
DemoVariablesList = [BatteryVoltage, TurnSignalLever]
DemoVariablesList.extend([TurnSignalFrontLeft, TurnSignalFrontRight])
DemoCapture.Variables = DemoVariablesList
...
#Dictionary
DemoDefines = {}
DemoDefines['Lever'] = TurnSignalLever
```

For example, to get the number of elements of a list, use the Python **len()** function.

# Appendix

Where to go from here

Information in this section

Limitations and Troubleshooting.....	140
Migrating Python Scripts from Python 3.6 to Python 3.9.....	147
Using the Message Reader API.....	152

## Limitations and Troubleshooting

### Where to go from here

### Information in this section

Limitations.....	140
Troubleshooting.....	145

## Limitations

### General limitations

**No 32-bit support** XIL API .NET applications support only 64-bit XIL API clients.

**No support of A2L variables** dSPACE XIL API .NET does not support the ECU access ports (*ECUMPort* and *ECUCPort*) that are specified in the ASAM AE XIL standard. You cannot access A2L variables with an MAPort implementation.

### Limitations when using Framework

**Only one Framework instance at the same time** It is not possible to use more than one instance of the Framework class simultaneously.

**Not supported mapping types** The StringMapping feature of the Framework, which allows the mapping of string parameters of methods used in the Framework and in the Testbench, is not supported.

**Restricted use of TargetState** Some of the states require additional information or arguments to be set.

Thus, only the following states in a framework configuration can be applied:

- EESPort - eCONNECTED
- MAPort - eSIMULATION\_RUNNING or eSIMULATION\_STOPPED

**Required state of the real-time application** When you work with the Framework implementation of dSPACE XIL API, the **Configure** method of the instantiated MAPort for platform access is always using **forceConfig = False**. The initialization of the Framework stops with an error message, if a real-time application is already running on the specified platform that differs from the real-time application specified in the port configuration file. If no real-time

application is running on the platform, the specified application will be downloaded.

**Not supported Framework features** dSPACE XIL API .NET does not support the following features that are contained in the ASAM standard:

- Framework Acquisition
- Framework Simulation
- Framework Watcher

### Limitations when using MAPort

**Limitations when using 64-bit integer variables** If you use 64-bit integer variables (INT64 and UINT64), you must note the following limitations:

- If you use the scaling attribute in the variable description, 64-bit integer variables are converted to the **double** data type. As a result, the range of valid values is restricted to  $\pm 2^{52}-1$ . Values outside the range will differ from the real values.

#### Note

You must provide appropriate measures in your application when reading or writing 64-bit integer values with scaling outside the valid value range.

- If you use trigger conditions with 64-bit integer values, their evaluation is processed with values converted to the **double** data type. The restricted value range of  $\pm 2^{52}-1$  can lead to missing or erroneous triggers.

#### Note

Do not use 64-bit integer values for trigger conditions.

- Stimulating of 64-bit integer values is not supported. Refer to [Limitations \(Real-Time Testing Guide !\[\]\(4695f05050b0d393767d0512587d4e50\_img.jpg\)](#)).

**Limitations when using DS1104** If you use a DS1104 as real-time platform, the following features are not supported:

- Stimulus handling
- Stop trigger is restricted to software trigger by a **DurationWatcher** object. Using a **ConditionWatcher** object is not supported.
- StopTriggerEvents for the host service are not set.
- Start trigger is restricted to the **posedge** and **negedge** functions of ASAM GES.

### Limitations when using the SignalGenerator

- Only parameters of the **eLOAD\_PARAMETER** type are supported for the SignalDescriptionSet. The **eSTART\_PARAMETER** type is not supported.
- The stimulation of the variables is executed by Real-Time Testing. You have to note therefore the limitations of Real-Time Testing too. Refer to [Limitations \(Real-Time Testing Guide !\[\]\(15cb01d00100e773a50f80002909e9a5\_img.jpg\)](#)).

**Limitations when using the `DataFileSegment`** If you use a `DataFileSegment`, only the following file formats are supported:

- MAT file (from any version of MATLAB)
- MDF 4.x

**Limitations when using the `SignalValueSegment` or `DataFileSegment`**

- The `eFORWARD` interpolation type is not supported by the `SignalValueSegment` and the `DataFileSegment`. An exception is generated if the `eFORWARD` interpolation type is used for stimulation.
- The `SignalValueSegment` only supports values of `FloatVectorValue` data type.

#### Note

The `eLINEAR` interpolation type is supported since dSPACE Release 2016-B. With this interpolation type, a linear interpolation between adjacent data points is performed.

At the same time the behavior of the `eBACKWARD` interpolation type changed:

- Up to and including XIL API 2016-A and Real-Time Testing 3.0, Real-Time Testing's `RM_SAMPLED` data streaming mode was used.
- As of XIL API 2016-B and Real-Time Testing 3.1, Real-Time Testing's `RM_BACKWARD` data streaming mode is used. The mode complies with the specification of the `eBACKWARD` interpolation method defined in the ASAM AE XIL API standard.

For details on the difference between the two interpolation methods, refer to [MatFile Class Description \(Real-Time Testing Library Reference\)](#).

