

AutomationDesk

Accessing Simulation Platforms

For AutomationDesk 6.5

Release 2021-A – May 2021

dSPACE

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

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.

© 2017 - 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	13
New Features	17
New Basic Features for Accessing Simulation Platforms.....	17
Basics and Instructions	19
Managing Platforms and Simulation Applications.....	20
Basics of Platforms and Simulation Applications.....	20
Basics on Platforms.....	21
Basics on Simulation Applications.....	22
Registering and Managing dSPACE Platforms.....	26
How to Register a dSPACE Platform.....	27
Synchronized Platform Management with Several dSPACE Products.....	29
Accessing the SCALEXIO or MicroAutoBox III System with Multiple dSPACE Products Simultaneously.....	31
Specifics on Working with Multiprocessor Systems with Optional Processors.....	34
Updating and Repairing the Firmware of dSPACE Real-Time Hardware.....	35
Basics on Firmware.....	36
How to Prepare the Firmware Update.....	36
How to Update Firmware.....	38
How to Repair Firmware.....	39
Managing Simulation Applications Manually.....	41
Basics on the Platform Manager.....	41
How to Load and Start a Simulation Application Manually.....	43
How to Stop and Restart a Simulation Application Manually.....	45
How to Terminate a Simulation Application Manually.....	46
Automating Simulation Application Management.....	48
Basics on the Platform Management Library.....	49
How to Automate the Loading and Starting of Simulation Applications.....	51
How to Automate the Stopping and Restarting of Simulation Applications.....	53
How to Automate the Termination of Simulation Applications.....	55

Working with an XIL API Framework.....	57
Basics on Working with an XIL API Framework.....	57
How to Create a Framework Configuration.....	59
How to Add a Port to a Framework Configuration.....	61
How to Create a Mapping File and Add it to a Framework Configuration.....	64
How to Specify the Variable Mapping of a Framework Configuration.....	66
How to Initialize an XIL API Framework.....	68
How to Migrate Projects to Use XIL API Framework.....	70
How to Work with a Third-Party XIL API Framework.....	71
Accessing Simulation Platforms via the XIL API Convenience Library.....	75
Basics on Accessing Simulation Platforms via the XIL API Convenience Library.....	76
How to Build a Basic Sequence for Accessing Simulator Variables.....	78
How to Download and Start a Simulation Application.....	81
How to Make Simulator Variables Available.....	84
How to Write Variable Values.....	87
How to Read Variable Values.....	91
How to Capture Data.....	96
How to Add Plots of the Captured Data to your Report.....	102
How to Release Resources After Data Capturing.....	105
How to Specify Signals for Stimulating.....	106
How to Stimulate Simulator Variables.....	108
How to Release Resources after Variable Stimulation.....	112
How to Access Platforms via a Third-Party XIL API Server.....	114
Accessing Simulation Platforms via XIL API.....	117
Basics of the XIL API Library Elements.....	118
Overview of the XIL API Library Elements.....	119
Example of an XIL API Sequence.....	121
How to Prepare a Project for Using the XIL API Library.....	123
How to Initialize a Model Access Port.....	126
How to Make Simulator Variables Available to XIL API Library Automation Blocks.....	127
How to Write and Read Variables.....	129
How to Get Application Properties.....	133
How to Capture Data.....	134
Basics on Stimulating Variables via AutomationDesk.....	137
How to Migrate a Variable Pool's Data Container to an XIL API Mapping Data Object.....	138

Reference Information	141
Automation Blocks.....	142
Platform Management.....	142
GetPlatform.....	144
GetPlatformManagement.....	145
GetRegisteredPlatformNames.....	146
GetSimulationApplication.....	147
GetSimulationState.....	148
LoadSimulationApplication.....	150
RefreshPlatformConfiguration.....	151
StartSimulation.....	152
StopSimulation.....	153
UnloadSimulationApplication.....	154
XIL API Framework.....	155
CheckValues.....	156
GetValues.....	157
SetValues.....	158
TaskName (Data Object).....	159
Variables (Data Object).....	159
Vendor (Data Object).....	160
XIL API Convenience (Model Access).....	160
Main Elements.....	161
InitMAPort.....	162
Read.....	164
ReadValues.....	165
ReleaseMAPort.....	167
Write.....	168
WriteValues.....	169
Capture.....	171
AddCustomPlotsToReport.....	172
AddPlotsToReport.....	173
ConfigureStartCondition.....	175
ConfigureStopCondition.....	177
ConfigureStopDuration.....	179
GetCaptureResult.....	181
GetCaptureState.....	182
GetDictionaryFromCaptureResult.....	184
InitializeCapture.....	186

ReleaseCapture.....	188
StartCapture.....	189
StopCapture.....	190
StreamToDisk.....	191
GetRecordedData.....	191
StartStreamToDisk.....	193
SignalGenerator.....	194
ConfigureSignalSegment.....	195
DestroyOnTarget.....	196
InitializeSignalGenerator.....	197
LoadToTarget.....	200
SaveSignalGenerator.....	201
StartSignalGenerator.....	202
ReleaseSignalGenerator.....	203
SimulationApplication.....	203
LoadSimulationApplication.....	204
StartSimulation.....	205
StopSimulation.....	206
XIL API (Model Access).....	207
Main Elements.....	208
Mapping (Data Object).....	209
PortConfig (Data Object).....	211
Testbench (Data Object).....	211
TestbenchFactory (Data Object).....	212
Common.....	213
Duration.....	214
Duration (Data Object).....	214
DurationFactory (Data Object).....	216
MetaInfo.....	216
ErrorInfo (Data Object).....	217
TaskInfo (Data Object).....	217
VariableInfo (Data Object).....	218
Symbol.....	219
Symbol (Data Object).....	220
SymbolFactory (Data Object).....	221
CaptureResult.....	222
CaptureEvent (Data Object).....	222
CaptureResult (Data Object).....	223

ExtractSignalValue.....	225
GetMetaData.....	226
GetSignalGroupNames.....	227
GetSignalGroupValue.....	228
SetMetaData.....	229
 Capturing.....	230
Capture (Data Object).....	231
CaptureResultReader (Data Object).....	232
CaptureResultWriter (Data Object).....	233
CaptureState (Data Object).....	234
CapturingFactory (Data Object).....	235
ClearConfiguration.....	236
Fetch.....	237
GetCaptureResult.....	239
GetMinBufferSize.....	240
GetState.....	241
GetVariables.....	242
ReleaseCapture.....	243
SetMinBufferSize.....	244
SetStartTriggerCondition.....	245
SetStopTriggerCondition.....	246
SetVariables.....	247
Start.....	248
Stop.....	249
 Script.....	250
Script (Data Object).....	250
TargetScriptFactory (Data Object).....	252
TargetScriptsFileReader (Data Object)	252
 SignalGenerator.....	253
SignalGenerator (Data Object).....	254
SignalGeneratorFactory (Data Object).....	256
SignalGeneratorReader (Data Object).....	257
SignalGeneratorWriter (Data Object).....	257
DestroyOnTarget.....	258
InitSignalGeneratorSTZReader.....	259
InitSignalGeneratorSTZWriter.....	260
LoadSignalGenerator.....	261
LoadToTarget.....	262
ReleaseSignalGenerator.....	263
SaveSignalGenerator.....	264

SetAssignments.....	264
StartSignalGenerator.....	265
Signal.....	266
ScriptParameterInfo (Data Object).....	267
SignalDescriptionSet (Data Object).....	267
SignalFactory (Data Object).....	268
SignalSegment (Data Object).....	269
SignalDescription (Data Object).....	270
SignalDescriptionSetReader (Data Object).....	271
SignalDescriptionSetWriter (Data Object).....	272
ValueContainer.....	273
Attributes (Data Object).....	273
BaseValue (Data Object).....	274
DataType (Data Object).....	275
SignalGroupValue (Data Object).....	276
SignalValue (Data Object).....	277
ValueFactory (Data Object).....	277
InitBaseValue.....	278
WatcherHandling.....	279
Watcher (Data Object).....	279
WatcherFactory (Data Object).....	280
InitConditionWatcher.....	281
InitDurationWatcher.....	284
MAPort.....	285
MAPort (Data Object).....	286
MAPortFactory (Data Object).....	288
MAPortConfiguration (Data Object).....	289
CreateCapture.....	290
CreateSignalGenerator.....	291
Get(DataType).....	292
Get(TaskNames).....	293
Get(VariableNames).....	294
InitMAPort.....	295
IsReadable.....	297
IsWritable.....	298
Read.....	299
ReleaseMAPort.....	300
Write.....	301

Commands And Dialogs.....	302
Platform Manager.....	302
Assembly View.....	305
Clear Flash.....	306
Clear Flash Options - Clear Complete Flash Memory.....	307
Clear Flash Options - Clear Flash Application.....	308
Clear Flash Options - Clear Flight Recorder Data.....	308
Clear Flash Options - Clear Nonvolatile Data.....	309
Clear System.....	310
Collapse.....	310
Create Support Info.....	311
Expand.....	312
Explore Logged Data.....	312
Manage Platforms.....	314
Network View.....	316
Pause.....	317
Platform Manager.....	317
Properties (Platform/Device).....	319
Real-Time Application - Load.....	319
Real-Time Application / Offline Simulation Application - Load.....	320
Real-Time Application / Offline Simulation Application - Load and Start.....	321
Real-Time Application - Load to Flash (DS1006/DS1104/MicroAutoBox II).....	322
Real-Time Application - Load to Flash (DS1007/MicroLabBox/MicroAutoBox III/SCALEXIO).....	323
Real-Time Application - Load to Flash and Start (DS1007/MicroLabBox/MicroAutoBox III/SCALEXIO).....	324
Real-Time Application - Reload.....	325
Refresh Interface Connections.....	325
Refresh Platform Configuration.....	326
Register Platforms.....	326
Reload.....	331
Reload and Start.....	332
Real-Time Application - Reload to Flash.....	332
Reload to Flash.....	333
Reload to Flash and Start.....	334
Show Connected Clients.....	335
Single Step.....	335
Start.....	336
Stop.....	337
Stop RTP.....	337

Stop RTPs.....	338
Unload.....	338
Update Firmware.....	339
XIL API Framework.....	342
Configure (XIL API Framework).....	343
Disable Framework Support.....	344
Edit (XIL API Framework).....	345
Enable Framework Support.....	346
Initialize (XIL API Framework).....	347
Insert (CheckValues).....	348
Insert (GetValues).....	348
Insert (SetValues).....	349
Shutdown (XIL API Framework).....	350
XIL API Convenience.....	350
Edit (TaskName).....	351
Edit (Variables).....	352
Edit (Vendor).....	354
XIL API.....	355
Edit (CaptureState).....	356
Edit (DataType).....	357
Edit (MAPortConfiguration).....	358
Edit as Dictionary.....	360
Export XIL API Mapping.....	361
Import XIL API Mapping.....	362
Insert (XIL API Elements).....	363
Release Capture.....	364
Release MAPort.....	365
Automation	367
Basics on Automating Simulation Platform Access.....	367
Limitations	369
Limitations for Platforms.....	369
Limitations When Using the XIL API Convenience Library.....	370
Limitations When Using the XIL API Library.....	371

Glossary	373
Index	389

About This Document

Contents

This document introduces you to AutomationDesk's means to access simulation platforms.

Required knowledge

Working with AutomationDesk requires:

- Basic knowledge in handling the PC and the Microsoft Windows operating system.
- Basic knowledge in developing applications or tests.
- Basic knowledge in handling the external device, which you control remotely via AutomationDesk.

dSPACE provides trainings for AutomationDesk. For more information, refer to <https://www.dspace.com/go/trainings>.

Legal information

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.
	Indicates a hazard that, if not avoided, could result in property damage.
	Indicates important information that you should take into account to avoid malfunctions.
	Indicates tips that can make your work easier.
	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files

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

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

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

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

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.

New Features

Introduction	Information on enhancements and new features of AutomationDesk.
---------------------	---

New Basic Features for Accessing Simulation Platforms

New features and migration	For information on new features of the current version of AutomationDesk, refer to New Features of AutomationDesk 6.5 (New Features and Migration ).
-----------------------------------	--

Basics and Instructions

Where to go from here

Information in this section

Managing Platforms and Simulation Applications.....	20
AutomationDesk provides automation capabilities to manage real-time applications on dSPACE real-time hardware and offline simulation applications on VEOS.	
Working with an XIL API Framework.....	57
Provides basic information for using an XIL API Framework.	
Accessing Simulation Platforms via the XIL API Convenience Library.....	75
Provides basic information on using the XIL API Convenience library for automated platform access.	
Accessing Simulation Platforms via XIL API.....	117
Provides basic information on automating access to XIL API.	

Managing Platforms and Simulation Applications

Introduction

Before you can run an automated test with platform access, you have to register the dSPACE platform and manage the loading and execution of the simulation applications on it manually or in an automated way.

Where to go from here

Information in this section

Basics of Platforms and Simulation Applications	20
General information on platforms and simulation applications.	
Registering and Managing dSPACE Platforms	26
Registering a platform makes it accessible.	
Updating and Repairing the Firmware of dSPACE Real-Time Hardware	35
Managing Simulation Applications Manually	41
The Platform Manager allows you to manage simulation applications manually.	
Automating Simulation Application Management	48
The Platform Management library allows you to automate some general features of the platform management.	

Basics of Platforms and Simulation Applications

Introduction

General information on platforms and simulation applications.

Where to go from here

Information in this section

Basics on Platforms	21
General information on platforms.	
Basics on Simulation Applications	22
General information on simulation applications.	

Basics on Platforms

Introduction

Platforms are used for carrying out calibration, measurement, or diagnostics tasks. They provide access to other entities, such as dSPACE real-time hardware or VEOS.

Supported dSPACE platforms

Platforms for accessing dSPACE real-time hardware dSPACE real-time hardware is used for function prototyping (for example, with MicroAutoBox) or HIL simulation (for example, with a modular system based on the DS1006 Processor Board).

AutomationDesk provides platforms that support dSPACE real-time hardware:

dSPACE Real-Time Hardware	Platform Used in AutomationDesk
Modular system based on a single single-core DS1006 Processor Board	DS1006 Processor Board
Modular system based on a single multicore DS1006 Processor Board	Multiprocessor System
Modular system based on multiple DS1006 Processor Boards	Multiprocessor System
Modular system based on a single multicore or multiple DS1007 PPC Processor Boards	DS1007 PPC Processor Board
DS1104 R&D Controller Board	DS1104 R&D Controller Board
MicroAutoBox II (all variants)	MicroAutoBox
MicroAutoBox III	MicroAutoBox III
MicroLabBox	DS1202 MicroLabBox
SCALEXIO system (with one or multiple processing units or DS6001 Processor Boards)	SCALEXIO

Platform for accessing environment VPUs Environment VPUs are used for virtual ECU testing purposes. They let you simulate applications offline. For offline simulation, AutomationDesk supports the VEOS platform.

Supported manual platform management

For manual platform management, AutomationDesk provides functions that allow you to:

- Register the platforms. For further information, refer to [How to Register a dSPACE Platform](#) on page 27.
- Load, start and stop applications. For further information, refer to [Basics on the Platform Manager](#) on page 41.

Supported automated platform management

For automating platform management, AutomationDesk provides two methods:

- For a convenient way to load, start and stop simulation applications, you can use the blocks of the Platform Management library in your sequences. For further information, refer to [Platform Management](#) on page 142.

Note

The Platform Management library is based on the dSPACE PlatformManagement API. All enhancements of the API are automatically accessible by the Platform Management library.

- For full platform management functionality, you can use the dSPACE PlatformManagement API in your Python code. For further information, refer to [dSPACE Platform Management API Reference](#).

Note

If the Platform Management API is not installed automatically with the products you purchased, you have to explicitly execute the dSPACE Python Extensions setup. For further information, refer to [How to Install dSPACE Software \(Installing dSPACE Software\)](#).

Features of platforms for accessing dSPACE real-time hardware

For the features of the platforms supported by AutomationDesk, refer to:

- [DS1006 Features](#)
- [DS1007 Features](#)
- [DS1104 Features](#)
- [MicroAutoBox II Features](#)
- [MicroAutoBox III Hardware Installation and Configuration](#)
- [MicroLabBox Features](#)
- [SCALEXIO – Hardware and Software Overview](#)

Related topics

Basics

Basics on Simulation Applications.....	22
Registering and Managing dSPACE Platforms.....	26

References

Platform Management.....	142
Platform Manager.....	302

Basics on Simulation Applications

Introduction

Simulation applications are the applications to be executed on a platform.

Terminology

Simulation application is the generic term for:

- *Real-time application*
- *Offline simulation application*

Real-time applications are applications to be executed on a dSPACE platform, and offline simulation applications are applications to be executed on VEOS.

Simulation application is a synonym of *executable application*, which is a term used in ConfigurationDesk.

Platform-independent management functions

The following functionality to manage simulation applications is available on any platform type:

- Loading the simulation application to the platform. If available, you can choose between the platform's RAM and flash memory. If there is a simulation application running, it is stopped and overwritten. Whether the simulation application starts automatically depends on its start settings.
- Terminating the execution of the running simulation application and resetting the platform.

Platform-dependent management functions

The following functionality to control the execution of simulation applications is supported depending on the platform type:

- Stopping and restarting the execution of a simulation application.
- Unloading the simulation application from the platform it has been loaded to. This puts the platform in a safe state and makes it available for new tasks.

Tip

Whether your platform supports execution control of simulation applications depends on the `RealTimeApplication` property of the related platform type object.

In the Platform Management API Reference, look up the interface description of your platform named `<platform type>Platform`. The Properties table contains a description and the data type of `RealTimeApplication`. If the type is `ControllableRealTimeApplication`, execution control of simulation applications is supported.

Currently, SCALEXIO platforms and VEOS support this feature.

Specifying the simulation application to be executed

In AutomationDesk's platform management, the path and name of the variable description file (SDF) is used to specify a simulation application.

The SDF is generated when a simulation application is built. This file associates the executable code with information used to access variables of a simulation application.

Execute platform management functions	AutomationDesk offers two ways to manage simulation applications: <ul style="list-style-type: none">▪ <i>Manually</i> using the Platform Manager in AutomationDesk. For more information, refer to Managing Simulation Applications Manually on page 41.▪ <i>Automated</i> using the blocks of the Platform Management library in an AutomationDesk sequence. For more information, refer to Automating Simulation Application Management on page 48.
--	--

Firmware compatibility guidelines	<p>Firmware and real-time application Firmware is downward-compatible.</p> <p>This means:</p> <ul style="list-style-type: none">▪ You can use the firmware from a dSPACE Release in the following cases:<ul style="list-style-type: none">▪ The <i>firmware is of the same dSPACE Release</i> with which the real-time application was built.▪ The <i>firmware is of a newer dSPACE Release</i> than the dSPACE Release with which the real-time application was built.▪ You cannot use the firmware from a dSPACE Release if the <i>firmware is of an older dSPACE Release</i> than the dSPACE Release with which the real-time application was built. <p>Hardware dependency of the required firmware version</p> <ul style="list-style-type: none">▪ 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>.

Related topics

Basics

Automating Simulation Application Management.....	48
Basics on Platforms.....	21
Managing Simulation Applications Manually.....	41

Registering and Managing dSPACE Platforms

Introduction

For platform access, it is required to register the platform before you can use it. AutomationDesk provides the Platform Manager for this.

Note

Each time AutomationDesk starts, it scans the current hardware configuration and searches for registered platforms connected via bus interface. It also searches for connected platforms that do not need to be registered (MicroAutoBox connected via bus, and DS1104). The platforms that are found are displayed in the Platform Manager.

AutomationDesk does not scan for registered platforms connected via network interface.

Do this manually after you have started AutomationDesk, by clicking Platform Management - Refresh Platform Configuration on the Platforms ribbon.

Where to go from here

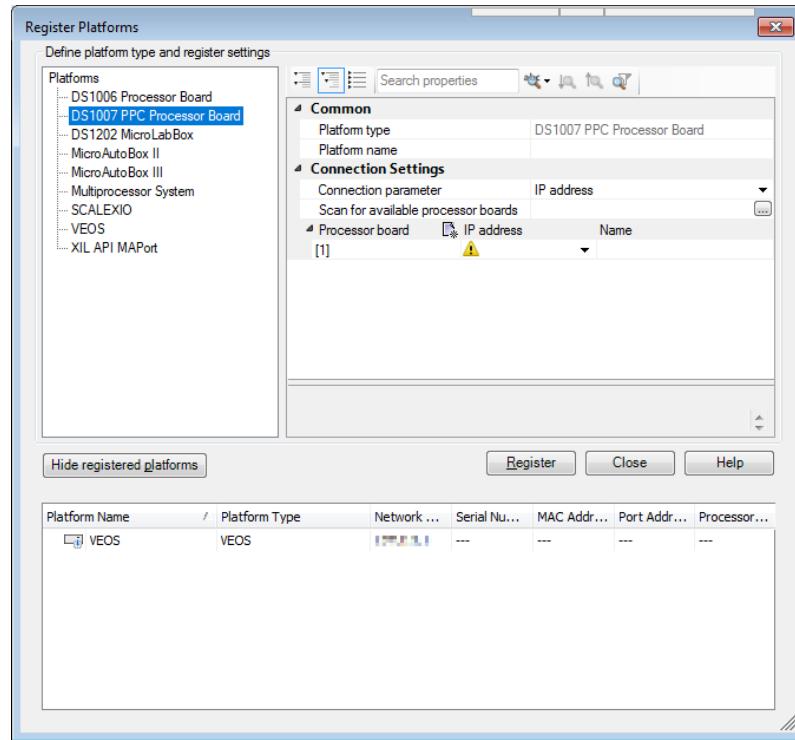
Information in this section

How to Register a dSPACE Platform.....	27
After installing dSPACE real-time hardware or VEOS, you have to make it known to AutomationDesk.	
Synchronized Platform Management with Several dSPACE Products.....	29
Several dSPACE products support synchronized platform management. The platform management instances in these products are synchronized.	
Accessing the SCALEXIO or MicroAutoBox III System with Multiple dSPACE Products Simultaneously.....	31
You can access a SCALEXIO or MicroAutoBox III system with multiple dSPACE products at the same time, if they run on the same PC, and you can access a SCALEXIO or MicroAutoBox III system from a second PC by using ControlDesk.	
Specifics on Working with Multiprocessor Systems with Optional Processors.....	34
You can configure the use of CPUs in multiprocessor systems.	

How to Register a dSPACE Platform

Objective	After installing a single dSPACE processor or controller board, a multiprocessor system, MicroAutoBox II/III, SCALEXIO system or VEOS, you have to register it to make it known to AutomationDesk.
Preconditions	<ul style="list-style-type: none">▪ To register a platform to access dSPACE real-time hardware, the hardware must be connected to the host PC.▪ MicroAutoBox II: Before you register the MicroAutoBox II, you should configure it using the DS1401ConfigGUI.exe utility, for example, to change the default IP address. The utility is located in <code><RCP_HIL_InstallationPath>\Exe</code>. For configuration details, refer to Connecting the MicroAutoBox II to the Host PC via Ethernet (MicroAutoBox II Hardware Installation and Configuration Guide).▪ To register a platform to access VEOS, VEOS must be installed on the host PC.
Exception	You do not need to register DS1104 boards. They support the plug & play feature and the Platform Manager registers them automatically.
Automated platform access	Platforms need to be registered manually before you can automate access to them.
Method	<p>To register a dSPACE platform</p> <p>1 On the Platforms ribbon, click Platform Management - Register Platforms.</p>

The Register Platforms dialog opens.



- 2 From the Platforms list, select the type of the platform you want to register.
- 3 Specify the connection settings for the dSPACE hardware or VEOS you want to register, for example, the platform type. If you work with a multiprocessor system, note the Multiprocessor Configuration Properties.

Tip

AutomationDesk helps you to avoid erroneous entries. Affected registration property settings are marked with the symbol. Move the mouse pointer over the symbol to open a tooltip with information on the reason for the error.

- 4 Click Register to complete the registration.
- The registered platform is displayed with its registration settings in the Registered platforms list.
- 5 Repeat steps 2 ... 4 for all the dSPACE platforms you want to register.
- 6 Click Close to close the Register Platforms dialog.

Result

You have registered dSPACE platforms independently of AutomationDesk projects. The platforms are displayed in the Platform Manager and can be added to AutomationDesk projects later on. The registration data is stored in the recent platform configuration.

Next step You can now assign the dSPACE real-time hardware or VEOS to the platform used in an AutomationDesk library that accesses platforms, for example, the XIL API library.

Related topics **References**

Manage Platforms.....	314
Refresh Interface Connections.....	325
Refresh Platform Configuration.....	326
Register Platforms.....	326

Synchronized Platform Management with Several dSPACE Products

Introduction Several dSPACE products support synchronized platform management. The platform management instances in these products are synchronized.

Synchronization of platform management instances Several dSPACE products have a **Platform Manager** that displays all the registered platforms with their components and running applications that can be accessed via the products. There are functions to register platforms, to manage the platform configuration, and to handle the real-time applications loaded to the platforms.

If you work simultaneously with several of these dSPACE products, each of them has its own platform management instance running. The instances contain consistent information about the connected platforms. This means that when you perform a platform management activity in one instance, the contents of all the other currently running platform management instances are synchronized accordingly.

Tip

A platform management instance provides information on platforms and the applications loaded to them only for those platforms that are supported by the respective dSPACE product.

Performing platform management activities The following table shows the platform management activities that are synchronized between the platform management instances. You can see which platform management activities are possible even if another platform management activity in another dSPACE product is currently running. Depending on the activity you perform, simultaneous access to the hardware and real-time

applications from several dSPACE products can be restricted, because exclusive access to a single platform or to all registered platforms might be necessary.

Note

Not every platform management activity is available in every dSPACE product.

Activity You Want to Perform in Platform Management Instance A	Activity That is Currently Running in Platform Management Instance B											
	Register Platforms	Refresh Interface Connections	Refresh Platform Configuration	Clear System	Manage Recent Platform Configuration	Load/Reload Real-Time Application	Stop RTP(s)	Unload Real-Time Application	Update Firmware	Clear Flash	Explore Logged Data	Online Calibration is Started
Register Platforms	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓
Refresh Interface Connections	-	-	-	-	-	-	-	-	-	-	-	
Refresh Platform Configuration	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	
Clear System	-	-	-	-	-	-	-	-	-	-	-	
Manage Recent Platform Configuration	-	-	-	-	-	-	-	-	-	-	-	
Load Real-Time Application	✓	-	✓	-	-	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	
Stop RTP(s)	✓	-	✓	-	-	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	
Unload Real-Time Application	✓	-	✓	-	-	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	
Update Firmware	✓	-	✓	-	-	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	
Clear Flash	✓	-	✓	-	-	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	
Explore Logged Data	✓	-	✓	-	-	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	
Go Online / Start Online Calibration	✓	-	✓	-	-	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	✓ ¹⁾	

¹⁾ Only possible for different platforms since the activity you want to perform requires exclusive platform access.

dSPACE products supporting platform management synchronization

Platform management synchronization is performed when you access dSPACE platforms *simultaneously* using any combination of the following software products:

- AutomationDesk as of Version 3.6p2
Note that automated access via Platform Management library and XIL API library might also require exclusive access to a platform.
- ConfigurationDesk as of Version 4.4
- ControlDesk as of Version 5.0

- Firmware Manager as of Version 1.0
- ModelDesk as of Version 3.1
- RTT Manager as of Version 2.1

Firewall settings

To enable platform management synchronization, the firewalls of the PCs must be configured to allow communication for simultaneous platform access.

Windows firewalls During the installation of dSPACE software, Windows firewalls are automatically configured to allow communication between the platform management instances for synchronization. You do not have to configure Windows firewalls manually. The first time you start a product involved in platform management synchronization, the firewall asks you to allow access. Confirm the product as trusted software.

Other firewalls If the host PC has a firewall different from the Windows firewall, configure that firewall manually to allow communication between the platform management instances of the products.

Related topics**Basics**

Accessing the SCALEXIO or MicroAutoBox III System with Multiple dSPACE Products Simultaneously.....	31
---	----

Accessing the SCALEXIO or MicroAutoBox III System with Multiple dSPACE Products Simultaneously

Accessing the SCALEXIO or MicroAutoBox III system from one PC

You can access a SCALEXIO or MicroAutoBox III system with the following dSPACE products at the same time if they run on the same PC:

- ConfigurationDesk
- ControlDesk
- AutomationDesk
- ModelDesk
- dSPACE XIL API MAPort implementation

Note

The products to access the SCALEXIO or MicroAutoBox III system must be installed from the same dSPACE Release.

Accessing the SCALEXIO or MicroAutoBox III system from a second PC

Another person can also access the SCALEXIO or MicroAutoBox III system by using ControlDesk running on a second PC.

Note

Consider the following restrictions when accessing the system from a second PC:

- Do not execute platform management functions from a second PC.
These are functions such as:
 - Downloading a real-time application to the SCALEXIO or MicroAutoBox III system
 - Starting/stopping the real-time application on the SCALEXIO or MicroAutoBox III system
- Access to the SCALEXIO or MicroAutoBox III system from a second PC is restricted to measurement and recording tasks:
 - Do not use ControlDesk's Signal Editor from a second PC to stimulate variables of the real-time application running on the SCALEXIO or MicroAutoBox III system
 - Do not use Real-Time Testing from a second PC to perform tests synchronously to the real-time application running on the SCALEXIO or MicroAutoBox III system
 - Do not use ControlDesk's Failure Simulation Module from a second PC to control the failure simulation hardware of the SCALEXIO system
- The products to access the SCALEXIO or MicroAutoBox III system (incl. ControlDesk running on a second PC) must be installed from the same dSPACE Release.

⚠ CAUTION

When you use AutomationDesk to carry out automated tests on a SCALEXIO or MicroAutoBox III system, do not use ControlDesk to change the values of model parameters that influence the currently running test, because this will falsify the results.

The person carrying out automated tests will not even be able to detect whether the SCALEXIO or MicroAutoBox III system is currently being accessed by ControlDesk.

Access levels

Access levels are provided by the system for accessing the:

- SCALEXIO or MicroAutoBox III system
- Real-time application (executable on the SCALEXIO or MicroAutoBox III system)

Depending on the action you perform, other users' access to the hardware and real-time application can be restricted. There are two access levels:

- Exclusive access
Other users have no access.
- Shared access
Other users have shared but not an exclusive access. Actions which need shared access can be carried out by several users at the same time.

Access required for specific actions

The following table shows actions in ConfigurationDesk, ControlDesk, AutomationDesk and ModelDesk for which each user needs exclusive or shared access to the hardware and/or to the real-time application.

Action	Required Access Level	
	To SCALEXIO or MicroAutoBox III System	To Real-Time Application
Registering hardware	Exclusive	None
Changing properties of the hardware ¹⁾	Change the system name or names of hardware components (for example, real-time PC or I/O unit)	Exclusive
	Change internal load description of a channel ^{[3), 4)}	Exclusive
	Change load rejection settings of a channel ^{[3), 4)}	Exclusive
Handling real-time applications	Load application	Shared
	Unload application	Shared
	Start application	Shared
	Stop application	Shared
ControlDesk's device state changes ⁵⁾	Start online calibration	Shared
	Start measuring/recording	Shared
Updating the firmware	Exclusive	None ²⁾

¹⁾ If you change a property, the hardware is accessed only when the new value is applied by ConfigurationDesk. The more properties are changed, the longer the update (and therefore the access) takes.

²⁾ Not possible when a real-time application is loaded

³⁾ Only possible in ConfigurationDesk

⁴⁾ Not applicable for MicroAutoBox III

⁵⁾ Only possible in ControlDesk. Each access level applies for the entire duration of the corresponding device state.

Tip

The required access levels are independent of the status of the ConfigurationDesk application (Connected to hardware or Not connected to hardware).

Examples of handling real-time applications

- While you *load* a real-time application, other users cannot unload it. However, they can start and stop it.
- While you *start* a real-time application, other users can access it, for example, to stop it.
- While one user starts ControlDesk's online calibration, other users cannot load or unload the real-time application. However, they can start and stop it.

Error messages

Whenever access is not possible, ConfigurationDesk, ControlDesk, AutomationDesk or ModelDesk displays an error message.

Specifics on Working with Multiprocessor Systems with Optional Processors

Introduction

Typically, all the processors in a multiprocessor system application are used in a real-time simulation. However, RTI-MP lets you generate an SDF file in which one or more of the processors are *optional* (refer to [How to Specify Optional CPUs in RTI-MP \(DS1006 Features\)](#)). This lets you disable the real-time applications of specific multiprocessor system members without having to rebuild the real-time application.

Disabling/enabling optional processors in AutomationDesk

To disable or enable the real-time application of specific multiprocessor system members, perform the following steps:

1. Open the SDF file of the multiprocessor system application in a text editor.
2. In the file, specify **disabled** or **enabled** as the state of the optional processor(s).
3. Save the SDF file under a new name.

Note

If you already added a variable description to the platform, it is not sufficient to save the changed SDF file under its original name and reload the variable description. Instead, you must specify a new file name and add the file as a new variable description.

4. Add the changed SDF file to the Multiprocessor System platform.

Tip

It is recommended to add the variable description with all the processor states set to **enabled** in the first step. This lets you visualize variables from all processors in instruments. When you disable processors later on, the variables of the disabled processors are displayed in the no-value view in instruments, but their variable connections remain valid.

Example where one optional processor is disabled

The following example applies to a multiprocessor system with three processors where the second processor is disabled.

In the SDF file, the state of the second processor is **disabled**. The state of the other processors is **enabled**.

```
[SLAVE]
Type=DS1006
ServiceId=2
BoardName=unknown
File=slave.ppc
state=disabled

[MASTER]
Type=DS1006
ServiceId=1
BoardName=unknown
File=master.ppc
state=enabled

[System]
Version=1.0
Status=Start
SystemType=MultiProcessorSystem
RTP=MASTER
RTB=SLAVE
RTB=SLAVE_B

[SLAVE_B]
Type=DS1006
ServiceId=3
BoardName=unknown
File=slave_b.ppc
state=enabled
```

Related topics**Basics**

[Basics on Multiprocessor System Platforms \(ControlDesk Platform Management\)](#)

Updating and Repairing the Firmware of dSPACE Real-Time Hardware

Introduction

Before you start a firmware update, you have to decide whether to update the firmware to a later version or to repair the currently installed firmware version.

Where to go from here

Information in this section

[Basics on Firmware](#)..... 36

Gives you information on the different kinds of firmware.

[How to Prepare the Firmware Update](#)..... 36

Before you start an update or repair process, some preparations have to be made.

[How to Update Firmware](#)..... 38

Gives you the instructions for the firmware update mode.

[How to Repair Firmware](#)..... 39

Gives you the instructions for the firmware repair mode.

Basics on Firmware

Introduction

You can execute a real-time application on dSPACE real-time hardware only if the different kinds of firmware are available. The loaded firmware version has to provide the functionality implemented in the real-time application.

Firmware features

The firmware for a hardware component provides basic functionality that is stored in a nonvolatile memory. For example, it includes functions for the communication between the host PC and the hardware, and can also provide I/O functions such as CAN or LIN protocol support, or complex I/O functions for an FPGA component.

The firmware archives provides all the relevant firmware components that are required for your hardware.

For further information, refer to [Firmware Manager Manual](#).

How to Prepare the Firmware Update

Objective

The preparation of a firmware update consists of specifying some general firmware settings.

Preconditions

The following preconditions must be fulfilled for configuring the general firmware settings:

- The real-time hardware must be connected to the host PC.
- The real-time hardware must be switched on.
- The required firmware archive must be available.

You can find the latest firmware archives on the dSPACE website at
<http://www.dspace.com/go/firmware>.

- If a real-time application is loaded to the board's flash memory, it is recommended to clear the flash before starting the update process to avoid unpredictable output signals.
- If a real-time application is running, it is stopped by the firmware management.
- If you have registered a multiprocessor system, you can update only one processor at a time.
- If you have registered a multicore system with additional I/O boards, you have to select the core to which the I/O boards are connected for the update of the entire system. The other cores will be updated, too.

Note

- The archive format for DS1007 and MicroLabBox changed with Firmware Archives 2.0 contained in dSPACE Release 2015-B. To open an archive in the new format, you must use Firmware Manager 2.0 or later.
- The archive format for SCALEXIO changed with Firmware Archives 2.1 contained in dSPACE Release 2016-A. To open an archive in the new format, you must use Firmware Manager 2.1 or later.

Method**To prepare the firmware update**

- 1 Open the Platform Manager.
- 2 If no real-time hardware is displayed in the Platform Manager, register the real-time hardware that you want to update.
- 3 Choose Update Firmware in the platform's context menu to open the Update Firmware Wizard.
 The wizard starts with the Select Mode dialog.
- 4 Select the firmware update mode.
 By default, the Update mode is set to update all firmware components of your real-time hardware with later firmware. With the Repair mode enabled, you can select the firmware components to be repaired.
 To switch to the repair mode, select Firmware repair mode in the Select Mode dialog.
- 5 Click Next to continue with the Select Firmware Archive dialog.

The latest firmware archive for the selected platform is automatically set. Optionally, browse for another firmware archive. This might be useful if you want to update to a firmware version other than the latest or repair user firmware, for example.

- 6 Click Next to continue with the Select Firmware Components dialog.

Result	You have configured the settings which are required for a firmware update process in update or repair mode.
---------------	---

Related topics	HowTos
<div style="background-color: #f0f0f0; padding: 5px;">How to Repair Firmware.....39 How to Update Firmware.....38</div>	

How to Update Firmware

Objective	Gives you the instructions for the firmware <i>update</i> mode.
------------------	---

Preconditions	The firmware update process has to be prepared with the Update Mode specified as described in How to Prepare the Firmware Update on page 36.
----------------------	--

Safety precautions	⚠ WARNING Risk of injury and/or material damage Updating the firmware can cause uncontrolled movements of connected devices. <ul style="list-style-type: none">▪ Disconnect actuators and sensors from the associated real-time hardware before you start the update process.
---------------------------	---

NOTICE Interrupting the update process disables the hardware If the firmware update is interrupted, for example, by switching off the power, you have to restart the update process.
--

Note Follow the instructions of the firmware management tool to correctly finish the firmware update process. For example, in some cases the hardware has to be rebooted to complete the firmware update.

Method**To update firmware**

- 1 In the Select Firmware Components dialog, click Update to start the firmware update process.
- In the Update column, the firmware components to be updated are marked and red. The components are not marked for update if the version of the currently installed firmware is identical to or later than the firmware available in the specified firmware archive.
- If there are updatable firmware components, the update process starts. You can see the progress in the Status column. The initial '--' entry is replaced by a percentage. If the progress information cannot be detected continuously, only the states 50% and 100% are displayed. If the process successfully finished, an OK is shown, otherwise an error message is displayed.
- If the firmware update will require more than 40 minutes, an estimate of the time is displayed. Then you can decide whether to start the process.
- Interrupting a running firmware update process is not possible.

Note

You must not switch off the hardware during the firmware update process. This will cause a corrupted firmware.

Follow the given instructions to complete the firmware update. For example, some firmware components require a hardware restart.

Result

You have updated the firmware components of your hardware.

Related topics**HowTos**

How to Prepare the Firmware Update.....	36
How to Repair Firmware.....	39

How to Repair Firmware

Objective

Gives you the instructions for the firmware *repair* mode.

Preconditions

The firmware update process has to be prepared with the Repair Mode specified as described in [How to Prepare the Firmware Update](#) on page 36.

Safety precautions**⚠ WARNING****Risk of injury and/or material damage**

Updating the firmware can cause uncontrolled movements of connected devices.

- Disconnect actuators and sensors from the associated real-time hardware before you start the update process.

NOTICE**Interrupting the update process disables the hardware**

If the firmware update is interrupted, for example, by switching off the power, you have to restart the update process.

Note

Follow the instructions of the firmware management tool to correctly finish the firmware update process. For example, in some cases the hardware has to be rebooted to complete the firmware update.

Method**To repair firmware**

- 1 In the Select Firmware Components dialog, select the firmware components to be repaired in the Update column.
You can select only firmware components, whose current and available versions are identical. If the versions differ, the components are not displayed at all.
- 2 In the Select Firmware Components dialog, click Repair to start the firmware repair process.
This command is enabled only if at least one firmware component is selected for repairing.
If there are updatable firmware components, the repair process starts. You can see the progress in the Status column. The initial '--' entry is replaced by a percentage. If the progress information cannot be detected continuously, only the states 50% and 100% are displayed. If the process successfully finished, an OK is shown, otherwise an error message is displayed.
If the firmware repair process will require more than 40 minutes, an estimate of the time is displayed. Then you can decide whether to start the process.
Interrupting a running firmware repair process is not possible.

Note

You must not switch off the hardware during the firmware repair process. This will cause a corrupted firmware.

Follow the given instructions to complete the firmware update. For example, some firmware components require a hardware restart.

Result You have repaired the firmware components of your hardware.

Related topics HowTos

How to Prepare the Firmware Update.....	36
How to Update Firmware.....	38

Managing Simulation Applications Manually

Introduction Using AutomationDesk's Platform Manager, you can manage simulation applications manually.

Where to go from here Information in this section

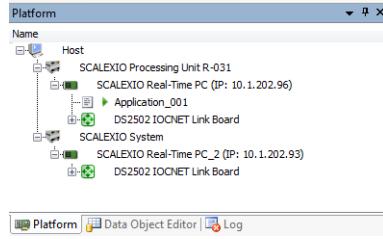
Basics on the Platform Manager.....	41
General information on the Platform Manager.	
How to Load and Start a Simulation Application Manually.....	43
Instructions on using context menus in the Platform Manager to load and start simulation applications.	
How to Stop and Restart a Simulation Application Manually.....	45
Instructions on using context menus in the Platform Manager to stop and restart simulation applications.	
How to Terminate a Simulation Application Manually.....	46
Instructions on using context menus in the Platform Manager to terminate simulation applications.	

Basics on the Platform Manager

Introduction To manage [platforms](#) and [simulation applications](#) manually.

Accessing the Platform Manager If you use the standard screen arrangement, the Platform Manager is displayed in a pane group together with the Data Object Editor and the Message Viewer. You can click its tab to display it. If the Platform Manager is not part

of the current screen arrangement, you must select Platform on the View ribbon in the submenu of Controlbars - Switch Controlbars.

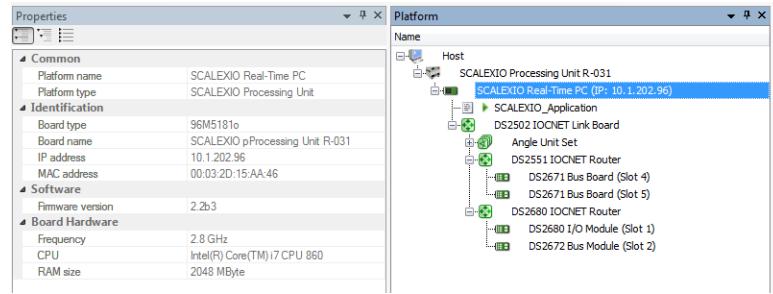


Platform Manager features

The Platform Manager provides the following major capabilities:

- Display platform configuration

The Platform Manager displays the registered platforms in a tree structure. Each hardware component or loaded simulation application is displayed as a node of the related platform entry. More detailed properties of a node can be displayed by using the **Properties** command in its context menu.



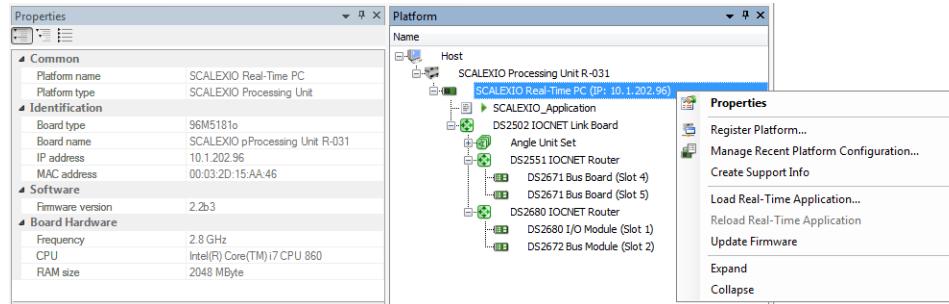
The different types of entries in the Platform Manager can be easily identified by their symbols. Symbols also indicate the current state. A running simulation application is shown by ▶ before the platform or component name, a stopped simulation application by ■. For an overview of the symbols used, refer to [Platform Manager](#) on page 317.

Tip

The property grid of a platform shows the Platform name attribute. You can use this unique name to specify the platform if you want to automate platform management.

- Manage platform configuration

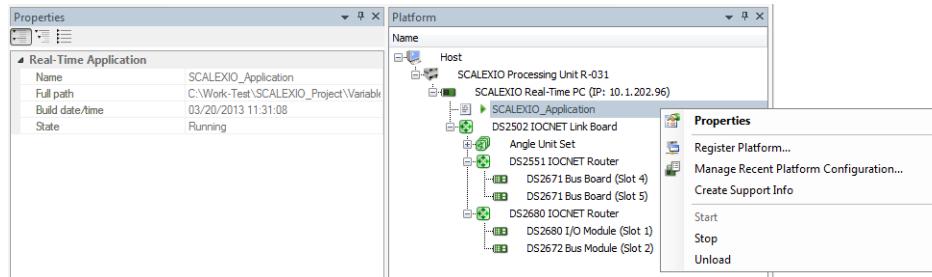
The Platform Manager provides functions to manage platform configuration in the context menus of the displayed tree nodes.



Platform management functions which do apply to all existing nodes are also available on the Platform ribbon. For detailed Information, refer to [Platform Manager](#) on page 302

- Manage simulation applications on platforms

The Platform Manager provides functions to manage simulation applications.



Related topics

Basics

Basics on Platforms.....	21
Basics on Simulation Applications.....	22

References

Platform Management.....	142
Platform Manager.....	302

How to Load and Start a Simulation Application Manually

Objective

To load and start a [simulation application](#) using AutomationDesk's [Project Manager](#).

Preconditions	<ul style="list-style-type: none">▪ The platform to execute the simulation application is connected to the PC.▪ The platform to execute the simulation application is registered. For more information, refer to How to Register a dSPACE Platform on page 27.▪ The Platform Manager is displayed.
----------------------	--

Method	<p>To load and start a simulation application manually</p> <ol style="list-style-type: none">1 In the Platform Manager, open the context menu of the platform you want to load the simulation application to.2 Select one of the following commands:<ul style="list-style-type: none">▪ Real-Time Application - Load to load the simulation application to the RAM of the platform. In the Select Real-Time Application dialog, specify the system description file of the simulation application to be loaded.▪ Real-Time Application - Load to Flash to load the simulation application to the flash memory of the platform. In the Select Real-Time Application dialog, specify the system description file of the simulation application to be loaded.▪ Real-Time Application - Reload to reload the simulation application that was previously loaded to the platform.The specified simulation application is loaded to the RAM or flash memory of the platform.▪ Using DS1006, DS1104, MicroAutoBox II, or Multiprocessor System based on DS1006 The simulation application is started automatically, if you specified this behavior when building the simulation application.▪ Using DS1007, MicroLabBox III, SCALEXIO, or VEOS The simulation application is not started automatically. For these platforms, you can use the Load and Start commands to start the application after the download.
---------------	---

Result	AutomationDesk loads the specified simulation application to the platform. If there is a previously loaded simulation application, it is overwritten. The simulation application and its execution state are displayed in the Platform Manager.
---------------	---

Related topics	Basics
<div style="background-color: #e0e0e0; padding: 5px;">Basics on Platforms.....21 Basics on Simulation Applications.....22</div>	

Basics on the Platform Manager.....	41
HowTos	
How to Stop and Restart a Simulation Application Manually.....	45
How to Terminate a Simulation Application Manually.....	46
References	
Real-Time Application - Load.....	319
Real-Time Application - Load to Flash (DS1006/DS1104/MicroAutoBox II).....	322
Real-Time Application - Load to Flash (DS1007/MicroLabBox/MicroAutoBox III/SCALEXIO).....	323
Real-Time Application - Load to Flash and Start (DS1007/MicroLabBox/MicroAutoBox III/SCALEXIO).....	324
Real-Time Application - Reload.....	325
Real-Time Application / Offline Simulation Application - Load.....	320
Real-Time Application / Offline Simulation Application - Load and Start.....	321
Reload.....	331
Reload and Start.....	332

How to Stop and Restart a Simulation Application Manually

Objective	To stop and restart a simulation application  using AutomationDesk's Platform Manager  .
Preconditions	<ul style="list-style-type: none"> ▪ The platform  executing the simulation application is connected to the PC. ▪ The platform executing the simulation application is registered. For more information, refer to How to Register a dSPACE Platform on page 27. ▪ There is a simulation application running on the platform. For more information, refer to How to Load and Start a Simulation Application Manually on page 43.
Restrictions	<p>The functionality to stop and restart a simulation application without reloading is only available on platforms that support execution control of simulation applications. For more information, refer to Basics on Simulation Applications on page 22.</p> <p>Currently, SCALEXIO platforms and VEOS support this feature.</p>
Method	<p>To stop and restart a simulation application manually</p> <ol style="list-style-type: none"> 1 In the Platform Manager, open the context menu of the simulation application entry you want to stop.

- 2 Choose Stop to stop the execution of the simulation application.
A stopped simulation application is shown by ■ before the simulation application entry in the Platform Manager.
- 3 In the Platform Manager, open the context menu of the simulation application entry you want to restart.
- 4 Choose Start to restart the execution of the simulation application.
A running simulation application is shown by ▶ before the simulation application entry in the Platform Manager.

Result	The simulation application has been stopped and restarted on the platform. The change of the execution state is displayed in the Platform Manager.
---------------	--

Related topics	Basics <table><tr><td>Basics on Platforms.....</td><td>21</td></tr><tr><td>Basics on Simulation Applications.....</td><td>22</td></tr><tr><td>Basics on the Platform Manager.....</td><td>41</td></tr></table> HowTos <table><tr><td>How to Automate the Loading and Starting of Simulation Applications.....</td><td>51</td></tr><tr><td>How to Load and Start a Simulation Application Manually.....</td><td>43</td></tr><tr><td>How to Terminate a Simulation Application Manually.....</td><td>46</td></tr></table> References <table><tr><td>Start.....</td><td>336</td></tr><tr><td>Stop.....</td><td>337</td></tr></table>	Basics on Platforms.....	21	Basics on Simulation Applications.....	22	Basics on the Platform Manager.....	41	How to Automate the Loading and Starting of Simulation Applications.....	51	How to Load and Start a Simulation Application Manually.....	43	How to Terminate a Simulation Application Manually.....	46	Start.....	336	Stop.....	337
Basics on Platforms.....	21																
Basics on Simulation Applications.....	22																
Basics on the Platform Manager.....	41																
How to Automate the Loading and Starting of Simulation Applications.....	51																
How to Load and Start a Simulation Application Manually.....	43																
How to Terminate a Simulation Application Manually.....	46																
Start.....	336																
Stop.....	337																

How to Terminate a Simulation Application Manually

Objective	To terminate a simulation application using AutomationDesk's Platform Manager .
------------------	---

Basics	How simulation applications are terminated depends on the platform type. For more information, refer to Basics on Simulation Applications on page 22. <ul style="list-style-type: none">▪ If your platform type supports execution control of simulation applications, as do SCALEXIO platforms and VEOS, you can unload the simulation application and put the platform in a safe state.▪ On all other platforms, the execution of the simulation application is terminated by stopping the real-time processor (RTP).
---------------	--

After a simulation application is terminated, it has to be reloaded to execute it again.