**No support for retriggering of the `Capture` class** The `Capture` class of the `MAPort` does not support retriggering. Access to the `Retriggering` property leads to an exception.

**Same duration unit for start and stop trigger** If you have specified `Capture.SetStartTrigger` and `Capture.SetStopTrigger` with different duration units, the setting of the last called trigger is used. That means, that the entire capture is based on samples or seconds.

**Note**

Because the converted value of the replaced duration setting will not match the configured timing behavior, we recommend to use the same duration unit for all capture settings.

**CaptureResult.Open does not provide data frame events** The information on the eDATAFRAMESTART and eDATAFRAMESTOP events are not available when you open a CaptureResult. They are only available if you use CaptureResultReader.Load to get the entire content of the captured MF4 file.

**Limitation of characters that can be used as define in a trigger condition** A define in the trigger condition of a ConditionWatcher can consists only of the following characters:

- a-z
- A-Z
- 0-9
- \_

The first character must not be a number.

**Supported scaling methods** The ASAM MDF standard contains multiple methods for converting the raw values to physical values with a physical unit. Only the following conversion types are supported by the dSPACE XIL API implementation:


- 1:1 conversion
- Linear conversion
- Rational conversion formula
- Algebraic conversion

**Enums in variable description file** Textual representations of enumerations in the variable description file are not supported. Use the values instead. For example, if you have specified 0=Off and 1=On, use 0 and 1 instead of Off and On.

**Not supported string data types** The following types specified for the ValueContainer are not supported:

- IStringValue
- IStringVectorValue
- IStringMatrixValue

Therefore, string data types captured in MF4 files are not supported.

This document contains only limitations relevant to dSPACE XIL API .NET. For further limitations when you use ASAM MDF files, refer to [Limitations for ASAM MDF Files \(Version 4.x\)](#) (ControlDesk Measurement and Recording ).

## Limitations when using EESPort

**Only manual and software trigger types supported** The ASAM AE XIL API standard provides three trigger types to start an error set:

- Manual trigger
- Software trigger
- Hardware trigger

The dSPACE XIL API implementation only supports the manual trigger and the software trigger. If you have specified a hardware trigger in your test application, an error message will be displayed.

**Not supported error categories** The error categories supported by a simulator depend on the signal characteristics. In your signallist, the supported error categories are listed in the `Faults_ECU` column. The numbers must be mapped to the hardware-specific failure classes, refer to [Failure Classes \(dSPACE XIL API Reference\)](#).

**Using mixed error categories with SCALEXIO** Multi-pin errors must be specified with at least three signals. A multi-pin error specified with two signals will be automatically realized as a regular *Short to Pin* error. Using mixed error categories is not supported.

If you want to use multi-pin errors and *Short to Pin* errors at the same time, you have to explicitly specify the fail rails for the multi-pin errors to be used.

**Maximum number of error sets** An error configuration can have a maximum of 500 error sets.

### Note

If you use ConditionWatcher objects as software triggers, the error configuration contains Condition and Defines entries which are all passed to the RTT (Real-Time Testing) sequence as sequence arguments. When you use SCALEXIO or VEOS, the size of the sequence arguments is limited to 10 kb. If the argument size exceeds 10 kb, the following exception is thrown:

**RSM error. Size of arguments too long.**

To reduce the argument size, split the error sets into multiple error configurations.

**Source and Measurement not supported** Source and measurement errors are not supported.

**Load type restrictions** Signals belonging to the following boards do not support `LoadType.ewITHOUT_LOAD`.

- DS293
- DS749
- DS789
- DS791
- DS793



**Minimum duration of a Dynamic or LooseContact error** The minimum duration of a Dynamic or LooseContact error is 1 ms. The minimum duration difference between two Dynamic or LooseContact errors in an error set must be therefore greater than 1 ms.

**Minimum duration of errors** Errors in an error set should be greater than 30 ms.

**No write-protection using Python** When an EESPort object is created in Python, the **Signals** and **LoadTypes** properties of the object are not read-only and new signals and/or load types can be inserted into the list.

**Restrictions when using High Current FIU** You have to note the following restrictions:

- A short circuit to ground or battery voltage can be set only as a single failure in the whole High Current FIU system.
- A short circuit to ground or battery voltage cannot be set simultaneously with an interrupt or short between pins in the High Current FIU system.
- A short circuit to ground or battery voltage on a high current pin and a short between pins on DS291 pins cannot be set simultaneously.

## Troubleshooting

---

### Solving problems

If you have a problem with dSPACE XIL API .NET implementation, look in dSPACE Help and click the **Contact** button. In **Contacting dSPACE Support**, you find instructions what you can do yourself before you contact dSPACE support.

---

### Clash of namespaces when using HIL API and XIL API in Python at the same time

If XIL API .NET via Python for .NET and the dSPACE HIL API Python implementation are used at the same time in a Python interpreter, it may come to clash of the ASAM namespace. In that case, classes of the namespace ASAM.HILAPI can not be accessed any more.

As a workaround, import ASAM.HILAPI first and thereby rename ASAM.HILAPI. Then import the Common Language Runtime and ASAM.XILAPI.