Preconditions	<ul style="list-style-type: none">▪ The platform executing the simulation application is connected to the PC.▪ The platform executing the simulation application is registered. For more information, refer to How to Register a dSPACE Platform on page 27.▪ There is a simulation application running on the platform. For more information, refer to How to Load and Start a Simulation Application Manually on page 43.
Possible methods	<p>There are two ways of terminating a simulation application:</p> <ul style="list-style-type: none">▪ If your platform type supports execution control of simulation applications, as do SCALEXIO platforms and VEOS, refer to Method 1.▪ On all other platforms, refer to Method 2.
Method 1	<p>To terminate a simulation application on a platform supporting execution control</p> <ol style="list-style-type: none">1 In the Platform Manager, open the context menu of the application entry you want to terminate.2 Select Unload to terminate the simulation application on the platform.
Method 2	<p>To terminate a simulation application on a platform without execution control support</p> <ol style="list-style-type: none">1 In the Platform Manager, open the context menu of the platform you want to terminate the simulation application on.2 Select Stop RTP for single processor platforms or Stop RTPs for multiprocessor platforms to terminate the execution of the running simulation application.
Result	<p>The simulation application is terminated and the platform can be used for a new task.</p> <p>On SCALEXIO and VEOS platforms, the entry of an unloaded simulation application disappears from the Platform Manager.</p> <p>A stopped RTP is still displayed in the Platform Manager, but its state icon is set to ■.</p> <p>If you loaded the simulation application to the platform's flash memory, you can remove it using the Clear Flash command. For details, refer to Clear Flash on page 306.</p>

Related topics	Basics						
	<table><tr><td>Basics on Platforms.....</td><td>21</td></tr><tr><td>Basics on Simulation Applications.....</td><td>22</td></tr><tr><td>Basics on the Platform Manager.....</td><td>41</td></tr></table>	Basics on Platforms.....	21	Basics on Simulation Applications.....	22	Basics on the Platform Manager.....	41
Basics on Platforms.....	21						
Basics on Simulation Applications.....	22						
Basics on the Platform Manager.....	41						
	HowTos						
	<table><tr><td>How to Load and Start a Simulation Application Manually.....</td><td>43</td></tr><tr><td>How to Stop and Restart a Simulation Application Manually.....</td><td>45</td></tr></table>	How to Load and Start a Simulation Application Manually.....	43	How to Stop and Restart a Simulation Application Manually.....	45		
How to Load and Start a Simulation Application Manually.....	43						
How to Stop and Restart a Simulation Application Manually.....	45						
	References						
	<table><tr><td>Stop RTP.....</td><td>337</td></tr><tr><td>Stop RTPs.....</td><td>338</td></tr><tr><td>Unload.....</td><td>338</td></tr></table>	Stop RTP.....	337	Stop RTPs.....	338	Unload.....	338
Stop RTP.....	337						
Stop RTPs.....	338						
Unload.....	338						

Automating Simulation Application Management

Introduction

AutomationDesk's Platform Management library provides blocks to automate some general features of the platform management.

Tip

If you want to only download a simulation application to a platform and start or stop it, it is recommended to use automation blocks of the XIL API Convenience library. Refer to [How to Download and Start a Simulation Application](#) on page 81.

Note

The Platform Management library is based on the dSPACE Platform Management API. If this API is not installed automatically with the products you purchased, you have to install the dSPACE *Test Automation APIs* product set. For more information, refer to [How to Install dSPACE Software \(Installing dSPACE Software\)](#).

Where to go from here

Information in this section

Basics on the Platform Management Library.....	49
General information on the Platform Management custom library.	

How to Automate the Loading and Starting of Simulation Applications	51
The use of automation blocks to load and start a specified simulation application on a specified platform.	
How to Automate the Stopping and Restarting of Simulation Applications	53
The use of automation blocks to stop and restart a simulation application on a specified platform.	
How to Automate the Termination of Simulation Applications	55
The use of automation blocks to terminate a running simulation application on a specified platform.	

Basics on the Platform Management Library

Introduction

To automate platform management, AutomationDesk provides the Platform Management [library](#).

Library overview

The [automation blocks](#) are arranged in [folders](#) following the structure of the classes of the objects they apply to.

PlatformManagement An object of this class contains a collection of all currently registered platforms. Only one instance of this class is needed at the same time. All data and methods concerning platform management are derived from this object.

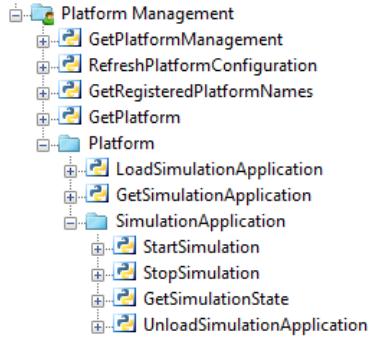
The automation blocks of this folder provide functions to create and refresh such an object. You can get a list of the names of all platforms and create a Platform object to access a platform with a specified name.

Platform An object of this class represents one specific [platform](#).

Using the automation blocks of this folder, you can load a specified simulation application to a platform and get a [SimulationApplication](#) object.

SimulationApplication An object of this class represents one [simulation application](#) that has been loaded to the platform.

Using the automation blocks of this folder, you can stop and start simulation applications, get the execution state they are in currently in and unload them from the platform.

**Access via Exec block**

The Python module **platformmanagementlib** provides methods with the same names as the corresponding automation blocks. Method arguments correspond in the same order to the block's input data objects. The output data object corresponds to the method's return value.

Example

```

import platformmanagementlib
MyPlatformManagement = \
    platformmanagementlib.GetPlatformManagement()
  
```

Enhancing library functionality

You can combine automation blocks of the Platform Management library with Exec blocks containing objects, methods and properties of the dSPACE Platform Management API.

There is a demo project in <DocumentsFolder>\Platform Management that shows how to create a platform management sequence that can be used with several platforms. This sequence uses blocks from the library and customized Exec blocks.

COM object handling

Automation blocks of the Platform Management library create VirtualCOM objects to communicate with the current platform management. You can assign these VirtualCOM objects to Variant data objects in your project. Using the "_AD_" alias, you can access them across your blocks, sequences and folders.

Once a VirtualCOM object is created, it can be used until AutomationDesk is closed, unless you explicitly do not release it.

You will find more detailed information about COM object handling and VirtualCOM object handling with some examples in the following application notes:

- COM object handling
[UsingCOMInAutomationDeskApplicationNote.pdf](#)
- VirtualCOM object handling
[UsingVirtualCOMInAutomationDeskApplicationNote.pdf](#)

Related topics**Basics**

Basics on Platforms.....	21
Basics on Simulation Applications.....	22
Basics on the Platform Manager.....	41

References

Platform Management.....	142
Platform Manager.....	302

How to Automate the Loading and Starting of Simulation Applications

Objective

To load and start a [simulation application](#) using the Platform Management library.

Preconditions

- The [platform](#) you want to access has to be registered manually. For instructions, refer to [How to Register a dSPACE Platform](#) on page 27.
- The name of the platform you want to access is required as input. You can get the name from the [Project Manager](#). For information, refer to [Basics on the Platform Manager](#) on page 41.
- The path and name of the [variable description file](#) (SDF) of the simulation application to execute are required as input.

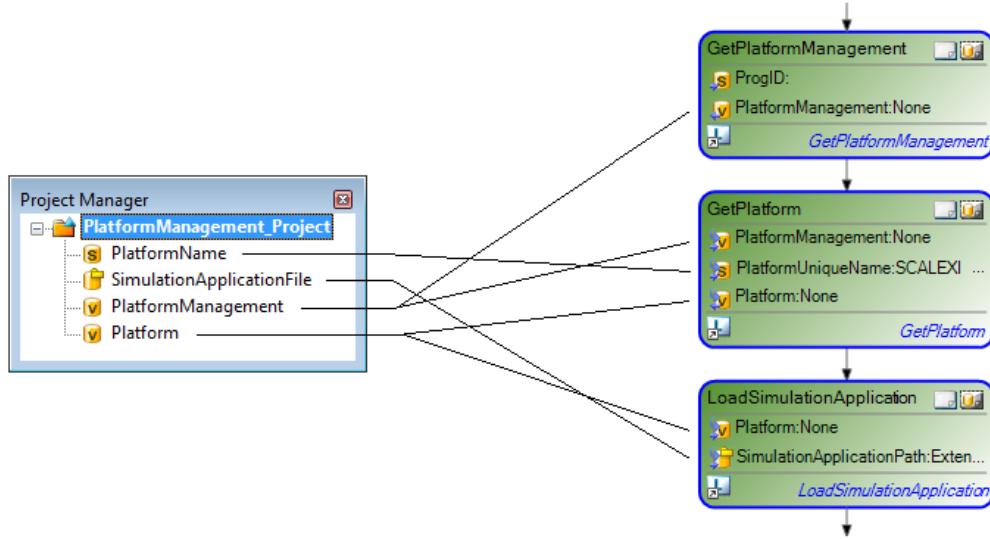
Method**To automate loading and starting of simulation applications**

- Add the following data objects to your project to parameterize the input data:
 - A String data object. In the Data Object Editor, parameterize it with the unique platform name of the platform you want to load the simulation application to.
 - A File data object. Parameterize it with the name and path of the system description file of the simulation application you want to load.
- Add the following data objects to your project to provide instantiated objects for referencing:
 - A Variant data object to store the instantiated PlatformManagement VirtualCOM object
 - A Variant data object to store the instantiated Platform VirtualCOM object
- From the Library Browser, drag a GetPlatformManagement block from the Platform Management library to the Sequence Builder. This instantiates a PlatformManagement object.

- 4 In the Data Object Editor, set the block's PlatformManagement data object as a reference to the project-specific Variant data object that stores the instantiated PlatformManagement object.
- 5 Drag a GetPlatform block from the Platform Management library to the Sequence Builder. This instantiates a Platform object of the platform with the specified unique name.
- 6 Set the block's PlatformManagement data object as a reference to the project-specific Variant data object that stores the instantiated PlatformManagement object.
- 7 Set the block's PlatformUniqueName data object as a reference to the project-specific String data object that stores the unique platform name.
- 8 Set the block's Platform data object as a reference to the project-specific Variant data object that stores the instantiated Platform object.
- 9 Drag a LoadSimulationApplication block from the Platform Management library to the Sequence Builder. This loads the specified simulation application to the platform.
- 10 Set the block's Platform data object as a reference to the project-specific Variant data object that stores the instantiated Platform object.
- 11 Set the block's SimulationApplicationPath data object as a reference to the project-specific File data object with the variable description file.

Result

You created a sequence that automatically loads a selected simulation application to a specified platform.



At execution time, a PlatformManagement object provides access to all currently registered platforms. It is used to get a Platform object to access a platform with a specified unique name. This platform object lets you to load the specified simulation application to the platform.

If there was a previously loaded simulation application, it is overwritten. You can define whether to start the simulation application automatically when building the simulation application.

Tip

Instead of reading the unique platform name from the Platform Manager, you can get a list of all recently registered platforms using the `GetRegisteredPlatformNames` block. For details, refer to [GetRegisteredPlatformNames](#) on page 146

Related topics**Basics**

Basics on Platforms	21
Basics on Simulation Applications	22
Basics on the Platform Management Library	49

HowTos

How to Automate the Termination of Simulation Applications	55
--	----

References

GetPlatform	144
GetPlatformManagement	145
LoadSimulationApplication	150

How to Automate the Stopping and Restarting of Simulation Applications

Objective

To stop and restart a simulation application  without reloading it.

Preconditions

You have loaded and started a simulation application to a [platform](#)  by using the Platform Management [library](#) How to Automate the Loading and Starting of Simulation Applications on page 51.

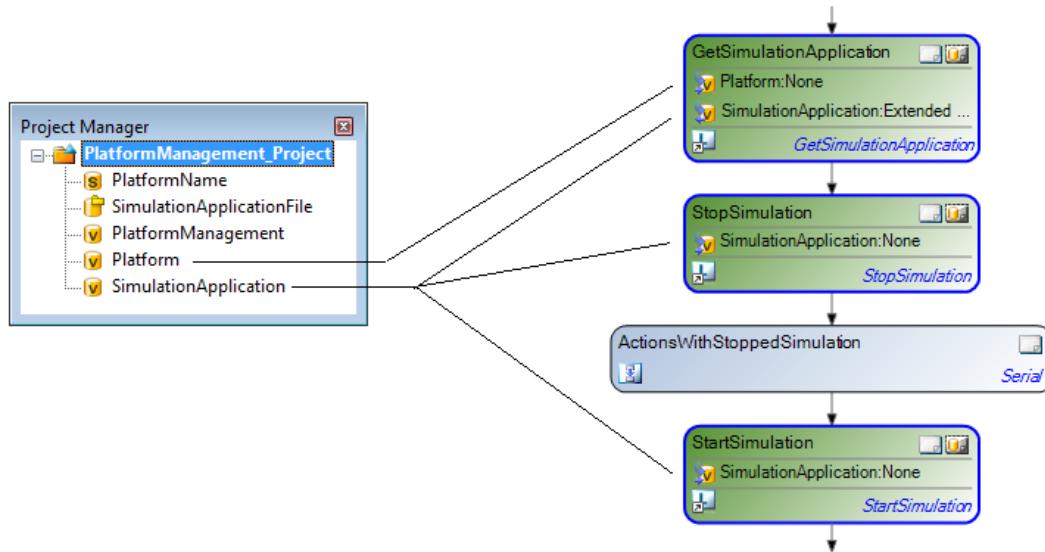
Restrictions

The functionality to stop and restart a simulation application without reloading it is available only on platforms that support execution control of simulation applications. For more information, refer to [Basics on Simulation Applications](#) on page 22.

Currently, SCALEXIO platforms and VEOS support this feature.

Method	To automate the stopping and restarting of simulation applications
	<p>1 Add the following data object to your project to provide instantiated objects for referencing:</p> <ul style="list-style-type: none"> ▪ A Variant data object to store the instantiated SimulationApplication object <p>2 Drag a GetSimulationApplication block from the Platform Management library to the Sequence Builder. This instantiates a SimulationApplication object of the simulation application that is currently running on the specified platform.</p> <p>3 Set the block's Platform data object as a reference to the project-specific Variant data object that stores the instantiated Platform object.</p> <p>4 Set the block's SimulationApplication data object as a reference to the project-specific Variant data object that stores the instantiated SimulationApplication object.</p> <p>5 Drag a StopSimulation block from the Platform Management library to the Sequence Builder. This stops the specified simulation application.</p> <p>6 Set the block's SimulationApplication data object as a reference to the project-specific Variant data object that stores the instantiated SimulationApplication object.</p> <p>7 Drag a StartSimulation block from the Platform Management library to the Sequence Builder. This starts the simulation application that has been stopped.</p> <p>8 Set the block's SimulationApplication data object as a reference to the project-specific Variant data object that stores the instantiated SimulationApplication object.</p>

Result	You created a sequence that automatically stops and restarts a currently running simulation application on a specified platform.
--------	--



At execution time, the Platform object from the loading procedure (see [How to Automate the Loading and Starting of Simulation Applications](#) on page 51) is used to get the SimulationApplication object. The related simulation application is stopped and later on restarted.

Related topics**Basics**

Basics on Platforms.....	21
Basics on Simulation Applications.....	22
Basics on the Platform Management Library.....	49

HowTos

How to Automate the Termination of Simulation Applications	55
--	----

References

GetSimulationApplication.....	147
StartSimulation.....	152
StopSimulation.....	153

How to Automate the Termination of Simulation Applications

Objective

To put a [Platform](#) in a safe state and mark it as being available for new tasks.

Preconditions

You have loaded a simulation application to a platform by using the Platform Management library and fulfilled all the preconditions for this. For more information, refer to [How to Automate the Loading and Starting of Simulation Applications](#) on page 51.

Restrictions

This functionality is available on platforms that support execution control of [simulation applications](#). For more information, refer to [Basics on Simulation Applications](#) on page 22.

Currently, SCALEXIO platforms and VEOS support this feature.

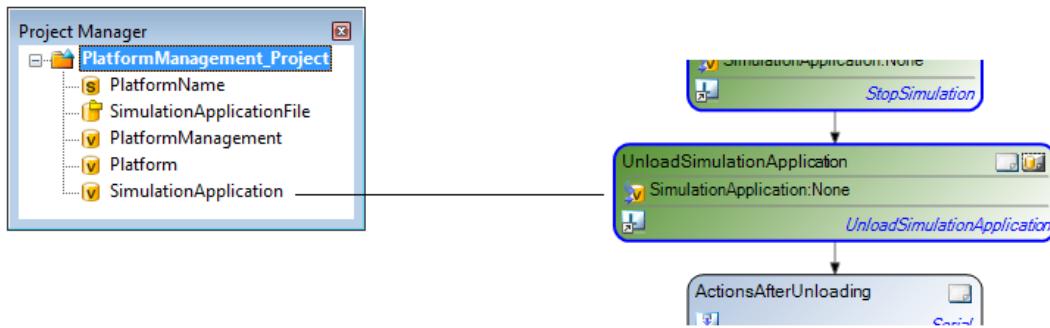
Method**To automate the termination of simulation applications**

- 1 Drag an UnloadSimulationApplication block from the Platform Management library to the Sequence Builder. This unloads the specified simulation application.

- 2 Set the block's SimulationApplication data object as a reference to the project-specific Variant data object that stores the instantiated SimulationApplication object.

Result

You created a sequence that automatically unloads a specified simulation application from the platform it had been loaded to.



At execution time, the Platform object from the loading procedure (see [How to Automate the Loading and Starting of Simulation Applications](#) on page 51) is used to get the SimulationApplication object. The related simulation application is unloaded from the platform and can no longer be accessed.

Related topics

Basics

Basics on Platforms.....	21
Basics on Simulation Applications.....	22
Basics on the Platform Management Library.....	49

References

GetSimulationApplication.....	147
UnloadSimulationApplication.....	154

Working with an XIL API Framework

Introduction	You can configure the access to variables of the simulation application centrally for all AutomationDesk projects by using an XIL API Framework.
---------------------	--

Where to go from here	Information in this section
	<p>Basics on Working with an XIL API Framework..... 57 General information on configuring and initializing an XIL API Framework.</p> <p>How to Create a Framework Configuration..... 59 Instructions for editing an XIL API Framework configuration file.</p> <p>How to Add a Port to a Framework Configuration..... 61 Instructions for configuring ports for an XIL API Framework.</p> <p>How to Create a Mapping File and Add it to a Framework Configuration..... 64 Instructions for managing which mapping files are used in an XIL API Framework configuration.</p> <p>How to Specify the Variable Mapping of a Framework Configuration..... 66 Instructions for configuring the variable mapping for an XIL API Framework.</p> <p>How to Initialize an XIL API Framework..... 68 Instructions for initializing an XIL API Framework as specified in a framework configuration file.</p> <p>How to Migrate Projects to Use XIL API Framework..... 70 Instructions for migrating project-specific testbench configurations to a framework configuration.</p> <p>How to Work with a Third-Party XIL API Framework..... 71 Instructions for working with a third-party XIL API implementation.</p>

Basics on Working with an XIL API Framework

Introduction	By using an XIL API Framework  , you can configure the access to the test system centrally for all AutomationDesk projects  . For example, the access to the variables  of the simulation application  .
---------------------	--

XIL API Testbench and XIL API Framework

The XIL API standard defines two layers for accessing the test infrastructure:

- The [XIL API Testbench layer](#) 

This layer lets you configure ports of different types so you can access variables in a simulation application, for example. This decouples test software from test hardware. If the hardware changes, only the related port configuration has to be adjusted. In AutomationDesk, the port configuration and value mapping is project-specific.

- The [XIL API Framework layer](#) 

This layer provides a component that centrally holds the access information for the entire test infrastructure. Changes in the configuration apply to all open AutomationDesk projects.

After you specified the configuration, you can parameterize data objects by selecting valid values from a list. This is more convenient and avoids typing errors.

Earlier AutomationDesk versions supported the XIL API Testbench. As of AutomationDesk 5.5, XIL API Framework is supported as well and you can choose which layer to work with. If an XIL API Framework is initialized, the related automation blocks use the framework. Otherwise, the XIL API Testbench is used.

Tip

- It is recommended to work with the XIL API Framework.
- For information on migrating from XIL API Testbench to XIL API Framework, refer to [How to Migrate Projects to Use XIL API Framework](#) on page 70.

For more information on the ASAM XIL API standard, refer to:

- [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#)
- [ASAM_AE_XIL_Generic-Simulator-Interface_BS-2-4_CSharp-API-Technology-Reference-Mapping-Rules_V2-1-0.pdf](#)

Typical workflow

The following steps describe a typical workflow for implementing automated variable access:

- Specify the framework configuration in the following XML files by using the Mapping Editor:
 - Framework configuration file
Refer to [How to Create a Framework Configuration](#) on page 59.
 - Port configuration files
Refer to [How to Add a Port to a Framework Configuration](#) on page 61.
 - Mapping files
Refer to [How to Create a Mapping File and Add it to a Framework Configuration](#) on page 64 and [How to Specify the Variable Mapping of a Framework Configuration](#) on page 66.

- Specify which framework configuration file to use and initialize the framework. Refer to [How to Initialize an XIL API Framework](#) on page 68.

Demo projects	Demo framework configuration files are located in <DocumentsFolder>/FrameworkConfiguration, and a demo project that uses an XIL API Framework is located in <DocumentsFolder>/XIL API Convenience. If you want to work with a custom XIL API MAPort platform, check the ControlDesk user documentation for more information. Refer to Example of Using a Custom XIL API MAPort Implementation (ControlDesk Platform Management) .
----------------------	--

How to Create a Framework Configuration

Objective	To configure an XIL API Framework via the Mapping Editor .
Preconditions	<ul style="list-style-type: none"> ▪ Depending on the file you want to edit, the following input data is required: <ul style="list-style-type: none"> ▪ The name of the platform to be accessed. ▪ The path and name of the variable description files (SDF) that specify the simulation applications and their variables to be accessed. ▪ The paths and names of existing port configuration files (XML) that parameterize the ports to be configured. ▪ The path and name of the existing mapping files (XML) that contain information of the variables to be accessed.
Method	<p>To create a framework configuration file</p> <ol style="list-style-type: none"> 1 On the Platforms ribbon, click Framework – Edit. If no XIL API Framework configuration file was specified yet, the Mapping Editor dialog opens and displays an empty file. Otherwise, the Mapping Editor displays the file that is specified via the Configure command. 2 In the File menu, select New. The New Configuration dialog opens. 3 In the Framework configuration file edit field, specify the path and name of the framework configuration file to be created. 4 If you want to create a model access port in the framework configuration according to a variable description file that you specify in this dialog, select Create default MAPort. The Platform and the Port Configuration file fields are available.

If you want to specify other ports, clear Create default MAPort. You can specify other ports and additional model access ports later. Refer to [How to Add a Port to a Framework Configuration](#) on page 61.

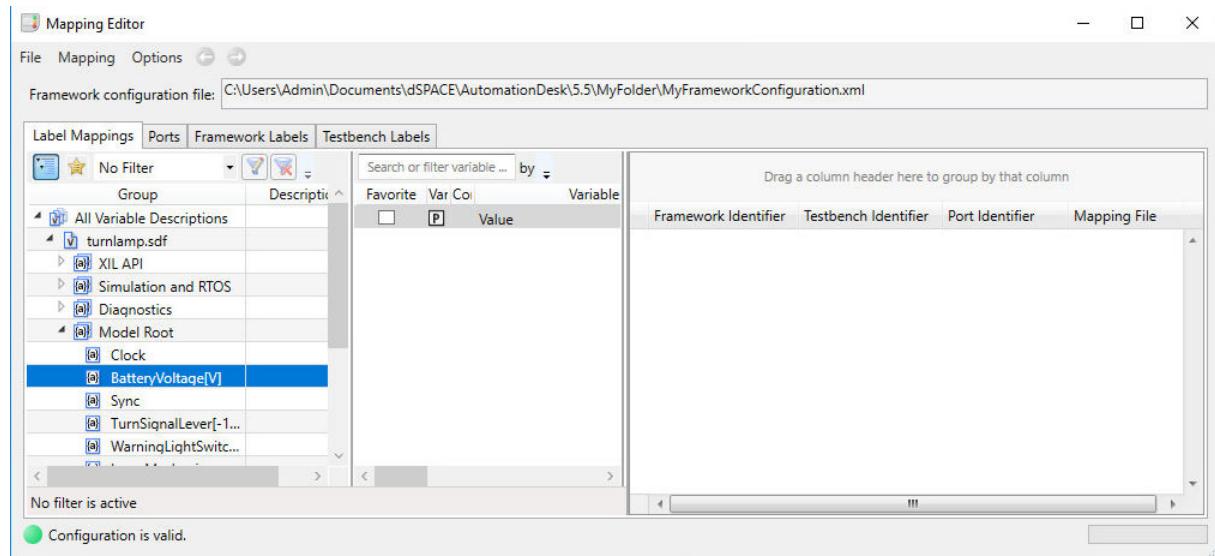
- 5 From the Platform list, select the type of the platform to be accessed.
- 6 In the Port configuration file edit field, specify the path and name of the variable description file (SDF) of the simulation application or the port configuration file (XML) of the port to be accessed.
- 7 Click OK to close the dialog.
- 8 Press Ctrl+S to save the framework configuration.

The XIL API Framework configuration file and the related mapping files are saved. If you created a model access port for dSPACE platforms, a corresponding port configuration file is generated and also saved.

Result

You created a framework configuration file. It is displayed in the Mapping Editor.

For the following illustration the variable description file of the AutomationDesk turn-signal demo simulation application is used. On the Label Mappings page, you can browse the model tree. The mapping table is empty.



On the Ports page, the created model access port is displayed.

Tip

- Optionally, you can specify an existing mapping file (XML) in the Import mapping edit field of the New Configuration dialog to add it to the framework configuration. You can display the list of mapping files that are currently part of the framework configuration via Edit Mapping Files.
- If you specified a mapping file, its content is displayed on the Label Mappings, Framework Labels and Testbench Labels pages.
- By default, the Framework – Edit command on the Platforms ribbon opens the framework configuration that is specified via the Framework – Configure command.

You can now specify which mapping files to use in the framework configuration. Refer to [How to Create a Mapping File and Add it to a Framework Configuration](#) on page 64.

Next step

If you want to add an additional model access port or another port to the framework configuration, refer to [How to Add a Port to a Framework Configuration](#) on page 61.

Otherwise, you can now create a mapping file or add an existing mapping file to the framework configuration. Refer to [How to Create a Mapping File and Add it to a Framework Configuration](#) on page 64.

Related topics

Basics

[Basics on the Mapping Editor \(AutomationDesk Basic Practices\)](#)

References

[Mapping Editor \(AutomationDesk Basic Practices\)](#)

[Mapping Editor Dialog \(AutomationDesk Basic Practices\)](#)

[New \(Framework Configuration\) \(AutomationDesk Basic Practices\)](#)

How to Add a Port to a Framework Configuration

Objective

To configure ports for an XIL API Framework via the [Mapping Editor](#).

Preconditions

- The framework configuration is open in the Mapping Editor. Refer to [How to Create a Framework Configuration](#) on page 59.

- Depending on the port type, the following information is required as input data:
 - The name of the [platform](#) to be accessed.
 - If you add a [model access port](#) for a dSPACE platform: The path and name of the [variable description file](#) (SDF) that specifies the simulation applications to be accessed, including their variables.
 - For all other port types: The paths and names of existing port configuration files (XML) that parameterize the ports to be configured.

Method	To add a port to a framework configuration
	<ol style="list-style-type: none">1 In the Mapping Editor, select the Ports page.2 From the context menu in the table of ports, select New Element. The New Port dialog opens.3 In the Type field, select the type of the port to be added. If you select MAPort to specify a model access port, the Platform field and the SDF file field of the dialog are available. Otherwise, the Port configuration file field is available.4 In the Name field, enter a name for the new port.5 In the Testbench field, select the installed XIL API implementation to be accessed.6 If you add an MAPort, the Platform edit field is available to select the type of platform on which the simulation application is running. If you select the platform type Other, the Port configuration file field is available instead of the SDF file field. This lets you specify MAPorts for non-dSPACE platforms.7 If you add an MAPort for dSPACE platforms, the SDF file edit field is available to specify the path and name of the variable description file (SDF) of the simulation application to be accessed.8 If you add a port of a type other than an MAPort for dSPACE platforms, the Port configuration file field is enabled to specify the path and name of the port configuration file (XML) of the port to be added.9 In the Target state field, select the state to which the port is set when it is initialized.10 Click OK to close the dialog.

Result

You added the specified port to the port table on the Ports page.

Name	Type	Vendor	Product	Product Version	Init Order	Shutdown Order	Target State	Configuration File
dSPACE VEOS MAPort	MAPort	dSPACE GmbH	XIL API	2017-B	0	0	eSIMULATION_RUNNING	VEOS\MAPortConfigVEOS.xml

Configuration is valid.

If you specified a variable description file, the model paths of the variables are displayed in the model tree on the Label Mappings page and the model access port is displayed on the Ports page.

Tip

- You can modify the port's properties in the related table cells.
- You can delete a port via Delete Element in the table's context menu.
- You can use port configuration files that you use with the dSPACE XIL API Implementation if you change their file name extension from PORTCONFIG to XML.

Next step

You can now create a mapping file or add an existing mapping file to the framework configuration. Refer to [How to Create a Mapping File and Add it to a Framework Configuration](#) on page 64.

Related topics**Basics**

[Basics on the Mapping Editor \(AutomationDesk Basic Practices\)](#)

References

[Mapping Editor \(AutomationDesk Basic Practices\)](#)
[New Element \(Port\) \(AutomationDesk Basic Practices\)](#)

How to Create a Mapping File and Add it to a Framework Configuration

Objective

To specify which mapping files are used in an [XIL API Framework](#) configuration.

Possible methods

For a framework configuration, the mapping of variable aliases to model paths is stored in mapping files (XML). A framework configuration can consist of multiple mapping files. The mapping files that are part of the framework configuration are configured in the configuration's mapping file list. The same mapping file can be used in multiple framework configurations.

You can add mapping files to the framework configuration via the following methods:

- You can create and add a new mapping file to edit the variable mappings later in the Mapping Editor. Refer to Method 1.
- You can add an existing mapping file, for example, if you want to use the same mapping file in multiple framework configurations. Refer to Method 2.
- You can export the contents of a [Mapping](#) data object to a mapping file. Refer to [How to Migrate Projects to Use XIL API Framework](#) on page 70.

Precondition

The framework configuration is open in the [Mapping Editor](#). Refer to [How to Create a Framework Configuration](#) on page 59.

Method 1**To create a mapping file and add it to a framework configuration**

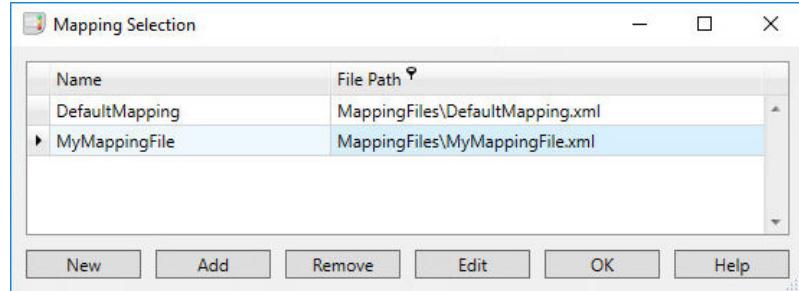
- 1 From the Mapping Editor's Mapping menu, select Edit Mapping Files. The Mapping Selection dialog opens and displays a list of the mapping files that are open in the Mapping Editor.

Note

The Mapping Selection dialog also opens automatically when you add the first mapping by dragging a variable from the model tree to the mapping table on the Label Mappings page.

- 2 Click New and specify the path and name of the mapping file (XML) to be created.

The name and file path are added to the list of open mapping files.



- 3 Click OK to close the dialog.

Method 2

To add an existing mapping file to a framework configuration

- 1 From the Mapping Editor's Mapping menu, select Edit Mapping Files. The Mapping Selection dialog opens and displays a list of the mapping files that are open in the Mapping Editor.
- 2 Click Add and specify the path and name of the mapping file (XML) to open. The name and file path are added to the list of opened mapping files.
- 3 Click OK to close the dialog.

Result

You added another mapping file to the framework configuration.

You can use it when you specify to which file a specific mapping information is written. You can select the added file in the Mapping File settings in the Mapping Editor.

If you added an existing mapping file, the contained information is added to the tables on the Label Mappings, Framework Labels, and Testbench Labels pages.

Tip

- If you use the Remove button in the Mapping Selection dialog to remove a file from the list of open mapping files, the file is updated and closed. Then, the related mapping information is removed from the Label Mappings, Framework Labels, and Testbench Labels pages. The mapping file is not deleted from the file system.
You cannot remove all files from the list of mapping files. There must be at least one file for selection.
- If you use the Edit button to rename a mapping file, the file with the former name is not deleted.
Refer to [Edit Mapping Files \(AutomationDesk Basic Practices\)](#).

Next step	You can now specify additional mappings of the variables' model paths to aliases. Refer to How to Specify the Variable Mapping of a Framework Configuration on page 66.
------------------	---

Related topics	Basics Basics on the Mapping Editor (AutomationDesk Basic Practices)
	References Edit Mapping Files (AutomationDesk Basic Practices) Mapping Editor (AutomationDesk Basic Practices)

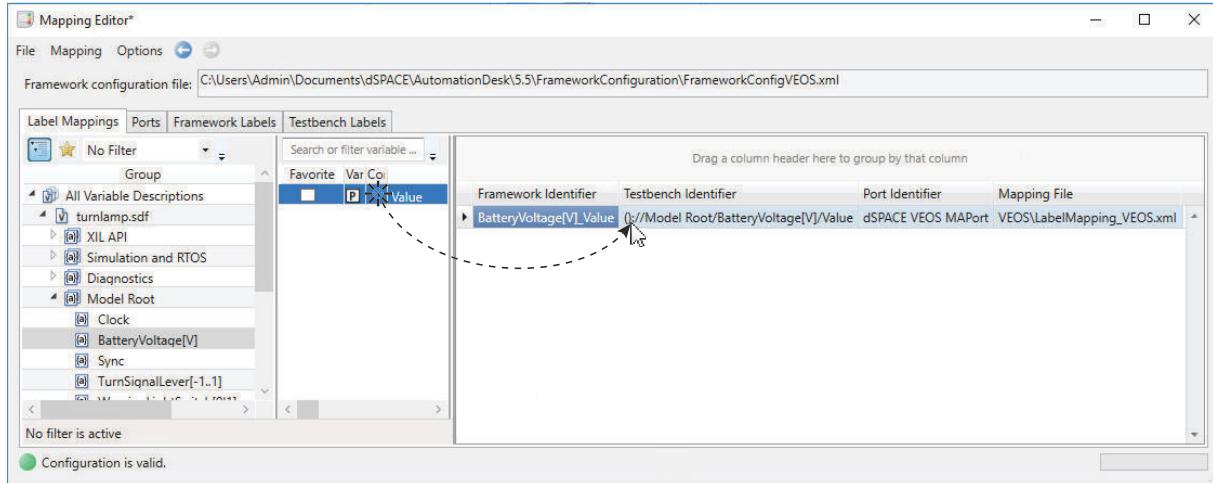
How to Specify the Variable Mapping of a Framework Configuration

Objective	To specify mappings of model paths to variable aliases for a model access port .
------------------	--

Preconditions	<ul style="list-style-type: none">▪ The framework configuration is open in the Mapping Editor. Otherwise, refer to How to Create a Framework Configuration on page 59.▪ The model tree on the Label Mappings page contains the variables that can be accessed via the ports. Otherwise, refer to How to Add a Port to a Framework Configuration on page 61.▪ The Ports page of the Mapping Editor contains the model access ports to be used. For adding a default port, refer to How to Create a Framework Configuration on page 59. For adding another port refer to How to Add a Port to a Framework Configuration on page 61.
----------------------	---

Method	To specify the variable mapping of a framework configuration <ol style="list-style-type: none">1 In the Mapping Editor, open the Label Mappings page.2 In the model tree, navigate to the variable you want to map.
---------------	--

- 3** From the variable list, drag the variable to the mapping table.



A new mapping is added to the mapping table. A unique alias is automatically generated for it and used as the framework identifier. The framework-related properties of the variable are added on the Framework Labels page and the testbench-related properties are added on the Testbench Labels page.

- 4** If you use the SDF file that provides the variable description in the variable list for the first time, the Mapping Selection dialog opens. In this dialog, select the mapping file to which you want to save the new variable mappings by default and click OK.
 - 5** Repeat the steps 2 and 3 for all variables that you want to access via the model access port.
 - 6** After you finished editing the framework configuration, click Close in the File menu and confirm to save the related framework configuration.
- The XIL API Framework configuration file and the related mapping files are saved. The Mapping Editor is closed.

Result

You specified the variable mapping and saved it together with the framework configuration.

Tip

- You can also add manually edited mappings to the mapping list. To do this, specify the variable properties on the Framework Labels and Testbench Labels pages. Then, you can add a new mapping by using the New Element command of the mapping table's context menu.
- The manual specification of mappings is facilitated by the consistency check of the Mapping Editor. It highlights fields that cause warnings in yellow and fields that cause errors in red.
- You can modify each field of the variable mapping by selecting another defined value for the field.
- You can open the page with the definition of the current value by using the Goto Definition command in a field's context menu.
- You can change the variable alias simultaneously in the mapping table and on the Framework Labels page by using the Rename command in a Framework Identifier field's context menu.

Refer to [Mapping Editor Dialog \(AutomationDesk Basic Practices\)](#).

Note

You can map different aliases to the same model path, but you cannot map different model paths to the same alias.

You can now initialize the XIL API Framework. Refer to [How to Initialize an XIL API Framework](#).

Related topics

Basics

[Basics on the Mapping Editor \(AutomationDesk Basic Practices\)](#)

References

[Mapping Editor \(AutomationDesk Basic Practices\)](#)

How to Initialize an XIL API Framework

Objective

To initialize an [XIL API Framework](#) as configured in a framework configuration file.

Preconditions	<ul style="list-style-type: none">▪ The platform to be used must be registered. For more information, refer to How to Register a dSPACE Platform on page 27.▪ The configuration of the framework must be specified in an XIL API Framework configuration file (XML). The path and name of the file is required as input data. For more information, refer to How to Create a Framework Configuration on page 59.
Method	<p>To initialize an XIL API Framework</p> <ol style="list-style-type: none">1 On the Platforms ribbon, click Framework – Configure. The XIL API Framework Configuration dialog opens.2 From the Framework implementation list, select the installed XIL API implementation to be accessed via the framework.3 In the Framework configuration edit field, specify the path and name of the existing framework configuration file.4 If you want the framework to be initialized automatically when AutomationDesk starts, select the Initialize framework on startup option.5 Click OK to close the dialog.6 On the Platforms ribbon, click Framework – Initialize.
Result	<p>You initialized the XIL API Framework with the specified framework configuration. This is indicated by the following:</p> <ul style="list-style-type: none">▪ On the Platforms ribbon, the Initialize icon is unavailable and the Shutdown icon is available.▪ If there are Mapping data objects in the projects that are open in the Project Manager, their icons turn from  to .▪ In the Mapping Viewer, the first line displays the path and name of the used XIL API Framework configuration file. <p>You can now parameterize automation blocks of the XIL API Convenience library by selecting information from the framework configuration. For more information, refer to XIL API Convenience (Model Access) on page 160.</p>

Note

The framework is shut down automatically when AutomationDesk is closed. Before you initialize the framework with a modified framework configuration or if you want to return to using the XIL API Testbench, you must explicitly shut down the framework. Refer to [Shutdown \(XIL API Framework\)](#) on page 350.

Related topics**References**

Configure (XIL API Framework).....	343
Initialize (XIL API Framework).....	347
Shutdown (XIL API Framework).....	350

How to Migrate Projects to Use XIL API Framework

Objective

You can migrate projects that used the [XIL API Testbench](#) for accessing simulation applications to use the [XIL API Framework](#) instead.

Preconditions

- The platform to be used must be registered. For more information, refer to [How to Register a dSPACE Platform](#) on page 27.
- No XIL API Framework must be initialized. Otherwise, refer to [Shutdown \(XIL API Framework\)](#) on page 350.

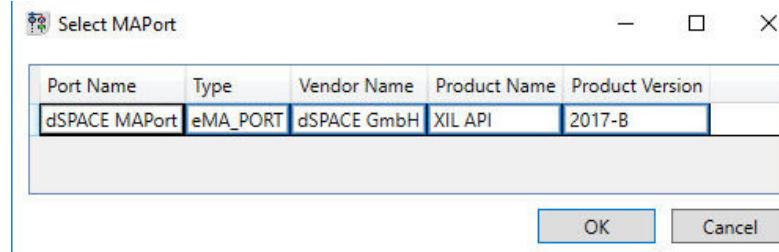
Basics

You can transfer the port configuration of a MAPortConfiguration data object and the variable mapping in a Mapping data object to the configuration for an XIL API Framework.

Method**To migrate projects to use XIL API Framework**

- 1 Create a framework configuration file, including a default model access port. To specify the port, use the same platform and variable description file as in the previously used MAPortConfiguration data object. Refer to [How to Create a Framework Configuration](#) on page 59.
- 2 In the Project Manager, select the Mapping data object of your project.
- 3 From the Mapping data object's context menu, select [Export XIL API Mapping](#), specify a mapping file (XML) to contain the variable mapping, and click OK.
The specified mapping file is created.
- 4 In the Mapping Editor, add the mapping file to the framework configuration. Refer to [How to Create a Mapping File and Add it to a Framework Configuration](#) on page 64.
- 5 Save the framework configuration and close the Mapping Editor.
- 6 Initialize the framework. Refer to [How to Initialize an XIL API Framework](#) on page 68.
- 7 In the Project Manager, double-click the MAPortConfiguration data object. The Select MAPort dialog opens.

- 8 In the dialog, select the port to be accessed and click **OK** to specify the MAPortConfiguration data object.

**Result**

You migrated your project that formerly used the XIL API Testbench for accessing a simulation application to now use the XIL API Framework.

Related topics**References**

Configure (XIL API Framework).....	343
Initialize (XIL API Framework).....	347

How to Work with a Third-Party XIL API Framework

Objective

You can configure the access to platforms centrally via a [framework](#) that is provided by a third-party XIL API server.

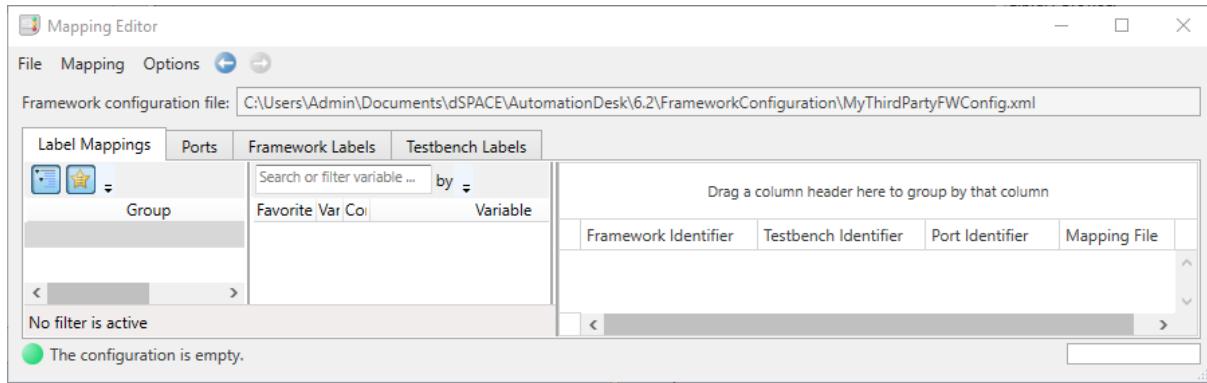
Preconditions

- The third-party [XIL API implementation](#) must be installed and accessible. Refer to the documentation that is provided by the vendor.
- The following information is required as input data:
 - The path and name of the required vendor-specific [model access port](#) configuration file
 - If applicable, the path and name of mapping files (XML)

Method**To work with a third-party XIL API framework**

- 1 Create a new framework configuration as described in [How to Create a Framework Configuration](#) on page 59. In the New Configuration dialog, specify the following settings:
 - Clear the **Create default MAPort** option.
 - From the **Platform** list, select **Other** as the type of the platform to be accessed.

An empty framework configuration opens in the Mapping Editor.



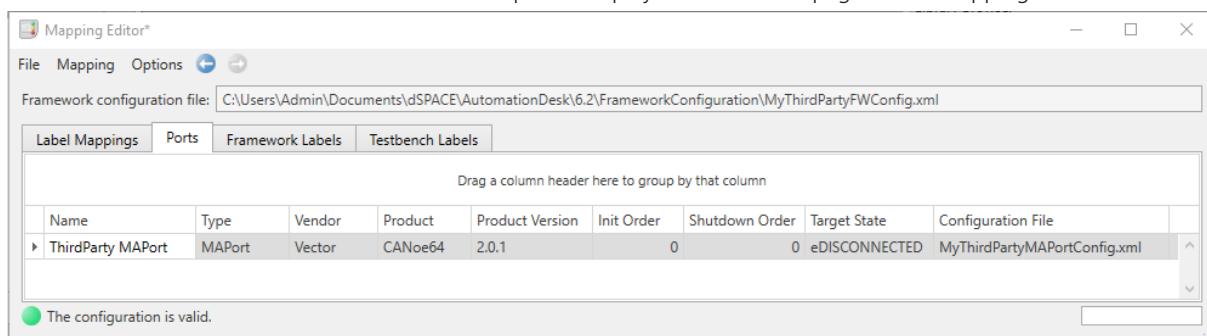
- 2 Add a new MAPort to the framework configuration as described in [How to Add a Port to a Framework Configuration](#) on page 61. In the New Port dialog, specify the following settings:
 - From the Type list, select MAPort as the type of the new port.
 - In the Name edit field, specify a name for the new port.
 - From the Testbench list, select the installed XIL API server you want to access.

Note

If the third-party XIL API implementation is not provided in the list of available testbenches, the XIL API implementation is not properly installed or accessible. Refer to the vendor's documentation.

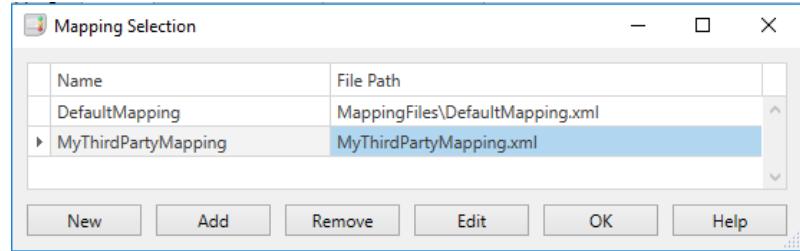
- In the Port configuration edit field, specify the path and name of the third-party model access port configuration file (XML).

The new port is displayed on the Port page of the Mapping Editor.

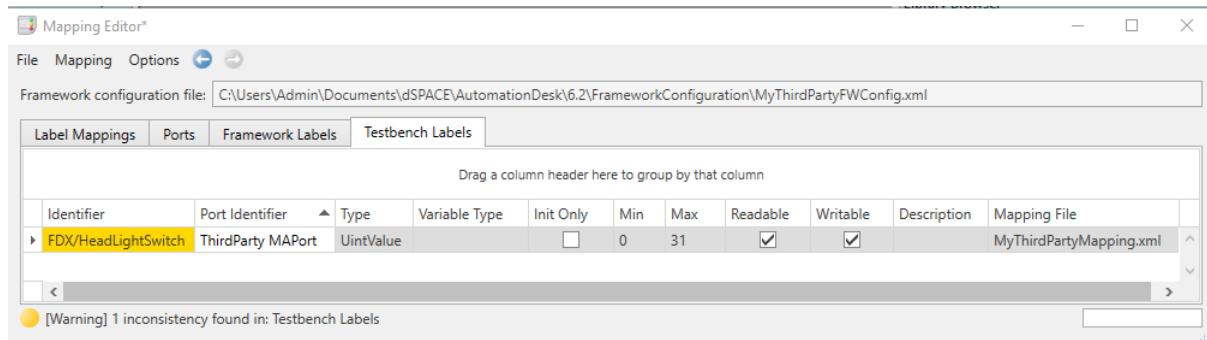


- 3 Add a new mapping file to the framework configuration. Refer to [How to Create a Mapping File and Add it to a Framework Configuration](#) on page 64.

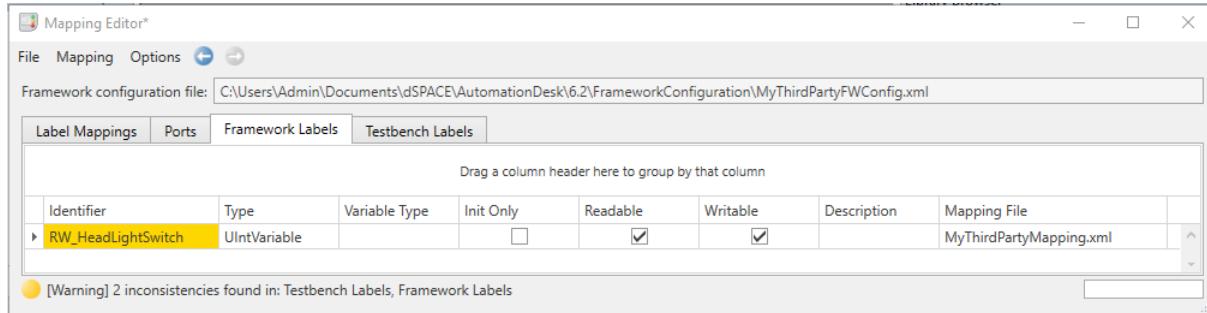
The new mapping file is contained in the Mapping Editor's mapping file list.



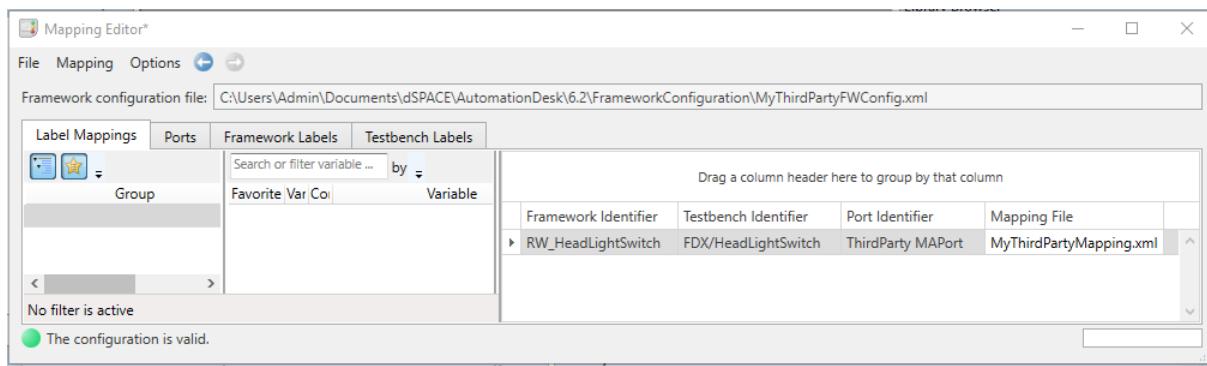
- 4 On the Testbench Labels page: For each variable that you want to access, add a new item to the table by using the New Element command on the context menu and specify the variable's testbench properties.



- 5 On the Framework Labels page: For each variable, add a new item to the table and specify the variable's framework properties.



- 6 On the Label Mappings page: For each variable, add a new item to the table and map the variable's Framework Identifier to its Testbench Identifier and to the Mapping File where this information is stored. You can specify the values by selection.



- 7 Initialize the third-party XIL API Framework as described in [How to Initialize an XIL API Framework](#) on page 68.

Result

You configured and initialized the third-party XIL API framework. You can work with it, for example, when you automate platform access by using automation blocks provided by the XIL API Convenience library. Refer to [Accessing Simulation Platforms via the XIL API Convenience Library](#) on page 75.

Related topics

Basics

Basics on Working with an XIL API Framework.....	57
--	----

Accessing Simulation Platforms via the XIL API Convenience Library

Introduction

AutomationDesk's XIL API Convenience library provides automation blocks that let you automate the access to simulation platforms.

Where to go from here

Information in this section

Basics on Accessing Simulation Platforms via the XIL API Convenience Library	76
General information for accessing simulation platforms via the XIL API Convenience library.	
How to Build a Basic Sequence for Accessing Simulator Variables	78
Instructions for building a sequence that performs the generic actions required for accessing simulator variables.	
How to Download and Start a Simulation Application	81
Instructions for downloading a simulation application to a platform and starting it.	
How to Make Simulator Variables Available	84
Instructions for specifying the mapping of the variables' model paths to aliases.	
How to Write Variable Values	87
Instructions for writing variable values to a running simulation application.	
How to Read Variable Values	91
Instructions for reading variable values of a running simulation application.	
How to Capture Data	96
Instructions for capturing the values of simulator variables over a period of time.	
How to Add Plots of the Captured Data to your Report	102
Instructions for adding plots of the captured data to the report.	
How to Release Resources After Data Capturing	105
Instructions for releasing no longer needed resources after capturing.	
How to Specify Signals for Stimulating	106
Instructions for specifying the signals for variable stimulation.	
How to Stimulate Simulator Variables	108
Instructions for stimulating the value of a simulator variable over a period of time.	