```
# import from ASAM.HILAPI module dSPACE as dS_HILAPI
from ASAM.HILAPI import dSPACE as dS_HILAPI
# now import the Common Language Runtime (PythonNET)
import clr
# now you can import the XILAPI .NET classes from the .NET assemblies via clr.AddReference
# for example:
assemblyString = "ASAM.XIL.Interfaces, Version=2.1.0.0, Culture=neutral, PublicKeyToken=bf471dff114ae984"
clr.AddReference(assemblyString)
assemblyString = "ASAM.XIL.Implementation.TestbenchFactory, Version=2.1.0.0, Culture=neutral,
PublicKeyToken=fc9d65855b27d387"
clr.AddReference(assemblyString)

from ASAM.XIL.Implementation.TestbenchFactory.Testbench import TestbenchFactory
from ASAM.XIL.Interfaces.Testbench.Common.Error import TestbenchPortException

# now you can use the HIL API classes by using the namespace HILAPI
# for example:
hilapiMAPort = dS_HILAPI.MAPort.MAPort(...)
```

### Using an Ethernet-to-RS232 converter

If you use an Ethernet-to-RS232 converter instead of a physical RS232 port of the host PC, software triggers and dynamic errors are not supported. Communication is also time-critical and can cause communication errors.

For example, the following messages might occur for high latencies of the network and the converter:

- *The software trigger condition of the error set was not satisfied in the predefined timeout of 1 second*
- *Hardware module id(s) '4' not found for 'DS5390\_04Pin\_01'*
- *Timeout occurred while waiting for trigger of error set 'ErrorSet4' at index '3'*

dSPACE recommends to use a specific Ethernet-to-RS232 converter. For information on installing and configuring this converter, refer to <https://www.dspace.com/go/eth2rs232>. However, the above mentioned latencies might also occur with this converter.

# Migrating Python Scripts from Python 3.6 to Python 3.9

## Where to go from here

## Information in this section

### [Main Changes in Python 3.9](#)..... 147

The end of life of Python 3.6 is scheduled for the end of December 2021. Therefore, dSPACE switches to Python 3.9 with dSPACE Release 2021-A.

### [Main Changes in Handling Python 3.9 with dSPACE Software](#)..... 148

With Python 3.9, some aspects of handling Python with dSPACE software have changed.

### [General Information on Using Python Installations](#)..... 149

If you work with dSPACE software from dSPACE Release 2020-B or earlier, which supports Python 3.6, and dSPACE software from dSPACE Release 2021-A or later, which supports Python 3.9, both Python versions are installed on the PC and can be used in parallel.

### [Technical Changes](#)..... 150

Migrating from Python 3.6 to Python 3.9 has caused minor changes in the Python API.

## Main Changes in Python 3.9

### Main reason for using Python 3.9 instead of Python 3.6

The end of life of Python 3.6 is scheduled for the end of December 2021. Therefore, dSPACE switches to Python 3.9 with dSPACE Release 2021-A.

### Related documents available on the Python website

For a short overview of the main changes in Python 3.9, refer to [Technical Changes](#) on page 150. For information on all the changes in the Python language and environment from Python 3.6 to Python 3.9, refer to Python Software Foundation at [www.python.org](http://www.python.org).

The Python Software Foundation provides the following documents:

- What's New from Python 3.6 to 3.7:  
<https://docs.python.org/3.7/whatsnew/3.7.html>
- What's New from Python 3.7 to 3.8:  
<https://docs.python.org/3.8/whatsnew/3.8.html>

- What's New from Python 3.8 to 3.9:  
<https://docs.python.org/3.9/whatsnew/3.9.html>

## Related topics

### Basics

Technical Changes..... 150

## Main Changes in Handling Python 3.9 with dSPACE Software

### Introduction

With Python 3.9, some aspects of handling Python with dSPACE software have changed.

### Libraries

The libraries and components used with Python 3.9 and distributed on dSPACE DVDs have changed as shown in the following table.

Package	Python 3.6 (Release 2020-B)	Python 3.9 (Release 2021-A)
comtypes	1.1.7	1.1.7
Core	3.6.8	3.9.1
cycler	0.10.0	0.10.0
future	0.18.2	0.18.2
grpcio	1.29.0	1.34.0
grpcio_tools	1.29.0	1.34.0
kiwisolver	1.2.0	1.3.1
lxml	–	4.6.2
matplotlib	3.2.1	3.3.3
numpy	1.18.4	1.19.3
pillow	7.1.2	8.0.1
pip	20.1.1	20.3.1
protobuf	3.12.2	3.14.0
pycparser	–	2.20
pyglet	1.5.5	1.5.11
pyparsing	2.4.7	2.4.7
pypubsub	4.0.3	4.0.3
Python-dateutil	2.8.1	2.8.1
pythonnet	2.4.2	2.5.3
pytz	2020.1	2020.4

Package	Python 3.6 (Release 2020-B)	Python 3.9 (Release 2021-A)
pywin32	227.10	300.10
scipy	1.4.1	1.5.4
six	1.15.0	1.15.0
wxPython	4.1.0	4.1.1
yapsy	1.12.2	1.12.2

## General Information on Using Python Installations

### Introduction

If you work with dSPACE software from dSPACE Release 2020-B or earlier, which supports Python 3.6, and dSPACE software from dSPACE Release 2021-A or later, which supports Python 3.9, both Python versions are installed on the PC and can be used in parallel.

### Limitations when using Python 3.6 and Python 3.9 in parallel

You can use both Python versions in parallel. However, you must observe the following limitations:

- You can set the file associations for PY and PYW files to only one Python version. This is usually the latest Python version you installed.

#### Note

If you install dSPACE Release 2021-A including SystemDesk, Python 3.6 will be installed last. Therefore, PythonWin 3.6 is registered to open Python files. To set the file associations to PythonWin 3.9, execute the **PythonInstaller.exe** from `<DVDRoot>\Products\Common\Python3.9` again or use the **Edit with PythonWin 3.9** command on the context menu of a Python file.

- Environment variables are used by both Python versions. You must set their values, for example, for PYTHONHOME, to the Python installation you want to work with. For an overview of environment variables set by Python, refer to: <https://docs.python.org/3.9/using/windows.html>.

#### Note

To avoid unintentional effects, which can be difficult to identify, you are recommended to not use environment variables. For the same reason, you should not extend the system path by a Python installation path.

---

**Using dSPACE test automation with both Python versions in parallel**

If a test automation script uses dSPACE Python Modules and you do not want to migrate the script, you have to work with both Python versions in parallel. The dSPACE Python Modules for Python 3.6 are available up to and including Release 2020-B.

**Note**

It is recommended to use only dSPACE software from the same Release when using Python scripts. Using both Python versions in the same application context, such as automating the access to an older version of ControlDesk via AutomationDesk or using AutomationDesk with an activated Real-Time Testing version that does not support Python 3.9, might cause a conflict.

## Technical Changes

---

**Introduction**

Migrating from Python 3.6 to Python 3.9 has caused minor changes in the Python API.

**Note**

Independently of the number of required manual modifications, you must test the migrated code.

---

**Migration issues**

The following changes in Python 3.9 require manual migration of your Python scripts.

**Python.NET: Return value of `IList` type is converted to `PyList`** Since Python.NET 2.4, return values of `System.Collections.Generic.IList` type have been converted to the `PyList` type. Because some methods and properties of an `IList` object are not available with a `PyList` object, dSPACE provided the modified Python.NET 2.4.1 and 2.4.2 packages, which suppressed the conversion.

As of Python.NET 2.5.3, the modifications by dSPACE are no longer provided. You must therefore migrate the methods and properties of an `IList` object to the related methods and properties of a `PyList` object. For example, you must use `len(MyObject)` instead of `MyObject.Count`.

**Note**

The created Python list is a copy of the .NET object. Modifications to the Python list therefore have no effect on the associated .NET object. To apply the modifications to the .NET object, assign the Python list afterward.

**Note**

All dSPACE XIL API .NET methods and properties returning `ICollection` types must be migrated accordingly except for the `ASAM.XIL.Interfaces.Testbench.Common.Capturing.ICapture.Variables` property that uses the `ObservableList` type.

**ctypes: Loading library after changing the PATH variable** Loading a library via `ctypes` can fail if the path to the library was added to the `PATH` variable beforehand.

For loading a library in a safe manner, use `ctypes.WinDLL(NameOfDll, winmode=8)` instead of `ctypes.windll.LoadLibrary(NameOfDll)`.

## Using the Message Reader API

---

### Where to go from here

### Information in this section

<a href="#">Introduction to the Message Reader API.....</a>	<a href="#">152</a>
<a href="#">dSPACE.Common.MessageHandler.Logging Reference.....</a>	<a href="#">158</a>

## Introduction to the Message Reader API

---

### Where to go from here

### Information in this section

<a href="#">Reading dSPACE Log Messages via the Message Reader API.....</a>	<a href="#">152</a>
You can read log messages of the dSPACE Log via the Message Reader API.	
<a href="#">Supported dSPACE Products and Components.....</a>	<a href="#">154</a>
Provides an overview of all dSPACE products and components whose messages you can access via the Message Reader API.	
<a href="#">Example of Reading Messages with Python.....</a>	<a href="#">155</a>
You can read the log messages via Python. You can combine multiple filters to display only messages according to your specifications.	
<a href="#">Example of Reading Messages with C#.....</a>	<a href="#">157</a>
You can read the log messages via C#. You can combine multiple filters to display only messages according to your specifications.	

## Reading dSPACE Log Messages via the Message Reader API

---

### Introduction

You can read log messages of the dSPACE Log via the Message Reader API.

---

### dSPACE Log

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

The dSPACE Log is saved as a collection of binary message log files. These files are created when a dSPACE product is running. A single run of a dSPACE product is called a *log session*.



**Note**

If the maximum file size for the binary message log file is reached, messages at the beginning of the dSPACE Log might get deleted. Contact dSPACE Support to solve this.

**Message Reader API**

You can use the Message Reader API to access all binary message log files of the dSPACE Log. You can combine multiple filters to display only log messages according to your specifications. For example, you can configure the Message Reader API to display only log messages from a specific dSPACE product.

The Message Reader API is available as of dSPACE Release 2020-A. For information on the dSPACE products and components that support the Message Reader API, refer to [Supported dSPACE Products and Components](#) on page 154.

**dSPACE.Common.MessageReader.dll** The Message Reader API is implemented by the `dSPACE.Common.MessageReader.dll` file.

The `dSPACE.Common.MessageReader.dll` for dSPACE XIL API .NET is located in the following subfolders:

- `MAPort\Main\bin`
- `EESPort\Main\bin`
- `PlatformManagementAPI\Main\bin`

For information on the interfaces, properties and methods, refer to [dSPACE.Common.MessageHandler.Logging Reference](#) on page 158.

**Supported dSPACE Releases**

The Message Reader API lets you access log messages written by dSPACE products since dSPACE Release 2016-B.

**Message Reader API change in dSPACE Release 2021-A**

There is a migration issue specific to the Message Reader API. The issue occurs if you use the API with Python. The issue was caused by the migration to Python 3.9/pythonnet 2.5.3 with dSPACE Release 2021-A.

There is no migration issue to consider if you use the API with C#.

**Specifying a product filter** As of dSPACE Release 2021-A, the **Products** property of the **MessageReaderSettings** class can no longer be used to set the list of products for which to filter in the log sessions. The Message Reader API provides the **SetProducts** method for this purpose.

The following table shows how to specify a product filter before and after migration:

**Using Message Reader API of ...****... dSPACE Release 2020-B and Earlier (Python 3.6)**