[How to Release Resources after Variable Stimulation.....](#) 112

Instructions for stopping signal generation and releasing no longer needed resources.

[How to Access Platforms via a Third-Party XIL API Server.....](#) 114

Instructions for working with a third-party XIL API implementation.

Basics on Accessing Simulation Platforms via the XIL API Convenience Library

Introduction

The provided [automation blocks](#) let you download and start a [simulation application](#) and access its [variables](#) during execution.

Basic concept of the XIL API Convenience library

The following [folders](#) are provided at the [library's](#) top level:

- The Electrical Error Simulation Port folder contains automation blocks for accessing connected electrical error simulation hardware. For more information on these blocks, refer to [Basics on Simulating Electrical Errors via the XIL API Convenience Library \(AutomationDesk Simulating Electrical Errors\)](#)
- The Model Access Port folder contains automation blocks to access a registered simulation platform. The usage of these blocks is described in the following.

The XIL API Convenience library's blocks aggregate blocks of the XIL API library to provide a convenient way to automate the stimulation and capture of simulator variables. For typical use cases in test automation it is recommended to use them rather than the XIL API library.

The XIL API Convenience library is implemented as a write-protected custom library. It provides templates of Exec blocks that use data objects of the XIL API library.

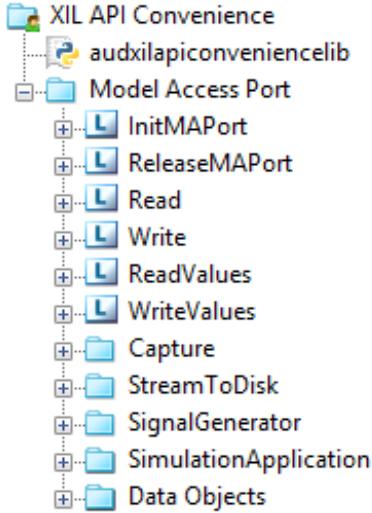
The XIL API Convenience and the XIL API library are both based on the ASAM XIL API standard, i.e., you can use them with any hardware-in-the-loop (HIL) system that provides an interface which complies with this standard.

For more information on the ASAM XIL API standard, refer to:

- [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#)
- [ASAM_AE_XIL_Generic-Simulator-Interface_BS-2-4_CSharp-API-Technology-Reference-Mapping-Rules_V2-1-0.pdf](#)

Overview of the Model Access Port folder

The Model Access Port library folder provides the following elements:



Model Access Port This folder provides the main blocks for handling an MAPort instance and reading and writing variables. The used data object types, such as the MAPort, are provided by the XIL API library.

Further automation blocks are grouped into the following library folders.

Capture The Capture folder provides blocks that you can use to capture data, i.e., to read the variable values enhanced by a time stamp periodically. The result of the capture is stored in a CaptureResult data object and you can add a plot of its contents to a report. You can configure when data capturing starts and when it stops by specifying a condition or a duration.

StreamToDisk The StreamToDisk folder provides blocks that you can use to stream the result of the data capture to a file on the host PC.

SignalGenerator The SignalGenerator folder provides blocks that you can use to stimulate simulator variables with a signal. Via the signal, you specify the progression of a variable's value over a period of time.

SimulationApplication The SimulationApplication folder provides blocks that you can use to download, start and stop simulation applications on a registered platform.

Data Objects The Data Objects folder contains templates for data objects with customized edit dialogs. If the framework is initialized, you can specify the data objects' values by selection.

For more information, refer to [XIL API Convenience \(Model Access\)](#) on page 160.

Typical workflow

The following steps describe a typical workflow to implement automated variable access:

- If you use the XIL API Framework, configure the variable access centrally as part of the the framework configuration.

If you use the XIL API Testbench, configure the variable access separately for each project in MAPortConfiguration objects and in Mapping data objects.

- Specify the signals that are used to stimulate variables by using the Signal Editor.
 - Download and start the simulation application on your platform.
 - Parameterize your simulation.
 - Specify condition and durations for data capturing.
 - Perform data capturing and stimulation.
-

Demo projects

You can find some demo projects for the XIL API Convenience library in <DocumentsFolder>/XIL API Convenience.

How to Build a Basic Sequence for Accessing Simulator Variables

Objective

You can build a basic [sequence](#) that performs the generic steps for accessing a signal [Selector](#) that is running on a [Platform](#).

Preconditions

- The platform to be used must be registered. For further information, refer to [How to Register a dSPACE Platform](#) on page 27.
 - The following information is required as input data:
 - The name of the platform to be accessed
 - The path and name of the [Variable description file](#) (SDF) that specifies the simulation application to be downloaded
 - If you work with an [XIL API Framework](#):
 - A framework configuration including a default [model access port](#) must be created. Refer to [How to Create a Framework Configuration](#) on page 59
 - The framework must be initialized. Refer to [How to Initialize an XIL API Framework](#) on page 68.
-

Method

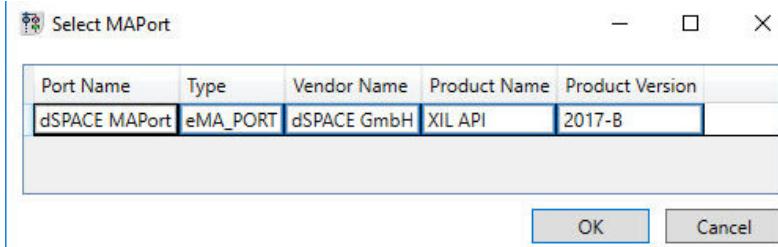
To build a basic sequence for accessing a simulation variables

- 1 On the Home ribbon, click Insert – Data Objects – MAPortConfiguration. This adds a data object to your project that will contain the specification of the platform and the simulation application to be accessed.

2 Specify the model access port configuration:

- If an XIL API Framework is initialized:

Double-click the MAPortConfiguration data object to open the Select MAPort dialog.

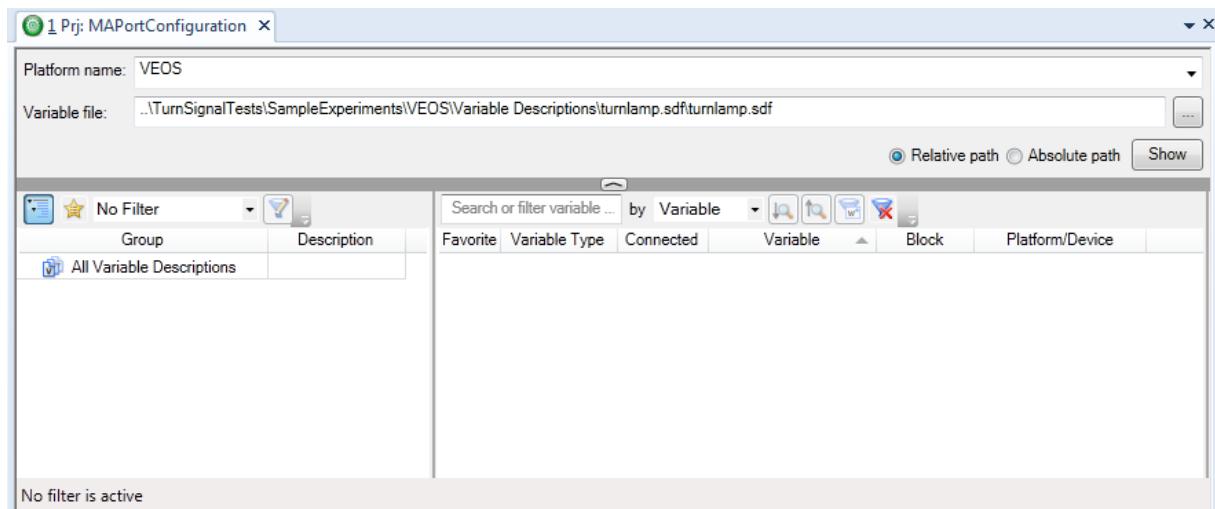


All the model access ports that you configured in the framework configuration are displayed. Select the model access port you want to access and click OK.

- If you work with the XIL API Testbench:

Open the MAPortConfiguration data object in the Variables pane and specify the platform name and the variable description file (SDF) of the simulation application to be accessed.

The illustration shows the specification of AutomationDesk's turn signal demo for VEOS that is used in the following as an example.



3 On the Home ribbon, click Insert — Data Objects — MAPort. This adds a data object to your project that will contain the instantiated model access port data object.

4 Drag a TryFinally block from the Main Library to your sequence. This ensures, that the blocks specified in the Finally path are executed after the Try path execution, even if an exception occurred.

5 Drag an InitMAPort block from the XIL API Convenience library's Model Access Port folder to the Try path. This initializes the access to the application that is running on the simulation platform.

By default, the block's MAPortConfiguration and MAPort data object are set as a reference to the related project-specific data object.

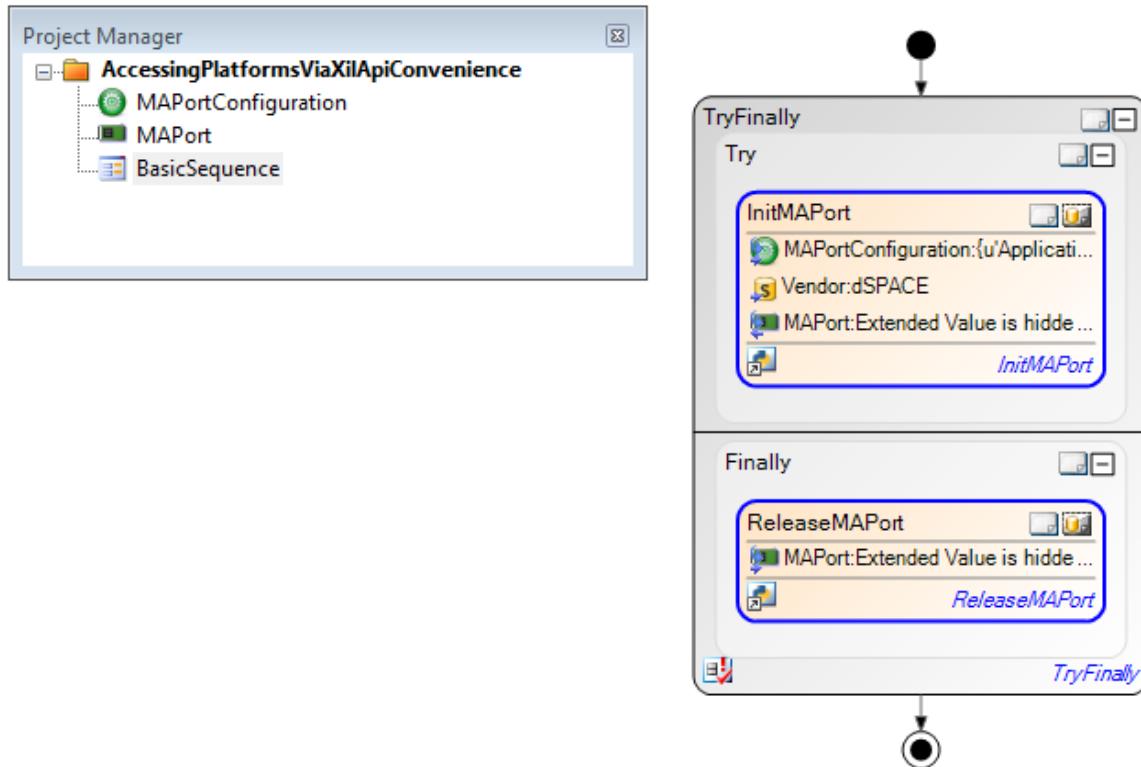
- 6 Only if you work with the XIL API Testbench, drag a ReleaseMAPort block to the Finally path. This releases the MAPort instance.

By default, the block's MAPort data object is set as a reference to the project-specific MAPort data object.

If you work with the XIL API Framework, the MAPort data object is automatically released when the framework is shut down.

Result

You created a basic sequence for automating the access to the variables of a running simulation application. The following illustrations shows the sequence to work with the XIL API Testbench.



If you work with the XIL API Framework, differing from the sequence above, the Finally path is left empty.

Next steps

You can now enhance the basic sequence by automating the start of your simulation application. For instructions, refer to [How to Download and Start a Simulation Application](#) on page 81.

Related topics**References**

InitMAPort.....	162
MAPort (Data Object).....	286
MAPortConfiguration (Data Object).....	289
ReleaseMAPort.....	167
TryFinally (AutomationDesk Basic Practices)	

How to Download and Start a Simulation Application

Objective

You can automate the download and start of a [simulation application](#) on a [platform](#) via the configuration of the XIL API Framework or via automation blocks of the XIL API Convenience library.

Possible methods

Depending on the XIL API layer you work with, you can use the following methods:

- If you work with the [XIL API Framework](#), you can download and start a simulation application during the initialization of the framework. Refer to Method 1.
- If you work with either [XIL API Testbench](#), you can download and start a simulation application via a [automation block](#). Refer to Method 2.

Preconditions

- For method 1:
 - You registered the platform to be used. For more information, refer to [How to Register a dSPACE Platform](#) on page 27.
 - You configured an XIL API Framework including a model access port. Refer to [How to Create a Framework Configuration](#) on page 59.
- For method 2:

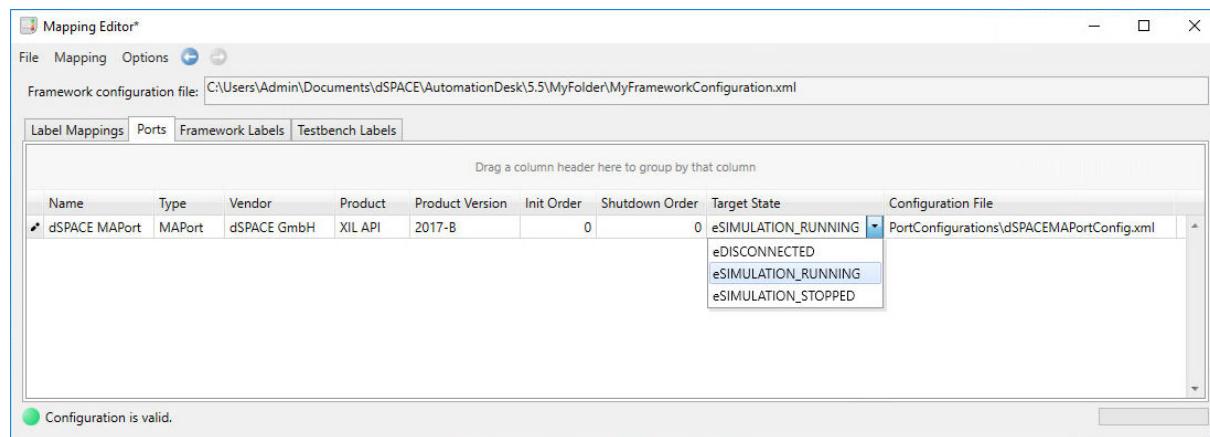
You have built a basic sequence for accessing your platform. For instructions, refer to [How to Build a Basic Sequence for Accessing Simulator Variables](#) on page 78.

Method 1

To download and start a simulation application during framework initialization

- 1 Open the framework configuration in the Mapping Editor.

- 2** On the Ports page, set the Target State of the model access port to eSIMULATION_RUNNING.



- 3** Save the framework configuration and close the Mapping Editor.
4 Use the created framework configuration to initialize the XIL API Framework as described in [How to Initialize an XIL API Framework](#) on page 68.

Method 2

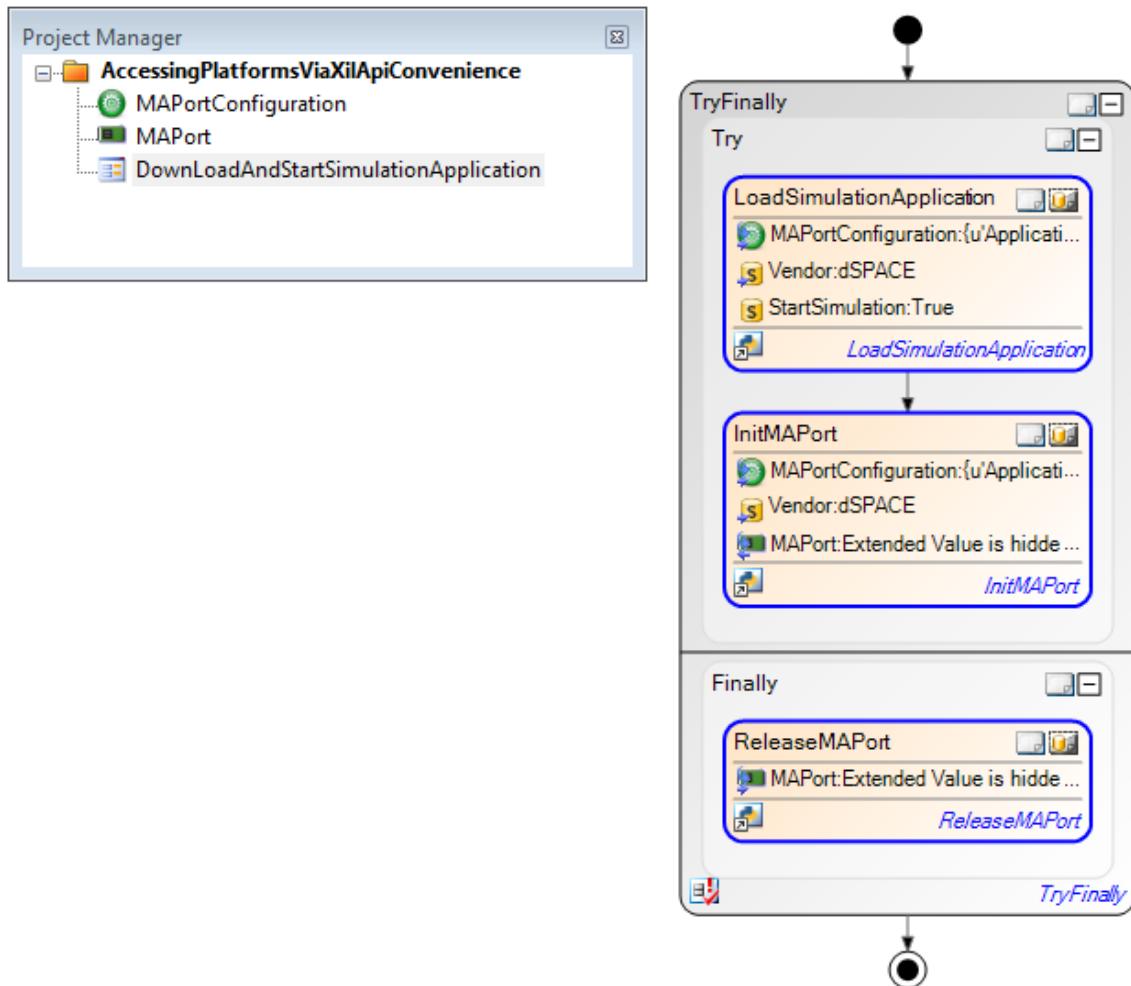
To download and start a simulation application via an automation block

- 1** From the XIL API Convenience library's Model Access Port – SimulationApplication folder, drag a LoadSimulationApplication block to the beginning of the Try path. This downloads and starts the simulation application on the platform.
 By default, the block's MAPortConfiguration data object is set as a reference to the related project-specific data object, and in the StartSimulation data object True is specified. This specifies to automatically start the simulation application after it is downloaded.

Result

Via method 1, you configured the XIL API Framework to download and start the simulation application automatically each time the framework is initialized.

Via method 2, you created a sequence to automate the download and start of a simulation application.



Next steps

Now you can make the simulator variables of the running simulation application available to the blocks of the XIL API Convenience library by specifying a variable pool. For instructions, refer to [How to Make Simulator Variables Available](#) on page 84.

Related topics

References

LoadSimulationApplication.....	204
--------------------------------	-----

How to Make Simulator Variables Available

Objective

You can specify the mapping of the [variables](#)' model paths to aliases for all open projects in a framework configuration or for a specific [project](#) in a [Mapping](#) data object.

Possible methods

Depending on the XIL API layer you work with, you can use the following methods to make simulator variables available:

- If you work with the [XIL API Framework](#), you can specify the variable mapping for all open projects in the framework configuration. Refer to Method 1.
- If you work with [XIL API Testbench](#), you can specify the variable mapping for a specific project in a Mapping data object. Refer to Method 2.

Preconditions

- For Method 1:
 - You configured an XIL API Framework including a model access port. Refer to [How to Create a Framework Configuration](#) on page 59.
 - You specified which files are used as mapping files in the framework configuration. Refer to [How to Specify the Variable Mapping of a Framework Configuration](#) on page 66.
- For Method 2:
 - You created a basic sequence to access the simulation application. For instructions, refer to [How to Build a Basic Sequence for Accessing Simulator Variables](#) on page 78.
 - The model paths of the variables to be accessed are required as input data.

Method 1

To make simulator variables available via the framework configuration

- 1 Open the XIL API framework configuration in the Mapping Editor.
- 2 On the Label Mappings page, specify the variable mapping as described in [How to Specify the Variable Mapping of a Framework Configuration](#) on page 66.
- 3 Save the framework configuration and close the Mapping Editor.
- 4 Use the created framework configuration to initialize the XIL API Framework as described in [How to Initialize an XIL API Framework](#) on page 68.
The variable mapping of the initialized framework is displayed in the Mapping Viewer.

Method 2

To make simulator variables available via a Mapping data object

- 1 In the Project Manager, select your project. On the Home ribbon, click **Insert – Data Objects – Mapping**. This adds a data object to the top level of your project that will contain the variable mapping.

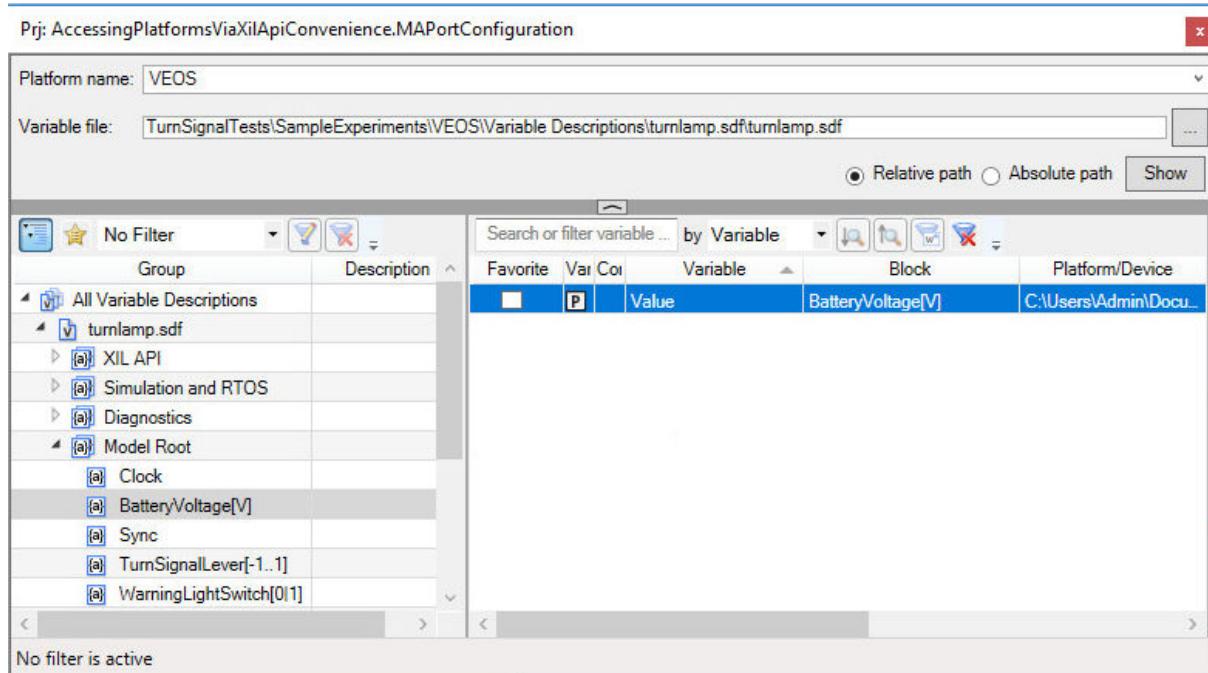
- 2 Click the Mapping data object to open it in the Mapping Viewer.
- 3 In the Project Manager, double-click the MAPortConfiguration data object to open it in the Variables pane.

Note

If you returned from working with the XIL API Framework:

1. Reset the MAPortConfiguration data object by using Clear Value.
2. Open the data object in the Variables pane.
3. In Platform name, specify the platform name.
4. In Variable file, specify the path and name of the variable description file (SDF).

- 4 In the Variables pane, click Show.
- The model tree and variable list display the variables that are contained in the variable description. You can browse the model tree by expanding the tree nodes.
- 5 Browse the model tree and in the variable list, select a variable that you want to access.



- 6 Add the selected variable to the variable mapping by dragging it to the mapping table in the Mapping Viewer.

Mapping Viewer						
AccessingPlatformsViaXilApiConvenience.Mapping						
Alias	Identifier	VariableType	Description	Port	IdentifierType	
BatteryVoltage_V_Value	0://Model Root/BatteryVoltage[V]/Value	Parameter		Port	StringValue	
*						

7 Optionally, you can edit the variable's mapping in the Mapping Viewer.

8 Repeat steps 5 to 7 for all variables you want to access.

The contents of the Mapping data object are displayed in the Mapping Viewer.

Result

With method 1, you specified the variable mapping for all open projects in the framework configuration and with method 2 for a specific project in a Mapping data object.

Both methods make the simulator variables available to the automation blocks of the XIL API Convenience library.

The mapping that is shown below is used for the examples in the following.

Mapping Viewer						
AccessingPlatformsViaXilApiConvenience.Mapping						
Alias	Identifier	VariableType	Description	Port	IdentifierType	
BatteryVoltage_V_Value	0://Model Root/BatteryVoltage[V]/Value	Parameter		Port	StringValue	
TurnSignalLever_1_1_Value	0://Model Root/TurnSignalLever[-1..1]/Value	Parameter		Port	StringValue	
FrontLightEcu_TurnSignalRight	0://Model Root/FrontLightEcu/TurnSignalRight	Measurement		Port	StringValue	
*						

Next steps

You can now use the specified variable pool to read and write the simulator variables of the running simulation application. For instructions, refer to [How to Write Variable Values](#) on page 87.