```
# Specify products whose messages to read:
Settings = MessageReaderSettings()
```

**... dSPACE Release 2021-A and Later (Python 3.9)**

```
# Specify products whose messages to read:
Settings = MessageReaderSettings()
Settings.SetProducts(['ControlDesk', 'AutomationDesk'])
```

Using Message Reader API of ...	
... dSPACE Release 2020-B and Earlier (Python 3.6)	... dSPACE Release 2021-A and Later (Python 3.9)
Settings.Products.Add('ControlDesk')	
Settings.Products.Add('AutomationDesk')	

## Related topics

### Basics

[Supported dSPACE Products and Components..... 154](#)

### Examples

[Example of Reading Messages with C#..... 157](#)  
[Example of Reading Messages with Python..... 155](#)

### References

[MessageReaderSettings Class..... 163](#)

## Supported dSPACE Products and Components

### Supported dSPACE products and components

You can use the Message Reader API to access messages from the following dSPACE products and components:

- ASM KnC
- AutomationDesk
- Bus Manager (stand-alone)
- cmdloader
- ConfigurationDesk
- Container Management
- ControlDesk
- dSPACE AUTOSAR Compare
- dSPACE XIL API .NET Implementation
- Firmware Manager
- ModelDesk
- MotionDesk
- Real-Time Testing
- RTI Bypass Blockset
- SYNECT client
- SystemDesk
- TargetLink Property Manager
- VEOS

**Related topics****Basics**

[Reading dSPACE Log Messages via the Message Reader API.....](#) 152

## Example of Reading Messages with Python

**Introduction**

You can read the log messages via Python by using the `clr` module. You can combine multiple filters to display only messages according to your specifications.

**Referencing a message reader assembly**

You have to reference a `dSPACE.Common.MessageReader.dll` assembly. For information on the location of the assembly, refer to [dSPACE.Common.MessageReader.dll](#) on page 153.

In the following examples it is assumed that the dSPACE Installation Manager is installed and that the message reader assembly is installed in `C:\Program Files\Common Files\dSPACE\InstallationManager\bin`.

The following code references and imports the message reader assembly.

```
# Insert path of message log file access assembly:
import sys
AssemblyPath = r'C:\Program Files\Common Files\dSPACE\InstallationManager\bin'
if not sys.path.count(AssemblyPath):
    sys.path.insert(1, AssemblyPath)

# Add reference to assembly and import it:
import clr
clr.AddReference('dSPACE.Common.MessageReader')
from dSPACE.Common.MessageHandler.Logging import *
```

**Reading all messages**

The following example reads all existing message log files and prints all messages via Python. It is assumed that the message reader assembly is referenced and imported. Refer to [Referencing a message reader assembly](#) on page 155.

```
# Create message reader and print text of each message:
Reader = MessageReader(None)
for Message in Reader.ReadMessages():
    print(Message.MessageText)
Reader.Dispose()
```

**Filtering messages by severity, product, and session**

The following example reads and prints messages with a severity of `Error`, `SevereError`, or `SystemError`. Also, only messages of the last sessions of `ControlDesk` and `AutomationDesk` are read and printed. It is assumed that the message reader assembly is referenced and imported. Refer to [Referencing a message reader assembly](#) on page 155.

```

# Define error severities:
SEVERITY_ERROR = 3
SEVERITY_SEVERE_ERROR = 4
SEVERITY_SYSTEM_ERROR = 5

# Configure products and sessions whose messages to read:
Settings = MessageReaderSettings()
Settings.MaximalSessionCount = 1
Settings.SetProducts(['ControlDesk', 'AutomationDesk'])

# Create message reader and print text of each error message:
Reader = MessageReader(Settings)
for Message in Reader.ReadMessages():
    # Print error messages only:
    if Message.Severity == SEVERITY_ERROR or \
       Message.Severity == SEVERITY_SEVERE_ERROR or \
       Message.Severity == SEVERITY_SYSTEM_ERROR:
        print('%s: %s' % (Message.Session.ProductName, Message.MessageText))
Reader.Dispose()

```

### Note

The ReadMessages method returns an enumerator which must either read all messages or must be disposed when no longer used. It is not possible to use two enumerators interleaved, only one enumerator may read messages at a time. Refer to [MessageReader Class](#) on page 162.

## Filtering messages by time

Times are given by .NET DateTime objects. Times are given as UTC times (Coordinated Universal Time). You can obtain the current UTC time by `System.DateTime.UtcNow`.

The following example reads all messages after a certain start time. It is assumed that the message reader assembly is referenced and imported. Refer to [Referencing a message reader assembly](#) on page 155.

```

import System
Settings = MessageReaderSettings()
Settings.MessageTimeAfter = System.DateTime.UtcNow # Read messages after now

# Create message reader and print time and text of each message:
Reader = MessageReader(Settings)
for Message in Reader.ReadMessages():
    print('%s: %s' % (Message.UtcTimeStamp, Message.MessageText))
Reader.Dispose()

```

## Related topics

### Basics

[Reading dSPACE Log Messages via the Message Reader API..... 152](#)

Supported dSPACE Products and Components.....	154
References	
MessageReaderSettings Class.....	163

## Example of Reading Messages with C#

### Introduction

You can read the log messages via C#. You can combine multiple filters to display only messages according to your specifications.

### Referencing a message reader assembly

You have to reference a `dSPACE.Common.MessageReader.dll` assembly. For information on the location of the assembly, refer to [dSPACE.Common.MessageReader.dll](#) on page 153.

### Reading all messages

The following example reads all existing message log files and prints the messages:

```
using dSPACE.Common.MessageHandler.Logging;
...

// Create message reader and print text of each message:
using (MessageReader reader = new MessageReader(null))
{
    foreach (message in reader.ReadMessages())
    {
        Console.WriteLine(message.MessageText);
    }
}
```

### Filtering messages by severity, product, and session

The following example reads and prints messages with a severity of `Error`, `SevereError`, or `SystemError`. Also, only messages of the last sessions of `ControlDesk` and `AutomationDesk` are read and printed.

```

using dSPACE.Common.MessageHandler.Logging;
...

// Read the last log sessions of ControlDesk and AutomationDesk only:
MessageReaderSettings settings = new MessageReaderSettings();
settings.MaximalSessionCount = 1;
settings.Products.Add("ControlDesk");
settings.Products.Add("AutomationDesk");