Related topics

References

Clear Value (AutomationDesk Basic Practices)	
Mapping (Data Object)	209
Mapping Editor (AutomationDesk Basic Practices)	

How to Write Variable Values

Objective

You can write the variable values of a [simulation application](#) that is running on a [platform](#). The library provides [blocks](#) for accessing single [variables](#) or multiple variables at once.

Preconditions

- You created a basic [sequence](#) for platform access and configured the access to the simulator variables in the framework configuration or a [Mapping](#) data object. For instructions, refer to [How to Make Simulator Variables Available](#) on page 84.
- The following information is required as input data:
 - The name of the variables to be written
 - The variables values you want to write

Possible methods

You can write simulator variables by using the following methods:

- You can write a single simulator variable by specifying its model path in a string. Refer to Method 1.
- You can write multiple simulator variables by specifying their aliases and the related values in a dictionary. The aliases are mapped to the required model paths in the project's Mapping data object or in the framework configuration. Refer to Method 2.

Method 1

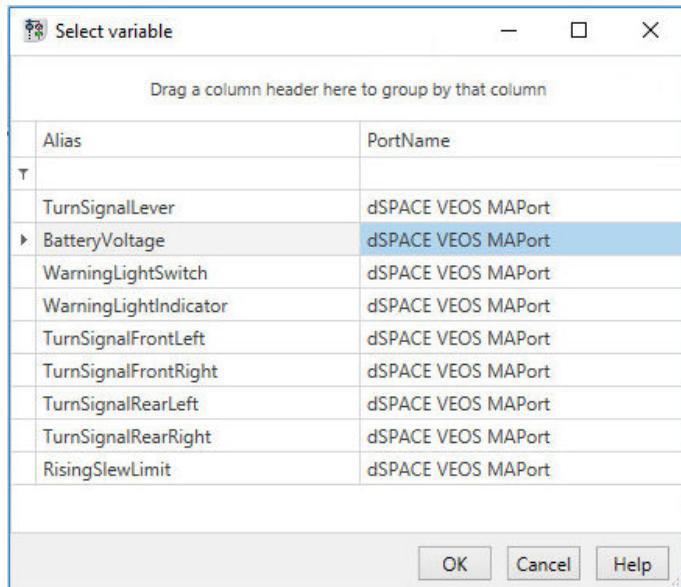
To write a single simulator variable

- 1 From the Model Access Port folder, drag a Write block to the Try path. This writes the specified value to the variable that is identified by the model path of the specified string.

By default, the block's **MAPort** data object is set as a reference to the project-specific **MAPort** data object.

2 Specify the variable to be written:

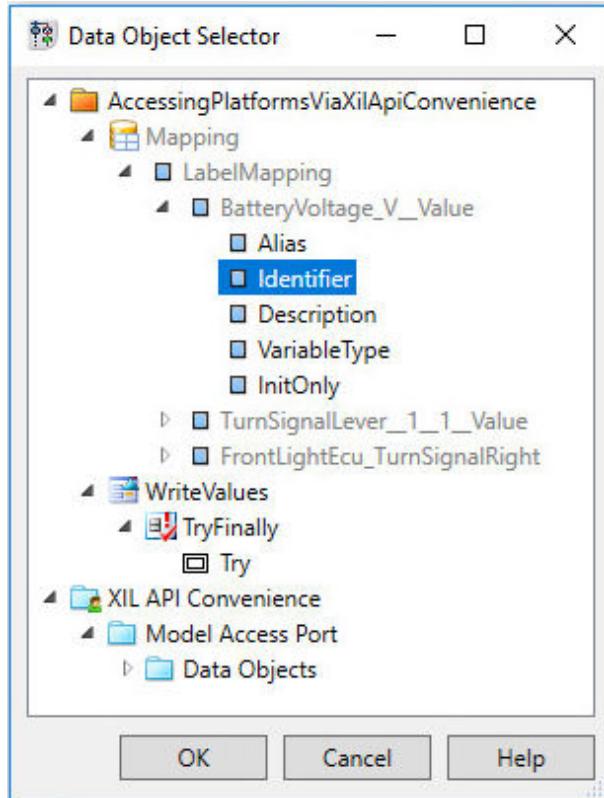
- If an XIL API Framework is initialized, double-click the Variable data object. The Select Variable dialog opens.



Select the variable to be written and click OK.

- If you work with the XIL API Testbench and the variables are available in the Mapping data object, select the Write block in the Sequence Builder.

In the Data Object Editor, click the Reference name edit field for the Variable data object and then the Browse button to open the Data Object Selector.



In the Data Object Selector, select the Mapping - LabelMapping - <VariableAlias> - Identifier object of the variable to be written and click OK.

Alternatively, you can drag the variable from the Mapping Viewer to the block's Variable data object in the Sequence Builder.

The Variable data object is set to the model path of the variable to be written.

- 3 In the block's Value data object, specify the value to be written to the variable.

Method 2

To write multiple simulator variables

- 1 From the Model Access Port folder, drag a WriteValues block to the Try path. This writes the specified variable values to the running simulation application.
By default, the block's MAPort data object is set as a reference to the project-specific MAPort data object.
- 2 If you work with the XIL API Testbench, set the block's VariablePool data object as a reference to the project-specific Mapping data object.

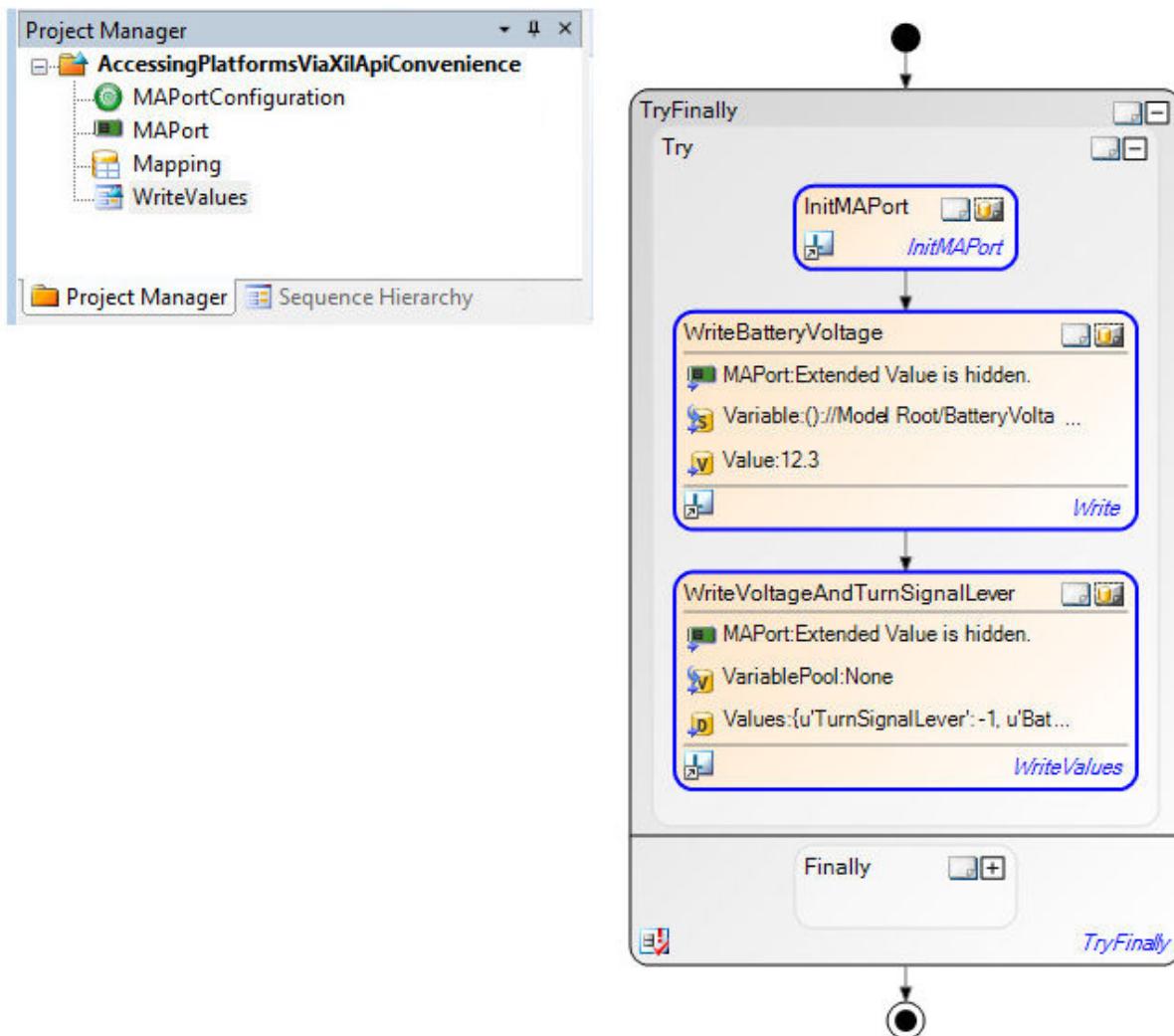
- 3 In the Values data object's key-value pairs, specify the variable aliases and the related values to be written by doing the following.

Drag the variables from the Mapping Viewer to the block's Values data object in the Sequence Builder. This adds the related aliases as keys to the dictionary.

Double-click the dictionary data object to open the Value Editor and specify the variables' values.

Result

You created a sequence that first writes a single value to the running simulation application and then multiple values. The following illustration shows the resulting sequence. If you work with an XIL API Framework, the Mapping data object is optional.



For example, select the battery voltage as the Variable data object of the Write block and specify 12.3 in the block's Value data object.

Set the Values data object of the WriteValues block to {"BatteryVoltage": 13.4, "TurnSignalLever": -1.0}.

Tip

You can write single or multiple simulator variables conveniently by using the SetValues block of the XIL API library. Refer to [How to Write and Read Variables](#) on page 129. It is recommended to use this method.

Next steps

You can read the values of simulator variables, for example, those that you have written. For instructions, refer to [How to Read Variable Values](#) on page 91.

Related topics

References

SetValues.....	158
Write.....	168
WriteValues.....	169

How to Read Variable Values

Objective

You can read variable values of a [signal Selector](#) that is running on a [platform](#). The library provides [blocks](#) for accessing single [variables](#) or multiple variables at once.

Preconditions

- You created a basic sequence for platform access and configured the access to the simulator variables in the framework configuration or a [Mapping](#) data object. For instructions, refer to [How to Make Simulator Variables Available](#) on page 84.
- The following information is required as input data:
 - The name of the variables to be read

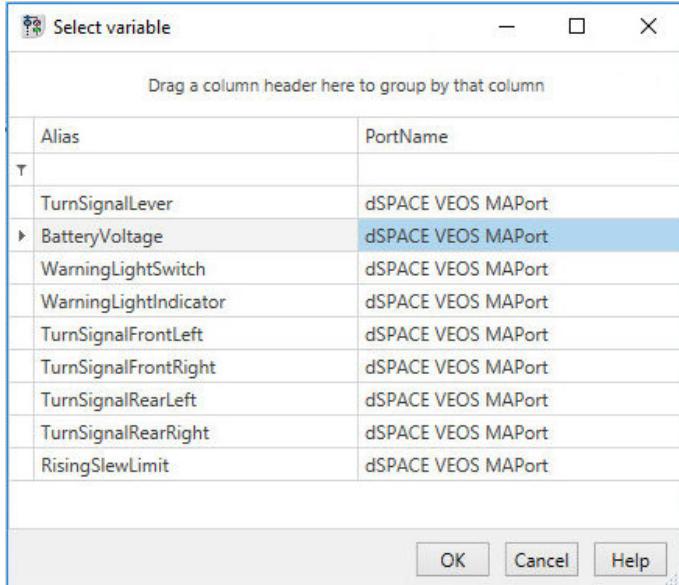
Possible methods

You can read simulator variables by using the following methods:

- You can read a single simulator variable by specifying its model path in a string. Refer to Method 1.
- You can read multiple simulator variables by specifying their aliases and the related values in a dictionary. The aliases are mapped to the required model paths in the project's Mapping data object or in the framework configuration. Refer to Method 2.

Method 1**To read a single simulator variable**

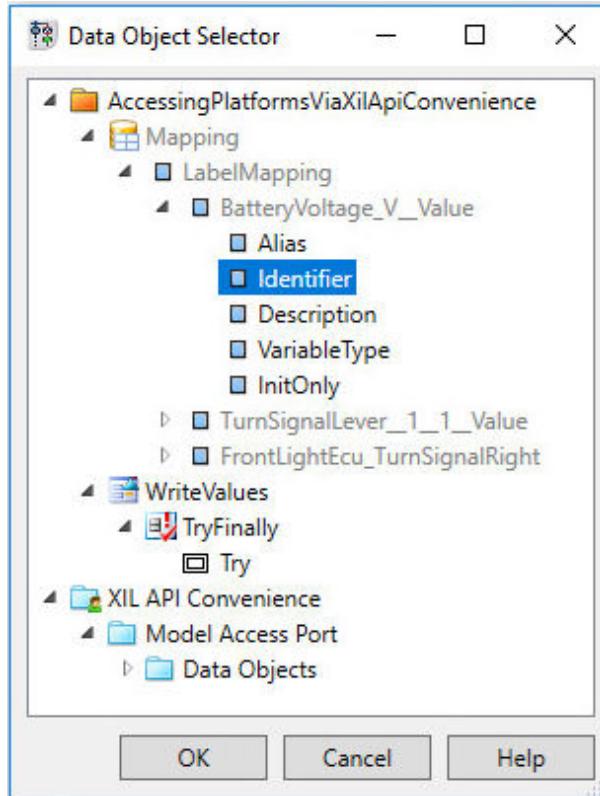
- 1 In the Project Manager, add a data object to your project to store the read value.
- 2 From the Model Access Port folder, drag a Read block to the Try path. This reads the variable that is identified by the model path of the specified string. By default, the block's MAPort data object is set as a reference to the project-specific MAPort data object.
- 3 Specify which variable is read:
 - If an XIL API Framework is initialized, double-click the Variable data object. The Select Variable dialog opens.



Select the variable to be read and click OK.

- If you work with the XIL API Testbench, select the Read block in the Sequence Builder.

In the Data Object Editor, click the Reference name edit field for the Variable data object and then the Browse button to open the Data Object Selector.



In the Data Object Selector, select the Mapping - LabelMapping - <VariableAlias> - Identifier object of the variable to be read and click OK.

Alternatively, you can drag the variable from the Mapping Viewer to the block's Variable data object in the Sequence Builder.

The Variable data object is set to the model path of the variable to be read.

- 4 Set the block's Value data object as a reference to the project-specific data object that stores the read value.

Method 2

To read multiple simulator variables

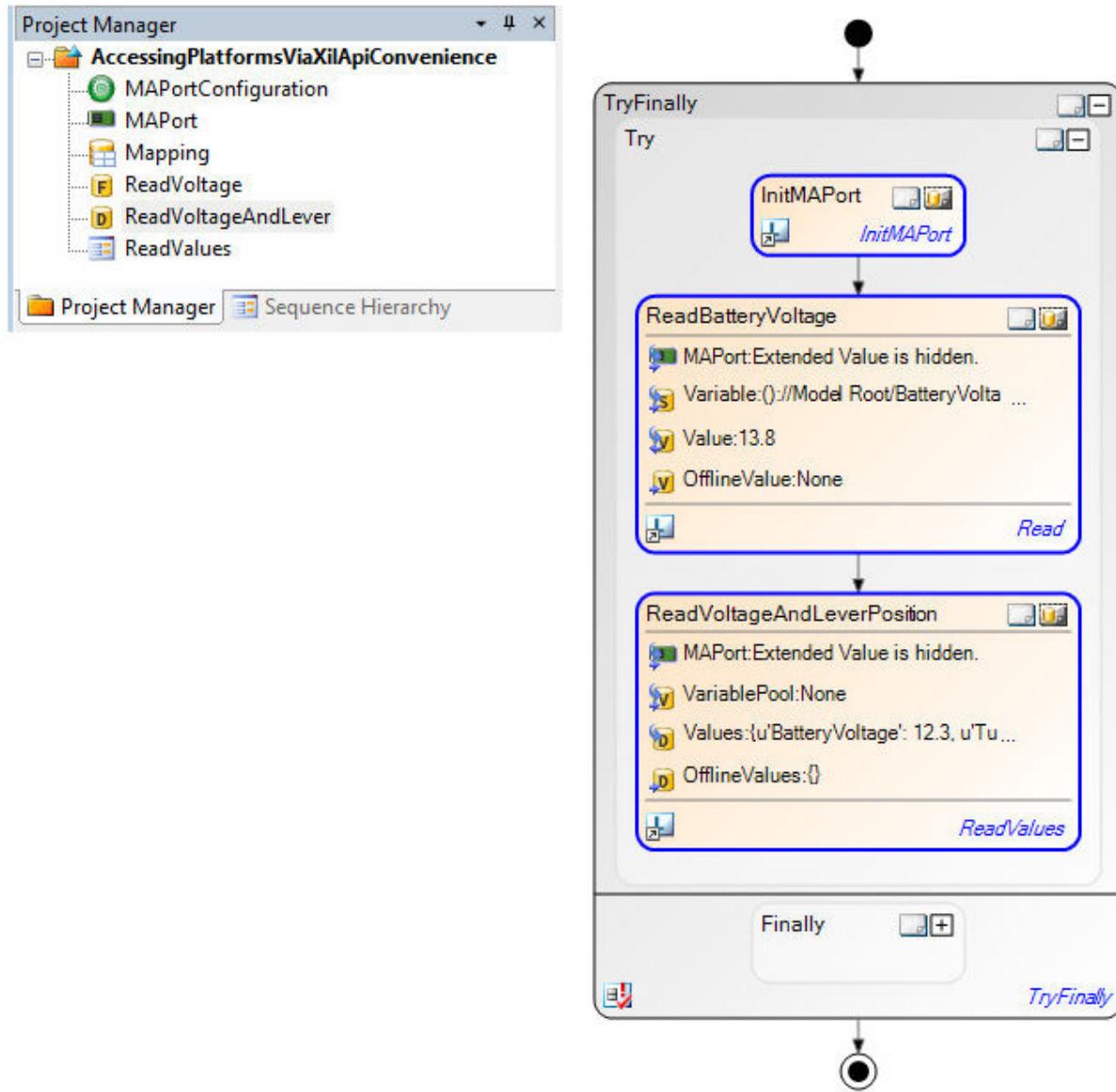
- 1 In the Project Manager, add a Dictionary data object to your project to store the read values.
- 2 In the directory's keys, specify the aliases of the variables to be read. If the dictionary is empty, all variables with mappings defined in the project's Mapping data object or in the framework's configuration are read.
- 3 From the Model Access Port folder, drag a ReadValues block to the Try path. This reads the values of the specified variables and writes them to the Dictionary data object.

By default, the block's **MAPort** data object is set as a reference to the project-specific **MAPort** data object.

- 4 If you work with the XIL API Testbench, set the block's **VariablePool** data object as a reference to the project-specific **Mapping** data object.
 - 5 Add a **Dictionary** data object to your project. By default, the dictionary is empty.
 - 6 From the **Mapping Viewer**, drag the variables to be read to the project-specific **Dictionary** data object. This specifies the related variable aliases in the directory's keys.
 - 7 Set the block's **Values** data object as a reference to the project-specific **Dictionary** data object.
-

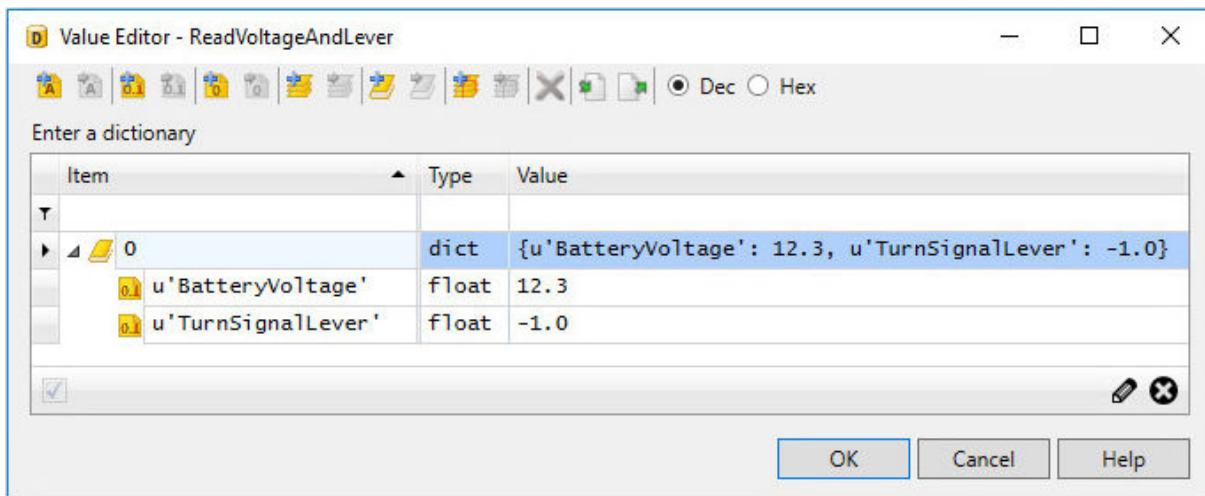
Result

You created a sequence that first reads a single value from the running simulation application and then reads multiple values. The following illustration shows the resulting sequence. If you work with an XIL API Framework, the **Mapping** data object is optional.



For example, you selected the battery voltage as the Variable data object of the Read block and specified `{'BatteryVoltage': None, 'TurnSignalLever': None}` in the project-specific dictionary object that references the Values data object of the ReadValues block.

After you executed the sequence, the project-specific dictionary contains the values of the read variables.



Next steps

You can capture the values of simulator variables over a period of time. For instructions, refer to [How to Capture Data](#) on page 96.

Related topics

References

GetValues.....	157
Read.....	164
ReadValues.....	165

How to Capture Data

Objective

You can capture the values of simulator variables [?](#) over a period of time.

Preconditions

- You created a basic [sequence](#) [?](#) for platform access and configured the access to the simulator variables. For instructions, refer to [How to Make Simulator Variables Available](#) on page 84.
- The following information is required as input data:
 - The platform-dependent name of the [Task](#) [?](#) that is to perform data capturing.
 - If you work with an [XIL API Framework](#) [?](#), you can set the MAPort data object's Target State to eSIMULATION_RUNNING or to eSIMULATION_STOPPED on the Ports page in the Mapping Editor before

you initialize the framework. This lets you later select an available task name in a dialog.

- If you work with the [XIL API Testbench](#), you can get the task names of the turn signal demo from related table in the [Results of Lesson 7 \(AutomationDesk Tutorial\)](#).
- Alternatively, you can get the taskname via the `GetTaskNames` block or method. Refer to [GetTaskNames](#) on page 293.
- The names of the variables to be captured
- The [blocks](#) to be performed during data capturing

Possible methods

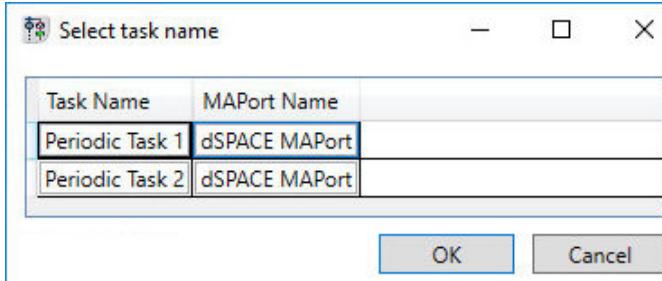
You can capture the values of simulator variables by using the following methods:

- To capture data for a fixed duration, refer to Method 1.
- To capture data controlled by a condition watcher, refer to Method 2.

Method 1

To capture data for a fixed duration

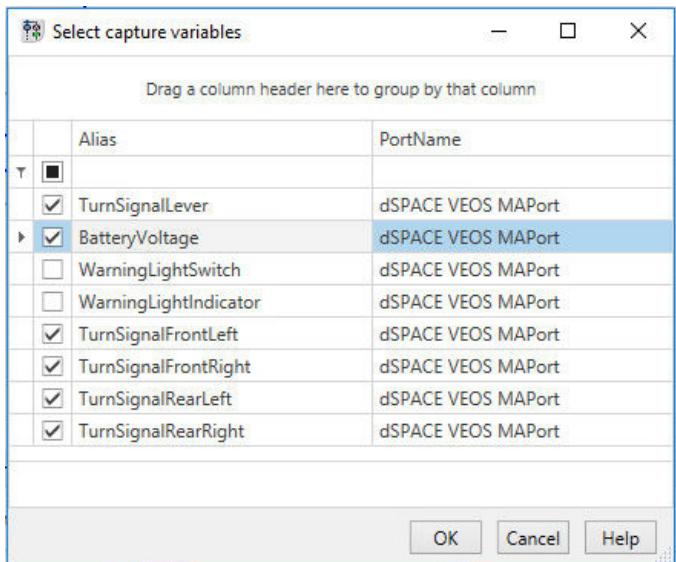
- 1 Add a Capture and a CaptureResult data object to your project to handle the instantiated capture and store its capture result.
- 2 Drag an InitializeCapture block from the **Model Access Port - Capture** folder and place it below the InitMAPort block in the basic sequence. This initializes the access to the capturing instance.
By default, the block's **MAPort** and **Capture** data objects are set as a reference to the related project-specific data object, and the **DefaultDuration** data object is set to **10** seconds. Make sure that the default duration is greater than the period of time that you want to measure.
- 3 Double-click the block's **TaskName** data object to specify the name of the task that is to perform data capturing:
 - If an XIL API Framework is initialized, the **Select Task Name** dialog opens and displays the available tasks for the specified model access ports.



Select the task name and click **OK**.

- If you work with the XIL API Testbench, the Value Editor opens. Specify the task name. For VEOS, for example, you can enter **Periodic Task 1**.
- 4 If you work with the XIL API Testbench, set the block's **VariablePool** data object as a reference to the related project-specific Mapping data object.