using (MessageReader reader = new MessageReader(settings))
{
    foreach (ILogMessage message in reader.ReadMessages())
    {
        // Print error messages only:
        if (message.Severity == Severity.Error
            || message.Severity == Severity.SevereError
            || message.Severity == Severity.SystemError)
        {
            Console.WriteLine(message.Session.ProductName + ": " + message.MessageText);
        }
    }
}

```

**Note**

The ReadMessages method returns an enumerator which must either read all messages or must be disposed when no longer used. It is not possible to use two enumerators interleaved, only one enumerator may read messages at a time. Refer to [MessageReader Class](#) on page 162.

**Related topics****Basics**

Reading dSPACE Log Messages via the Message Reader API.....	152
Supported dSPACE Products and Components.....	154

**References**

MessageReaderSettings Class.....	163
----------------------------------	-----

## dSPACE.Common.MessageHandler.Logging Reference

**Where to go from here****Information in this section**

ILogMessage Interface.....	159
To access information about a message as written to a log file.	

<a href="#">ILogSession Interface</a> .....	160
To access information about a message log session.	
<a href="#">MessageReader Class</a> .....	162
To read serialized messages written by dSPACE products.	
<a href="#">MessageReaderSettings Class</a> .....	163
To define the settings of a message reader.	
<a href="#">Severity Enumeration</a> .....	165
To specify the severity of a message.	

## ILogMessage Interface

**Namespace** `dSPACE.Common.MessageHandler.Logging`

**Description** To access information about a message as written to a log file.

**Properties** The element has the following properties:

Name	Description	Get/Set	Type
IsStartMessage	Gets a value indicating whether the message is a session start message.	Get	<i>Boolean</i>
IsStopMessage	Gets a value indicating whether the message is a session stop message.	Get	<i>Boolean</i>
MainModuleNumber	Gets the main module number of the message.	Get	<i>Integer</i>
MessageCode	Gets the error code of the message.	Get	<i>Integer</i>
MessageText	Gets the text of the message.	Get	<i>String</i>
ModuleName	Gets the module name of the message.	Get	<i>String</i>
Session	Gets the log session which issued the message.	Get	ILogSession (refer to <a href="#">ILogSession Interface</a> on page 160)
Severity	Gets the severity of the message.	Get	Severity (refer to <a href="#">Severity Enumeration</a> on page 165)
SubmoduleNumber	Gets the submodule number of the message.	Get	<i>Integer</i>
ThreadId	Gets the thread ID of the submitting thread.	Get	<i>Integer</i>
TimeStamp	Gets the time when the message was submitted. Given as local time in the time zone of the session.	Get	<i>DateTime</i>
UtcTimeStamp	Gets the time when the message was submitted in UTC time.	Get	<i>DateTime</i>

**Methods** The element has no methods.

## Related topics

### Basics

[Reading dSPACE Log Messages via the Message Reader API..... 152](#)

### Examples

[Example of Reading Messages with C#..... 157](#)  
[Example of Reading Messages with Python..... 155](#)

### References

[ILogSession Interface..... 160](#)  
[Severity Enumeration..... 165](#)

## ILogSession Interface

**Namespace** `dSPACE.Common.MessageHandler.Logging`

**Description** To access information about a message log session.

**Properties** The element has the following properties:

Name	Description	Get/Set	Type
CloseTime	Gets the time when the session was closed. Returns an undefined time (0, DateTimeKind.Unspecified) if the session is still open or was not closed successfully. Given as local time in the time zone of the session.	Get	<i>DateTime</i>
IsOpen	Gets a value indicating whether the session is still open. If true, the session is still open and new messages can be written.	Get	<i>Boolean</i>
IsValid	Gets a value indicating whether the session is valid. A session can become invalid if its log files are corrupted.	Get	<i>Boolean</i>
MetaData	Gets the products metadata as read from log file session info.	Get	<i>Dictionary&lt; String, String &gt;</i>
ProcessId	Gets the process ID of the log session.	Get	<i>Integer</i>



Name	Description	Get/Set	Type
ProductName	Gets the product name of the log session.	Get	<i>String</i>
SessionId	Gets the ID of the log session. This ID is unique in the context of its session reader.	Get	<i>Integer</i>
StartTime	Gets the sessions start time. Given as local time in the time zone of the session.	Get	<i>DateTime</i>
TimezoneName	Gets the standard time zone name of the session.	Get	<i>String</i>
TimezoneOffset	Gets the time zone offset of the session relative to UTC.	Get	<i>TimeSpan</i>
UtcCloseTime	Gets the time when the session was closed as UTC time. Returns an undefined time (0, <i>DateTimeKind.Unspecified</i> ) if the session is still open or was not closed successfully.	Get	<i>DateTime</i>
UtcStartTime	Gets the start time of the log session as UTC time.	Get	<i>DateTime</i>

## Methods

The element has the following methods:

Name	Description	Parameter <sup>1)</sup>	Returns
ToSessionTime	Converts UTC time to time zone used when the session was written.	<ul style="list-style-type: none"> <li>&lt;DateTime&gt; <b>utcTime</b>: Specifies the UTC time to convert.</li> </ul>	Time in the time zone of the logging session. <ul style="list-style-type: none"> <li>DateTime</li> </ul>

<sup>1)</sup> <Type> Name: Description

## Related topics

### Basics

[Reading dSPACE Log Messages via the Message Reader API..... 152](#)

### Examples

[Example of Reading Messages with C#..... 157](#)  
[Example of Reading Messages with Python..... 155](#)

## MessageReader Class

**Description** To read serialized messages written by dSPACE products.

**Constructor** The element has the following constructor:

Name	Description	Parameter <sup>1)</sup>	Returns
MessageReader	Initializes a new instance of the MessageReader class.	<ul style="list-style-type: none"><li>▪ &lt;MessageReaderSettings&gt;<sup>2)</sup> <b>settings</b>: Settings which allow to specify which sessions and messages are read. Can be null, causing all existing log files to be read.</li></ul>	None

<sup>1)</sup> <Type> Name: Description

<sup>2)</sup> Refer to [MessageReaderSettings Class](#) on page 163

**Properties** The element has no properties.

**Methods** The element has the following methods:

Name	Description	Parameter <sup>1)</sup>	Returns
Dispose	Performs application-specific tasks associated with freeing, releasing, or resetting unmanaged resources.	None	None

Name	Description	Parameter <sup>1)</sup>	Returns
ReadMessages	<p>Reads the messages written to the log files of the sessions up to now.</p> <p>The messages are returned in chronological order according to their time stamps.</p> <div> <b>Note</b> <p>The ReadMessages method returns an enumerator which must either read all messages or must be disposed when no longer used. It is not possible to use two enumerators interleaved, only one enumerator may read messages at a time.</p> </div>	None	<p>Messages read from log file.</p> <ul style="list-style-type: none"> <li>▪ IEnumerable&lt; ILogMessage (refer to <a href="#">ILogMessage Interface</a> on page 159) &gt;</li> </ul>

<sup>1)</sup> <Type> Name: Description

## Related topics

### Basics

[Reading dSPACE Log Messages via the Message Reader API.....](#) 152

### Examples

[Example of Reading Messages with C#.....](#) 157

[Example of Reading Messages with Python.....](#) 155

### References

[MessageReaderSettings Class.....](#) 163

## MessageReaderSettings Class

### Description

To define the settings of a message reader.

Used to filter the log sessions and messages read.

**Constructor**

The element has the following constructor:

Name	Description	Parameter <sup>1)</sup>	Returns
MessageReaderSettings	Initializes a new instance of the MessageReaderSettings class.	None	None

<sup>1)</sup> <Type> Name: Description**Properties**

The element has the following properties:

Name	Description	Get/Set	Type
DirectoryNames	Gets a list of specific directory names from which to read log files. If the list is empty, all standard directories are searched for log files.	Get	<i>List&lt; String &gt;</i>
MaximalSessionCount	Gets or sets the maximal number of log sessions read for each product. If the count is a positive number n, only the last n sessions are read. If the count is not positive, an unlimited number of sessions is read. The default value is zero, i.e., unlimited.	Get/Set	<i>Integer</i>
MessageTimeAfter	Gets or sets the minimal time for which messages are read, given as UTC time. Only messages submitted after the message time are read. The message time may be in the past. The message time must be given as valid UTC time. The default time is undefined, i.e., each message time is allowed.	Get/Set	<i>DateTime</i>
Products	Gets the list of product names for which to read log sessions. If the list is empty sessions of all products are read.	Get	<i>List&lt; String &gt;</i>
StartTimeAfter	Gets or sets the minimal start time for which sessions are read, given as UTC time. Only sessions which started after the start time are read. The start time may be in the past. The start time must be given as valid UTC time. The default time is undefined, i.e., each start time is allowed.	Get/Set	<i>DateTime</i>

**Methods**

The element has the following methods:

Name	Description	Parameter <sup>1)</sup>	Returns
SetDirectoryNames	Sets the list of specific directory names from which to read log files. You do not have to specify a list. If the list is empty, all standard directories are searched for log files.	<code>&lt;string[]&gt; names</code> : Array of directory names.	None
SetProducts	Sets the list of product names for which to read log sessions.	<code>&lt;string[]&gt; products</code> : Array of product names.	None

<sup>1)</sup> <Type> Name: Description**Related topics****Basics**
[Reading dSPACE Log Messages via the Message Reader API..... 152](#)
**Examples**
[Example of Reading Messages with C#..... 157](#)  
[Example of Reading Messages with Python..... 155](#)

## Severity Enumeration

**Description**

To specify the severity of a message.

**Enumeration values**

The enumeration has the following values:

Value	Name	Description
0	Trace	A trace message. Trace messages are usually not created. It depends on the host application if it is possible to configure the message handler to create trace messages.
1	Info	An information message.
2	Warning	A warning message.
3	Error	An error message.
4	SevereError	A severe error message.
5	SystemError	A system error message.
6	Question	A question message.
7	Advice	An advice message.

## Related topics

### Basics

[Reading dSPACE Log Messages via the Message Reader API..... 152](#)

### Examples

[Example of Reading Messages with C#..... 157](#)  
[Example of Reading Messages with Python..... 155](#)

**A**

activating errors 114

**B**

basics

- failure classes 76
- failure simulation 75

**C**

central FIU 97  
 comment area 101  
 Common Program Data folder 9  
 configuring signal files 106  
 creating dSPACE EESPort configuration files  
   manually 58  
   via API 62

**D**

deactivating errors 117  
 Documents folder 9  
 DS1450 bus FIU Board  
   migrating 105  
 DS282 Load module 87  
 DS289MK RSim module 87  
 DS291 FIU module 86  
 DS293 FIU module 87  
 DS5355 High Current FIU Controller Card 95  
 DS5390 High Current FIU 95  
 DS749 FIU module 90  
 DS789 Sensor FIU module 90  
 DS791 Actuator FIU module 92  
 DS793 Sensor FIU module 93  
 dSPACE XIL API .NET Implementation  
   license 12  
 dSPACE.Common.MessageReader.dll 152

**E**

electrical error simulation  
   hardware 96  
   multi-pin errors 82  
 electrical error simulation basics  
   for SCALEXIO FIU 78  
   for systems without SCALEXIO FIU 76  
 Electrical error simulation concept  
   SCALEXIO system 96  
 electrical errors 80  
 error configuration  
   specifying 108  
 evaluating signal files 107

**F**

failrail segment switch 98  
 failrails 98  
 failure classes  
   basics 76  
 failure insertion units 85  
 failure routing unit 98

failure simulation  
   basics 75  
 FIU concept  
   SCALEXIO system 96  
 format area 102

**L**

license  
   dSPACE XIL API .NET Implementation 12  
 Local Program Data folder 9  
 loose contacts 118  
   simulating 118

**M**

MAPort  
   capturing variables 42  
   configuring 40  
   forceConfig 40  
   reading variables 41  
   stimulating variables 43  
   supported platforms 40  
   variable path formats 41  
   writing variables 41  
 Message Reader API 152  
 multi-pin errors 82

**P**

pulsed switching 81, 118

**S**

signal files 99  
   configuring 106  
   evaluating 107  
   structure 100  
 signallist area 103  
 simulating 118  
 simulating loose contacts 118  
 simulating switch bouncing 118  
 specifying an error configuration 108  
 structure  
   signal files 100  
 switch bouncing 118

**U**

unstable pin failures 118  
 updating errors 116