- 5 Double-click the block's Variables data object to specify the list of variables to be captured.
 - If an XIL API Framework is initialized, the Select Capture Variables dialog opens and displays a list of the available variable aliases for the specified model access ports.



Specify the variable list by selecting the related checkboxes and click OK.

- If you work with the XIL API Testbench, the Value Editor opens. Specify a list of the aliases for the variables to be captured.

- 6 In the block's DefaultDuration data object, specify the duration of the capture in seconds.
- 7 Drag a StartCapture block to your sequence. This starts the configured capture on the simulation platform.
By default, the block's Capture data object is set as a reference to the related project-specific data object.
- 8 Below the StartCapture block, add the blocks, which implement the actions to be performed during the capturing according to your use case.
In the example below, these actions are represented by the PerformActionsDuringMeasurement Serial block.
- 9 Drag a GetCaptureResult block folder to your sequence. This finishes the capture and writes the captured data to the CaptureResult data object.
By default, the block's Capture and CaptureResult data objects are set as a reference to the related project-specific data object.
By default, the block's StopCapturing data object is set to 1 to stop capturing after the data is fetched.
- 10 In the block's WhenFinished data object, specify 1 to let capturing last until the capture's end criterion is met.
For this capture, the end criterion is met, when the default duration is reached.

Method 2**To capture data controlled by a condition watcher**

- 1 In addition to the instructions of the previous method, drag a `ConfigureStartCondition` block from the `Model Access Port - Capture` folder and place it below the `InitializeCapture` block.

This configures the capture to start when the specified condition is fulfilled, but only within the specified timeout after the `StartCapture` block was executed. When the timeout is reached the data capture starts.

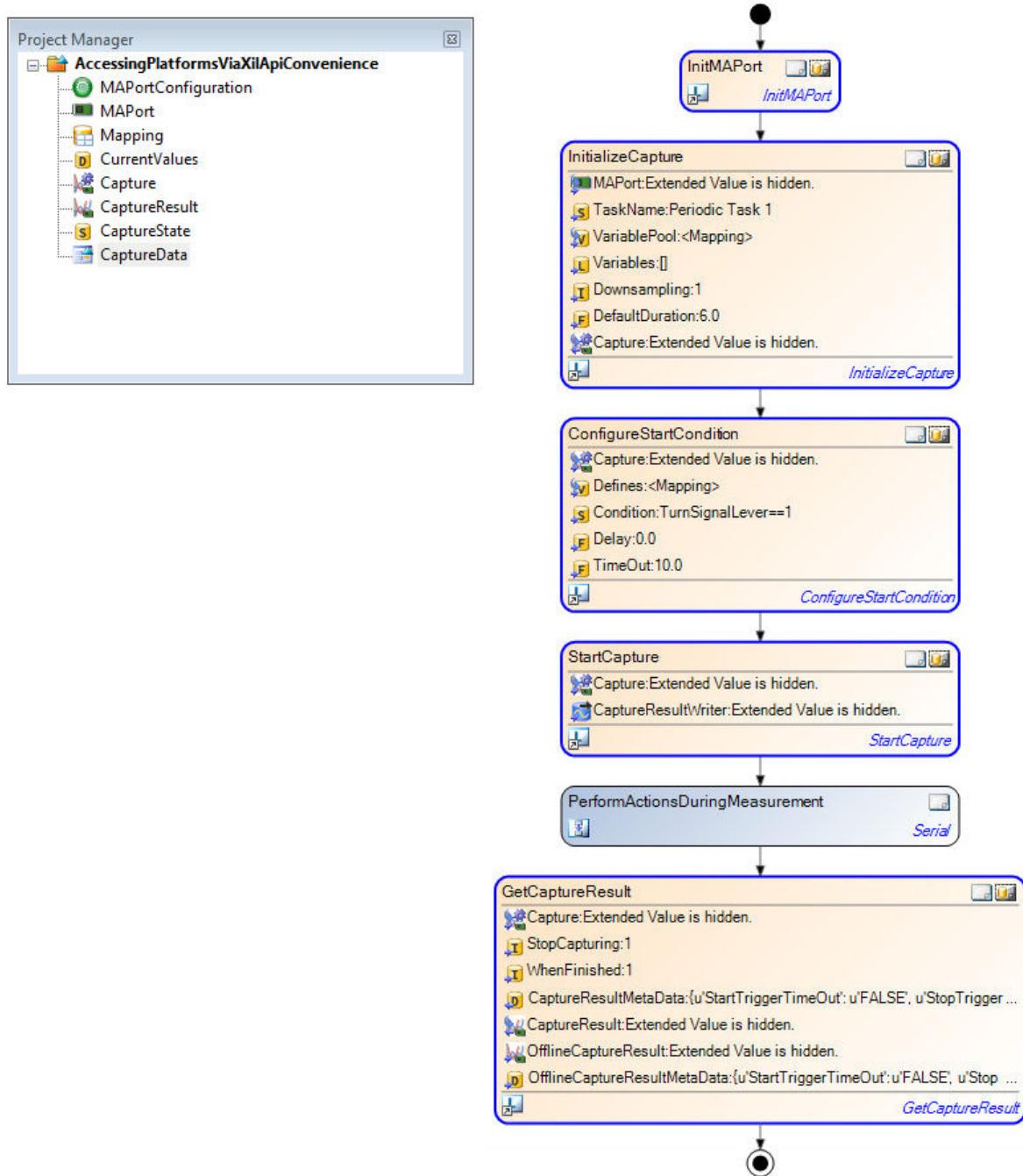
By default, the block's `Capture` data object is set as a reference to the related project-specific data object, and the `TimeOut` data object is set to **10** seconds. The timeout specifies how long to maximally wait after the execution of the `StartCapture` block for the condition to be fulfilled.
 - 2 If an XIL API Framework is initialized, the alias names are automatically available for specifying conditions.

If you work with the XIL API Testbench, set the block's `Defines` data object as a reference to the project-specific `Mapping` data object. This lets you use all the alias names that are contained in the `Mapping` data object when you specify the block's `Condition`.
 - 3 In the Data Object Editor, double-click `Condition` to open the Expression Editor and enter the start condition for capturing in the ASAM General Expression Syntax.
-

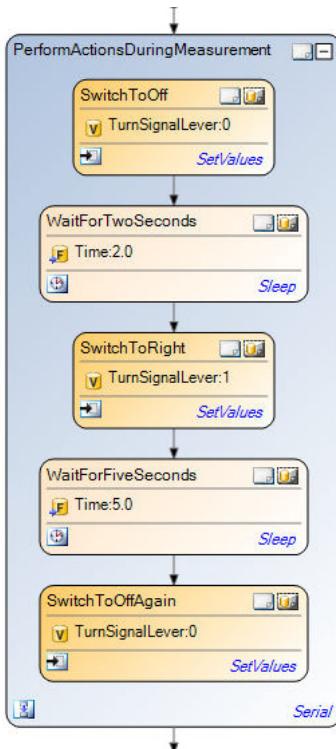
Result

You created a sequence that begins to capture data when a start condition is fulfilled and lasts for a fixed duration.

For an example, you can specify **6.0** in the `DefaultDuration` data object of the `InitializeCapture` block and `TurnSignalLever == 1` in the `Condition` data object of the `ConfigureStartCondition` block.



To implement switching of the turn-signal lever during the measurement, you can use SetValues blocks. For instructions, refer to [How to Write and Read Variables](#) on page 129.



When you run the sequence, the values of the variables contained in the variable pool are captured. Capturing starts when the turn-signal lever is switched to on and lasts for the duration of six seconds.

Tip

You can control the end of the data capture by using a condition or a duration watcher. For details, refer to [ConfigureStopCondition](#) and [ConfigureStopDuration](#).

Next steps

You can now visualize the result of the capture by adding plots to the report of your sequence. For instructions, refer to [How to Add Plots of the Captured Data to your Report](#) on page 102.

Related topics

References

Capture (Data Object).....	231
CaptureResult (Data Object).....	223
ConfigureStartCondition.....	175
ConfigureStopCondition.....	177

ConfigureStopDuration.....	179
GetCaptureResult.....	181
GetTaskNames.....	293
InitializeCapture.....	186
StartCapture.....	189

How to Add Plots of the Captured Data to your Report

Objective	You can add standardized and customized plots of the captured data to your report.
------------------	--

Preconditions	<ul style="list-style-type: none"> ▪ You created a sequence for data capturing. For instructions, refer to How to Capture Data on page 96. ▪ For a customized plot, the following information is required as input data: <ul style="list-style-type: none"> ▪ The title of the plots ▪ The names and grouping of the signals to be plotted ▪ The labels of the x-axis and y-axis for each plot
----------------------	--

Possible methods	<p>You can add plots of the captured data to the report by using the following methods:</p> <ul style="list-style-type: none"> ▪ You can add a standardized plot to the report, where every simulator variable is displayed in a separate diagram. Refer to Method 1. ▪ Alternatively, you can add a customized plot to the report, where you gather multiple simulator variables in common diagrams and specify a label for each variable . Refer to Method 2.
-------------------------	---

Method 1	To add a standardized plot of the captured data to your report
	<ol style="list-style-type: none"> 1 Drag an AddPlotsToReport block from the Model Access Port - Capture folder and place it below the GetCaptureResult block in the data capturing sequence. This adds a plot of each signal contained in the capture to the report of your sequence. By default, the block's Capture and CaptureResult data objects are set as the references to the related project-specific data objects.

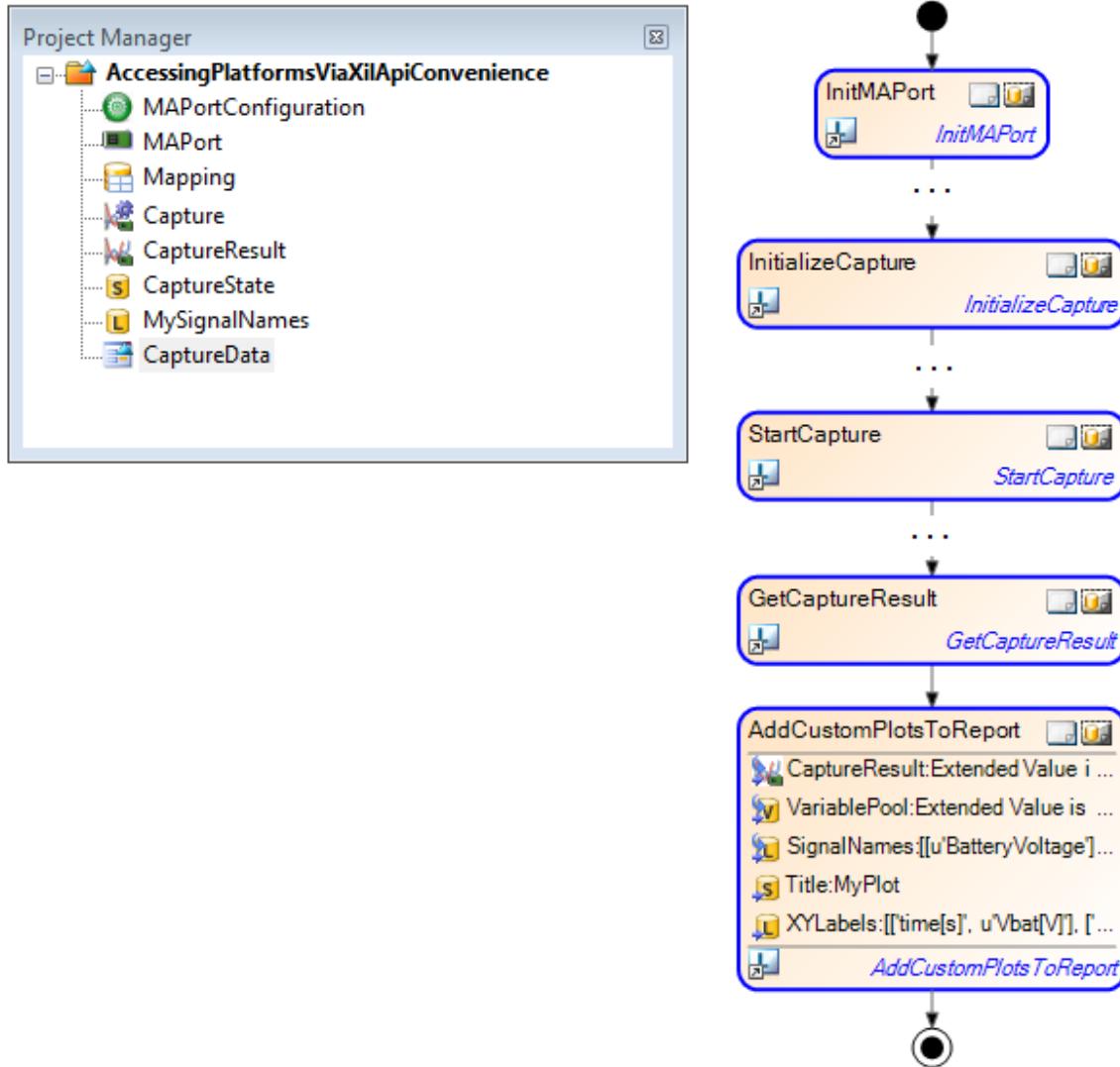
Method 2	To add a customized plot of the captured data to your report
	<ol style="list-style-type: none"> 1 Drag an AddCustomPlotsToReport block from the Model Access Port - Capture folder and place it below the GetCaptureResult block in the data capturing sequence. This adds a plot of the signals as they are specified in the SignalNames list to the report of your sequence.

By default, the block's `CaptureResult` data object is set as a reference to the related project-specific data object.

- 2 Set the block's `VariablePool` data object as a reference to the project-specific `Mapping` data object.
- 3 Add a `List` data object to your project to store which signal is to be plotted and how signals are to be grouped in common plots.
- 4 In the `List` data object, specify the aliases of the variables to be plotted. You can specify to plot more than one variable in a single diagram by gathering their aliases in a sublist.
- 5 Set the block's `SignalNames` data object as a reference to the list of the aliases.
- 6 In the block's `Title` data object, specify the string to be written as the plot's title.
- 7 In the block's `XYLabels` data object, specify the labels of the plots' axes. For each plot, specify a list of two strings, where the first string specifies the x-axis and the second the y-axis.

Result

With method 2, you enhanced your sequence with automation blocks that add plots of the captured data to the sequence's report.



For an example, drag all variables of the variable pool from the Mapping Viewer to the List data object. Via the Variable Editor, arrange the list contents to the following sub lists:

```
[['BatteryVoltage'],
 ['TurnSignalLever', 'TurnSignalLightFrontRight']]
```

In the block's XYLabels data object, specify the following list:

```
[['time[s]', 'Vbat[V]', 'time [s]', '[-1|0|+1]]']]
```

When you run the sequence, the following plots are added to the report.

Next steps	You can now release the no longer needed capturing result. For instructions, refer to How to Release Resources After Data Capturing on page 105.
-------------------	--

Related topics	References
	AddCustomPlotsToReport 172
	AddPlotsToReport 173

How to Release Resources After Data Capturing

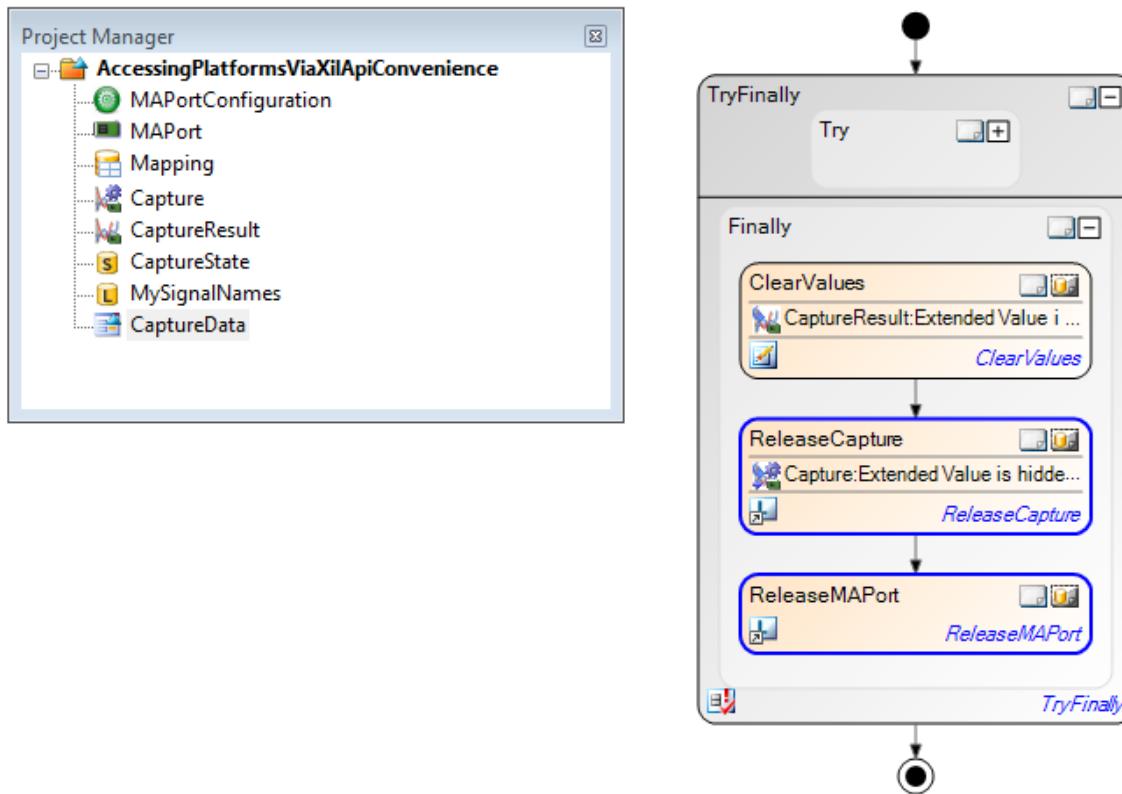
Objective	When the result of the data capture is no longer needed, you can release its allocated memory to reduce memory consumption.
------------------	---

Precondition	You created a sequence for data capturing. For instructions, refer to How to Capture Data on page 96.
---------------------	---

Method	To release resources after data capturing <ol style="list-style-type: none">1 Drag a ReleaseCapture block from the Model Access Port - Capture folder to the top of the Finally path of the sequence for data capturing. This releases the Capture data object. By default, the block's Capture data object is set as a reference to the related project-specific data object.2 Drag a ClearValues block from the Main Library to the top of the Finally path. This clears all data objects that were added to the block.3 Add a CaptureResult data object to the ClearValues block and set it as a reference to the related project-specific data object. This releases the CaptureResult data object.
---------------	---

Result

You enhanced your sequence with blocks that release no longer used resources after capturing.



If an XIL API Framework is initialized, the RealaseMAPort block is omitted from the Finally path. The MAPort data object is released when the framework is shut down.

Related topics**References**

[ClearValues \(AutomationDesk Basic Practices\)](#)

[ReleaseCapture](#)

188

How to Specify Signals for Stimulating

Objective

You can use AutomationDesk's [Signal Editor](#) to specify signals that define the progression of a variable's value over a period of time. You can store sets of

signals to an STZ file to use them later to stimulate simulator [variables](#) via the [blocks](#) of the XIL API Convenience [library](#).

Method

To specify signals for stimulating

- 1 On the Signal Editor ribbon, click View Sets - Signals to open the Signal Editor.
- 2 Click Signal Description Set - New to open a new signal set in the working area.
- 3 For each variable that you want to stimulate, drag an element from the Signal Selector's Signals category to the working area to add new signals to the signal set.
For each new signal, a row is added to the signal set.
- 4 In the working area, enter a signal name in the Signal Name edit field of each new signal.
- 5 From the Signal Selector, drag elements from the Segments category that specify the shape of a signal to its plot in the last column of the working area.
A segment of the selected type is added to the signal's plot.
- 6 Select each segment, and in the Properties pane, specify its properties.
The changes you make are shown in the signal's plot.
- 7 Click Signal Description Set - Save Signal Set As to save the signal set to an STZ file.
- 8 On the Home ribbon, click View Sets - Sequences and close the signal set in the working area to return to the Sequence Builder.

Result

You specified signals to stimulate variables and stored the signal set to an STZ file.

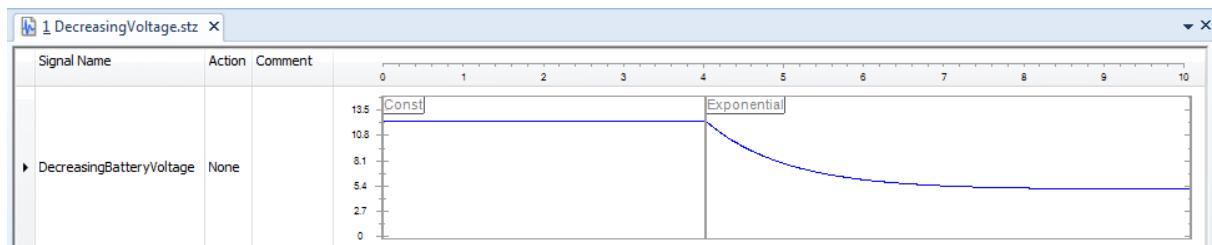
For an example, add a Segment Signal to the working area and rename the signal **DecreasingBatteryVoltage**

Add a Const segment to the signal with a duration of 4 seconds and a value of 12.3 V.

Add an Exponential segment to the signal with a Duration of 6 seconds, a Start value of 12.3 V and a Stop value of 5 V.

Save the signal set to an STZ file named **DecreasingVoltage.stz**.

The resulting signal set is shown in the following illustration.



Next steps	You can now use the signals that you specified in an STZ file to stimulate simulator variables. For instructions, refer to How to Stimulate Simulator Variables on page 108.
-------------------	--

Related topics	Basics Creating and Editing Signals (AutomationDesk Implementing Signal-Based Tests)  Using the Signal Editor (AutomationDesk Implementing Signal-Based Tests) 
	References Signal Editor (AutomationDesk Implementing Signal-Based Tests) 

How to Stimulate Simulator Variables

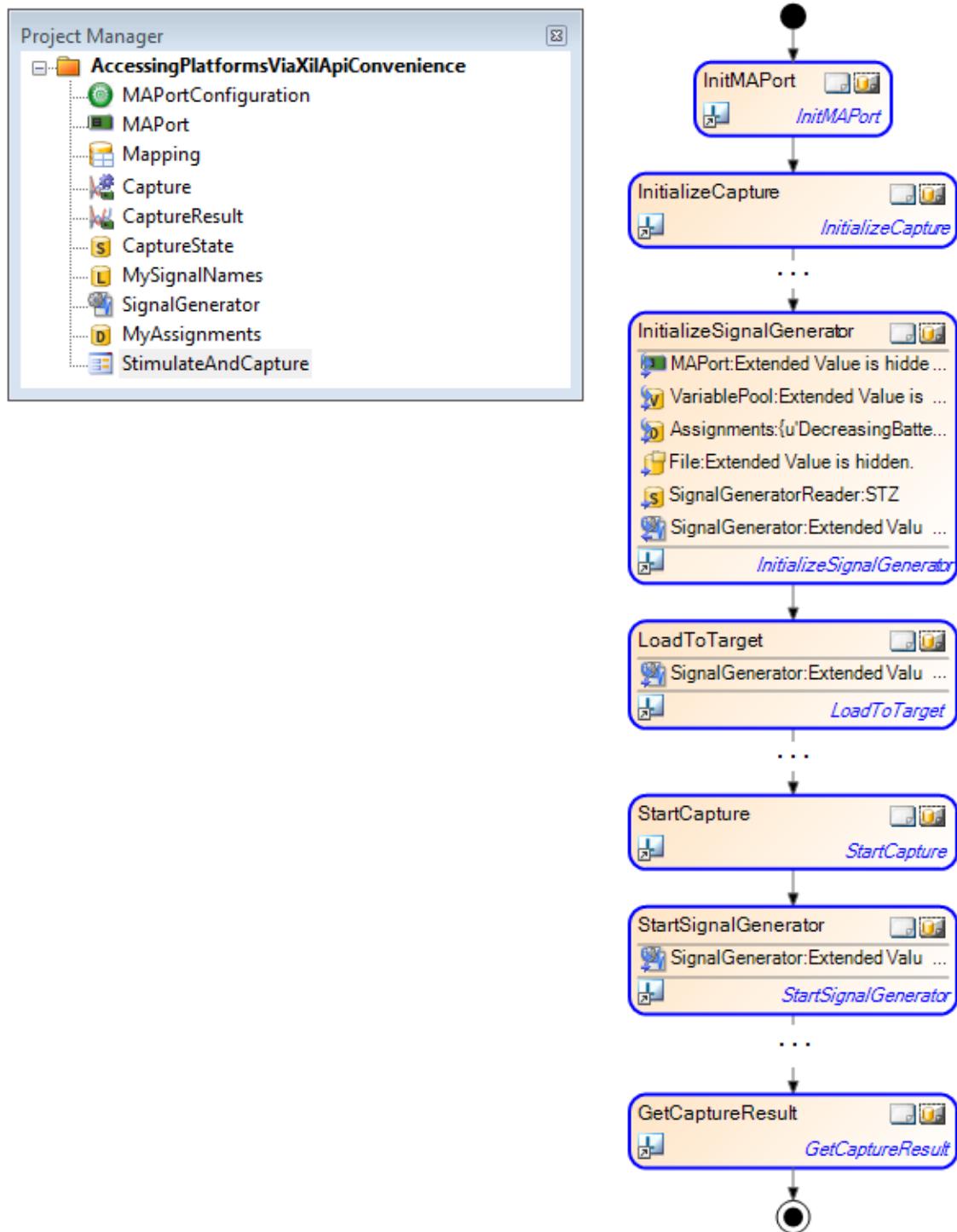
Objective	You can stimulate the value of a simulator variable over a period of time.
------------------	--

Stimulating variables on multiprocessor and multicore systems	If you stimulate variables on multiprocessor and multicore systems, you must specify the name of the member application that is to generate the signals in a custom property of the SignalGenerator data object. For more details, refer to SignalGenerator (Data Object) on page 254.
--	--

Restrictions	When stimulating simulator variables, Real-Time Testing is used. For information on preconditions and limitations on using Real-Time Testing, refer to Enabling Real-Time Testing for dSPACE Platforms (Real-Time Testing Guide)  and General Limitations for Real-Time Testing (Real-Time Testing Guide)  .
---------------------	--

Preconditions	<ul style="list-style-type: none">▪ You created a sequence for capturing data. For instructions, refer to How to Capture Data on page 96.▪ You specified the signals that stimulate the simulator variables in a signal set that is stored in an STZ file. For instructions, refer to How to Specify Signals for Stimulating on page 106.▪ The following information is required as input data:<ul style="list-style-type: none">▪ The path and name of the STZ file▪ The names of the signals to be used for stimulation and the aliases of the related variables
----------------------	---

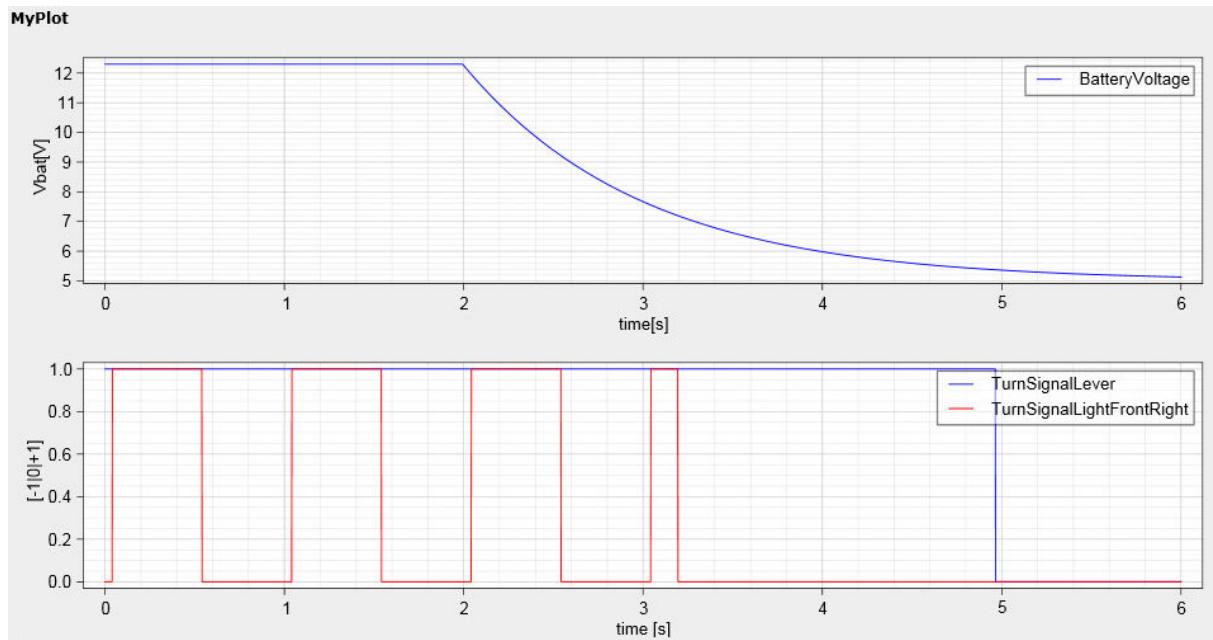
Method	To stimulate simulator variables
	<p>1 Add a SignalGenerator data object to your project to parameterize signal generating.</p> <p>2 Add a Dictionary data object to your project to store the assignments of signal names to variable aliases.</p> <p>In the example below, the Dictionary data object is renamed to MyAssignments.</p> <p>3 In the Dictionary data object, use key-value pairs to specify which signal is to stimulate which variable. In each key, specify the signal name and in the value the variable alias.</p> <p>4 Drag an InitializeSignalGenerator block from the Model Access Port - SignalGenerator folder and place it below the blocks that initialize the data capture. This initializes the access to the signal-generating instance.</p> <p>By default, the block's MAPort and SignalGenerator data objects are set as the references to the related project-specific data objects.</p> <p>5 Set the block's VariablePool data object as a reference to the project-specific VariablePool Mapping data object.</p> <p>6 Set the block's Assignments data object as a reference to the project-specific Dictionary data object that assigns signal names to variable aliases.</p> <p>7 In the block's File data object, specify the path and file name of the STZ file that stores the signal set which stimulates the variables.</p> <p>8 Drag a LoadToTarget block and place it below the InitializeSignalGenerator block. This downloads the configured signal generator to the platform.</p> <p>By default, the block's SignalGenerator data objects is set as the reference to the related project-specific data object.</p> <p>9 Drag a StartSignalGenerator block and place it below the StartCapture block. This starts the variable stimulation on the platform.</p> <p>By default, the block's SignalGenerator data object is set as the reference to the related project-specific data object.</p>
Result	You created a sequence that stimulates simulator variables with signals specified in an STZ file.



For an example, add the following item to the Directory data object that specifies the assignments of signal names to variable aliases:
`{ 'DecreasingBatteryVoltage': 'BatteryVoltage' }.`

In the InitializeSignalGenerator block's File data object, specify the name and path of the file that contains the signal description of the DecreasingBatteryVoltage signal.

When you run the sequence, the battery voltage is stimulated during data capturing. The following plots are added to the report.



Next steps

You can now stop generating signal and release no longer needed resources. Refer to [How to Release Resources after Variable Stimulation](#) on page 112.

Related topics

Basics

[Using the Value Editor \(AutomationDesk Basic Practices\)](#)

References

InitializeSignalGenerator.....	197
LoadToTarget.....	200
SignalGenerator (Data Object).....	254
StartSignalGenerator.....	202

How to Release Resources after Variable Stimulation

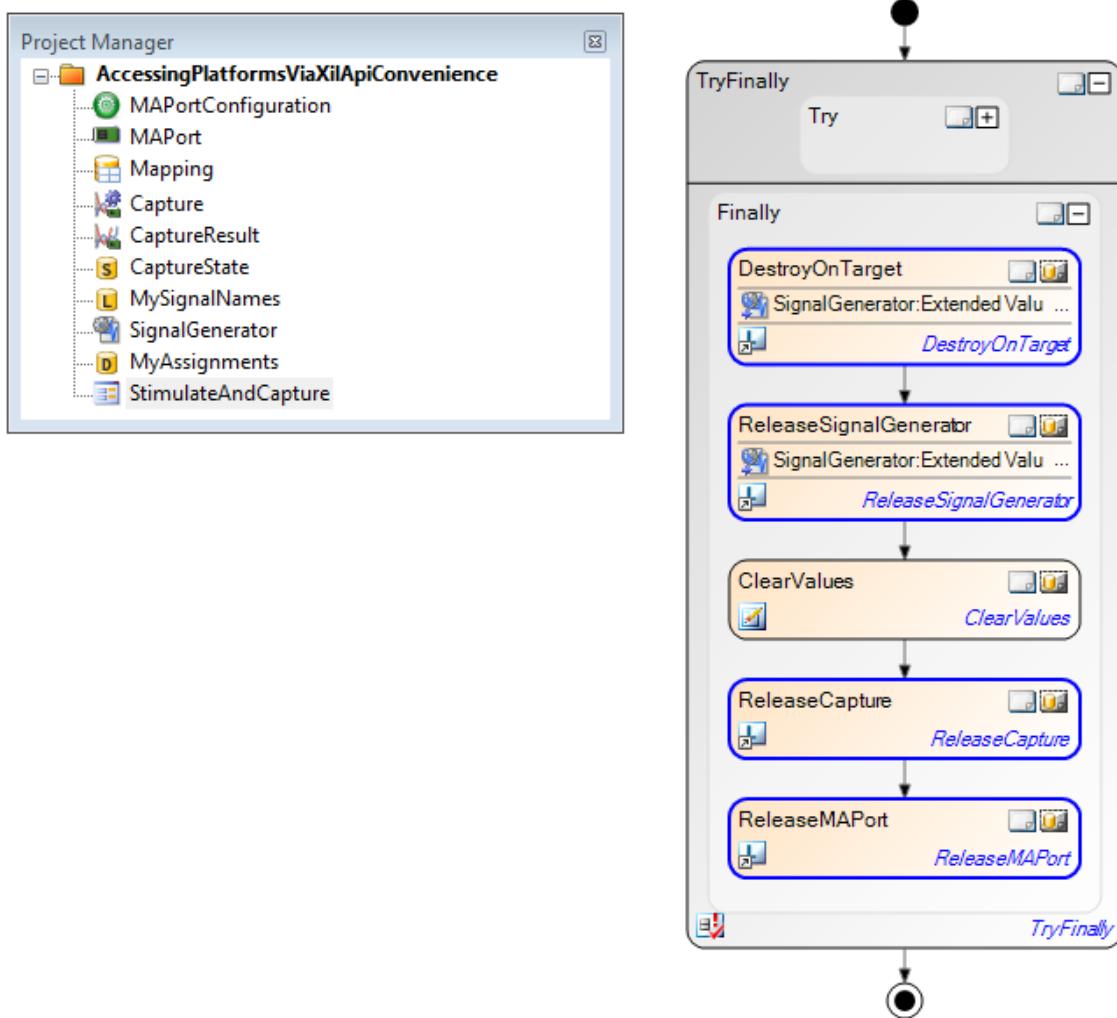
Objective When variable stimulation is no longer needed, you can stop signal generation and release its allocated memory to reduce memory consumption.

Precondition You created a sequence that stimulates and captures simulator variables. For instructions, refer to [How to Stimulate Simulator Variables](#) on page 108.

Method **To release resources after variable stimulation**

- 1** Drag a DestroyOnTarget block from the Model Access Port - SignalGenerator folder to the top of the Finally path of the stimulating and capturing sequence. This stops the signal generation on the platform. By default, the block's SignalGenerator data objects is set as the reference to the related project-specific data object.
- 2** Drag a ReleaseSignalGenerator block and place it below the DestroyOnTarget block. This releases the SignalGenerator data object. By default, the block's SignalGenerator data object is set as the reference to the related project-specific data object.

Result You enhanced your sequence with blocks that stop signal generation and release no longer needed resources after variable stimulation.



If an XIL API Framework is initialized, the RealaseMAPort block is omitted from the Finally path. The MAPort data object is released when the framework is shut down.

You have now completed a sequence that implements all steps to stimulate and capture simulator variables

Related topics

References

DestroyOnTarget.....	196
ReleaseSignalGenerator.....	203

How to Access Platforms via a Third-Party XIL API Server

Objective

You can automate the access to simulator platforms via a third-party XIL API server by using the XIL API Convenience library.

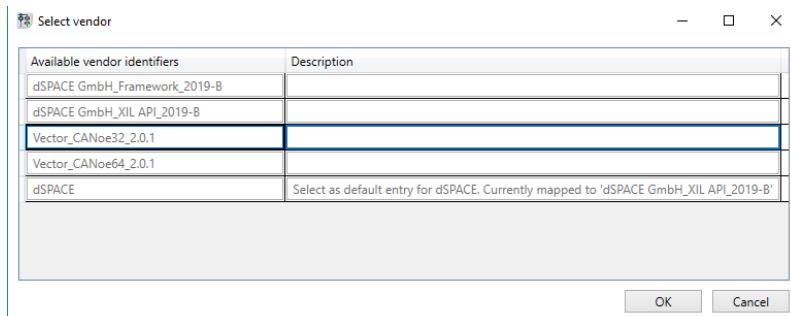
Preconditions

- The third-party XIL API server must be installed and accessible. Refer to the documentation that is provided by the vendor.
- The following information is required as input data:
 - The path and name of the required vendor-specific model access port configuration file
 - If you work with an XIL API framework:
 - The framework must be configured and initialized. Refer to [How to Work with a Third-Party XIL API Framework](#) on page 71.

Method

To access platforms via a third-party XIL API server

- 1 From the Library Browser, drag a Vendor data object from the XIL API Convenience library's Model Access Port - Data Objects folder to the top level of your project. This adds a data object to your project to specify the desired vendor of the XIL API implementation.
- 2 In the Project Manager, double-click the Vendor data object. The Select Vendor dialog opens and lets you specify the vendor by selection.



Note

If the third-party XIL API implementation is not provided in the list of available vendors, the XIL API server is not properly installed or accessible. Refer to the vendor's documentation.

Select the vendor identifier and click OK to close the dialog.

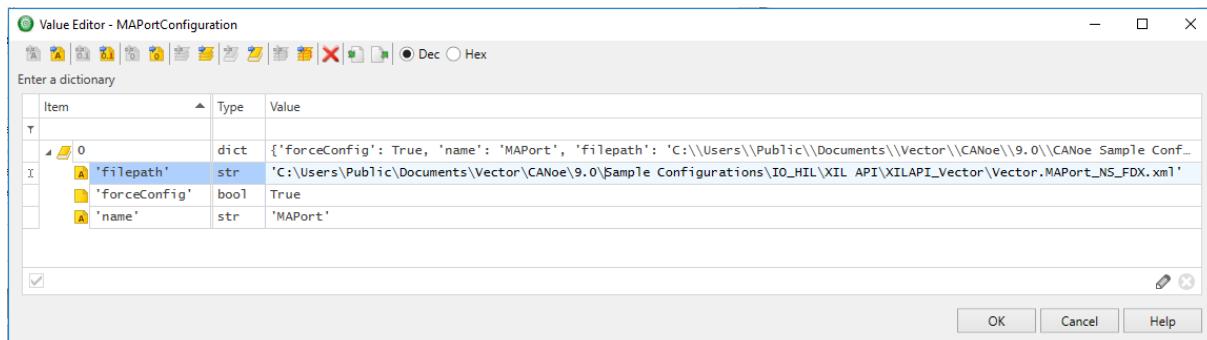
Tip

In all subsequent worksteps, use the project-specific Vendor data object as the reference for the Vendor data objects of all XIL API Convenience automation blocks.

- 3** On the Home ribbon, click Insert – Data Objects – MAPortConfiguration. This adds a data object to your project that contains the specification of the platform and the simulation application to be accessed.

- 4** On the MAPortConfiguration data object's context menu, select **Edit as Dictionary**.

The MAPortConfiguration data object is opened in the Value Editor and lets you specify the vendor-specific model access configuration, i.e. `name`, `filepath`, and `forceConfig`.



- 5** In the Project Manager, select your project. On the Home ribbon, click Insert – Data Objects – Mapping. This adds a data object to the top level of your project that contains the variable mapping.

- 6** Click the Mapping data object to open it in the Mapping Viewer.

- 7** In the Mapping Viewer, enter alias names for the variables that you want to access in the Alias column, and their model path in the Identifier column.

Drag a column header here to group by that column			
Alias	Identifier	VariableType	Description
RW_StateSwitch	FDX/EngineStateSwitch		
RW_EngineSpeed	FDX/EngineSpeedEntry		
RW_FlashLight	FlashLight		
RW_HeadLightSwitch	FDX/HeadLightSwitch		

- 8** Build a sequence to access simulator variables. Refer to [How to Build a Basic Sequence for Accessing Simulator Variables](#) on page 78.

Result

You created a basic sequence for automating the access to simulator platforms via third-party XIL API servers and you made simulator variables available.

Next steps

You can proceed, with the following steps:

- Write simulator variables. Refer to [How to Write Variable Values](#) on page 87.
- Read simulator variables. Refer to [How to Read Variable Values](#) on page 91.
- Capture data. Refer to [How to Capture Data](#) on page 96.
- Using signals to stimulate variables. Refer to [How to Specify Signals for Stimulating](#) on page 106.

Related topics

References

MAPort (Data Object).....	286
MAPortConfiguration (Data Object).....	289
Mapping (Data Object).....	209

Accessing Simulation Platforms via XIL API

Where to go from here

Information in this section

Basics of the XIL API Library Elements.....	118
Provides basic information on automating access to XIL API.	
Overview of the XIL API Library Elements.....	119
Provides an overview of AutomationDesk's XIL API library elements to access the XIL API.	
Example of an XIL API Sequence.....	121
Provides an example of automating access to connected hardware via XIL API.	
How to Prepare a Project for Using the XIL API Library.....	123
Instructions on preparing a sequence its data objects according to the XIL API Library example.	
How to Initialize a Model Access Port.....	126
Instructions on initializing a model access port (MAPort) for managing access to the simulation model.	
How to Make Simulator Variables Available to XIL API Library Automation Blocks.....	127
Instruction how to specify simulator variables as project-specific data objects via creating Variable data objects in the Project Manager.	
How to Write and Read Variables.....	129
Instructions on reading and writing variables of the HIL system.	
How to Get Application Properties.....	133
Instructions on getting simulation application properties of the HIL system.	
How to Capture Data.....	134
Instructions on capturing data from a HIL system.	
Basics on Stimulating Variables via AutomationDesk.....	137
Provides information on the XIL API library elements for stimulus handling via STZ file format.	
How to Migrate a Variable Pool's Data Container to an XIL API Mapping Data Object.....	138
Instruction how to migrate the a variable pool to an XIL API Mapping data object.	

Basics of the XIL API Library Elements

Introduction

The ASAM (Association for Standardisation of Automation and Measuring Systems) released the ASAM XIL API standard, defining an interface to connect test automation tools with any HIL (hardware-in-the-loop) system.

The ASAM XIL API consists of so-called packages and subpackages which contain a set of operations that are used to define measurement configurations, control the execution of measurements, obtain the measured data as results, etc.

For more information on the ASAM XIL API standard, refer to:

- [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#)
- [ASAM_AE_XIL_Generic-Simulator-Interface_BS-2-4_CSharp-API-Technology-Reference-Mapping-Rules_V2-1-0.pdf](#)

The AutomationDesk XIL API [library](#) supports parts of version 2.1.0 of ASAM XIL API and contains folders and subfolders to adapt its structure. It is now easier to use test automation with any HIL simulator. This increases test reuse, helping to protect investments and reduce development costs and time.

The supported features of the XIL API standard are:

- Testbench
- MAPort for model access
- EESPort for electrical error simulation

Compatibility

The XIL API 2.1.0 is the successor of the XIL API 2.0.1. Projects that use the XIL API 2.0.1 can also be used with the current XIL API.

XIL API Convenience library

AutomationDesk provides the XIL API Convenience library, which offers [automation blocks](#) based on the [data objects](#) and automation blocks in the XIL API library. It is implemented as a write-protected [custom library](#). XIL API blocks that are required to perform a certain action are combined in a single automation block in the XIL API Convenience library. It also facilitates migration from [platform](#) access via the XIL API library.

For further information, refer to [Accessing Simulation Platforms via the XIL API Convenience Library](#) on page 75.

Tip

It is recommended to use the automation blocks in the XIL API Convenience library. They facilitate the creation of XIL API-based tests, especially when you want to use it for electrical error simulation.

This document does not provide instructions for the electrical error simulation (EESPort). Look at the `ElectricalErrorSimulationPort` demo project in `<DocumentsFolder>\XIL API Convenience`

Related topics**Basics**

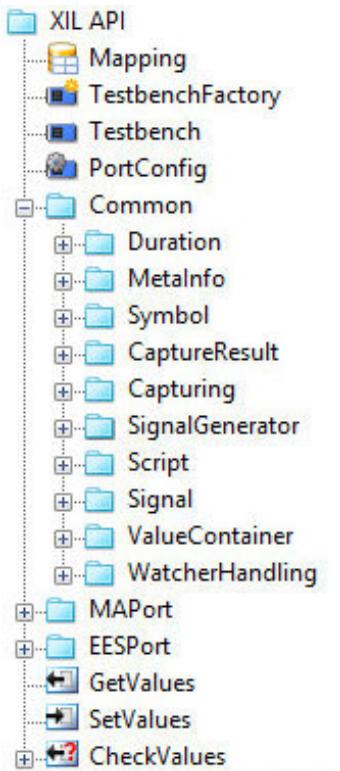
Packaging of AutomationDesk (AutomationDesk Introduction And Overview 

Overview of the XIL API Library Elements

Introduction

The AutomationDesk XIL API [library](#) provides some of the functionality of the ASAM Application Programming Interface for ECU Testing via Hardware-in-the-Loop Simulation (ASAM XIL API) to AutomationDesk.

There are [folders](#) and subfolders in the AutomationDesk XIL API library which reflect some packages and subpackages of the ASAM XIL API. The part of the ASAM XIL API that is currently implemented in the AutomationDesk XIL API library is shown in the illustration below:



Common

The Common folder reflects the Common package of the ASAM XIL API.

Duration The Duration folder provides [data objects](#) for handling durations that differ in unit and data type.

MetaInfo The MetaInfo folder provides data objects which you can use to acquire information on tasks and variables.

Symbol The Symbol folder provides the Symbol data object which you can use to obtain values by symbol.

CaptureResult The CaptureResult folder provides data objects and automation blocks[?] which you can use to control the execution of capturing and to obtain the measured data as results. For further information, refer to [CaptureResult](#) on page 222.

Capturing The Capturing folder provides data objects and automation blocks which you can use to acquire data in a continuous data stream. For further information, refer to [Capturing](#) on page 230.

SignalGenerator The SignalGenerator folder provides data objects and automation blocks which you can use to obtain a SignalDescriptionSet for stimulating variables in a real-time application.

Script The Script folder provides data objects to handle scripts that are executed on real-time hardware or VEOS synchronous to the real-time applications.

Signal The Signal folder provides the SignalDescriptionSet data object which you can use to obtain a set of signals for signal evaluation or stimulating variables in a real-time application.

ValueContainer The ValueContainer folder provides data objects and automation blocks which you can use to obtain and provide data. For further information, refer to [ValueContainer](#) on page 273.

WatcherHandling The WatcherHandling folder provides data objects and automation blocks which you can use, for example, to define the trigger definitions of captures. For further information, refer to [WatcherHandling](#) on page 279.

MAPort

The MAPort folder reflects the MAPort package of the ASAM XIL API and provides data objects and automation blocks in conjunction with the Model Access port (MAPort). For further information, refer to [MAPort](#) on page 285.

Tip

For a convenient way of implementing XIL API-based tests for model access, you can use the automation blocks in the XIL API Convenience library, refer to [XIL API Convenience \(Model Access\)](#) on page 160.

EESPort

The EESPort folder reflects the EESPort package of the ASAM XIL API and provides data objects which you can use for [electrical error simulation \(EES\)](#)[?]. For further information, refer to [XIL API \(Electrical Error Simulation\)](#) ([AutomationDesk Simulating Electrical Errors](#) ).

Tip

For a convenient way of implementing XIL API-based tests for electrical error simulation, you can use the automation blocks in the XIL API Convenience library, refer to [XIL API Convenience \(Electrical Error Simulation\)](#) ([AutomationDesk Simulating Electrical Errors](#)).

Related topics**Basics**

[Packaging of AutomationDesk \(AutomationDesk Introduction And Overview\)](#)

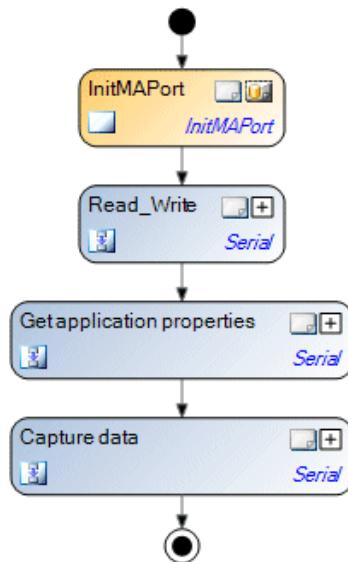
Example of an XIL API Sequence

Introduction

The following example contains the essential [automation blocks](#) and [data objects](#) to automate access to connected hardware via the [XIL API](#).

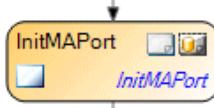
AutomationDesk sequence

The following [automation sequence](#) shows you a simple program to access an HIL system via the XIL API.



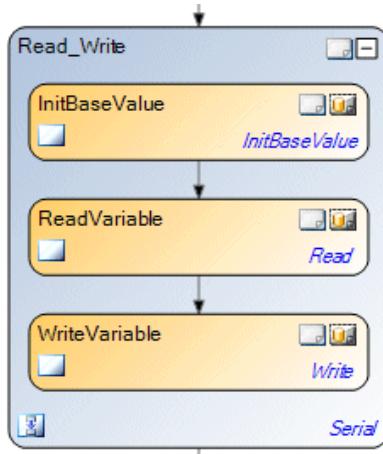
It is split into four parts for clarity:

- *InitMAPort*



To initialize the [Model access port \(MAPort\)](#) data object (see [How to Initialize a Model Access Port](#) on page 126).

- *Read_Write*



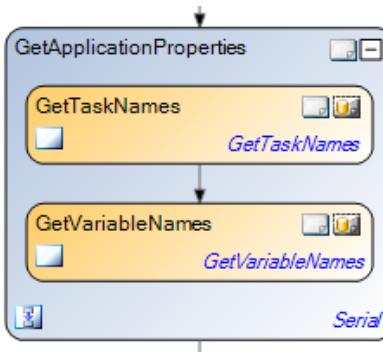
- To read and write the values of a [variable](#).

- To return the value of a variable.

- To set the value of a variable.

(see [How to Write and Read Variables](#) on page 129)

- *Get application properties*



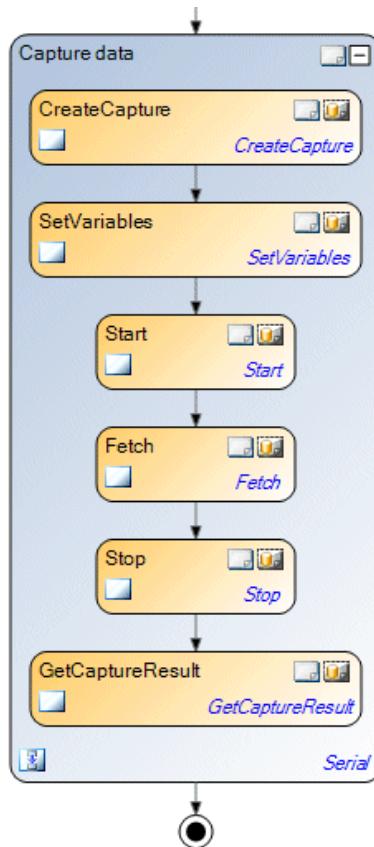
- To return the names of all available [tasks](#).

- To set the variables for capturing.

- To return the variables which are selected for capturing.

(see [How to Get Application Properties](#) on page 133)

- Capture data



- To create a Capture object with the given task.
 - To set the variables for capturing.
 - To start data logging.
 - To catch the measurement results.
 - To stop data logging.
 - To catch the measurement results.
- (see [How to Capture Data](#) on page 134)

Demo projects

Further AutomationDesk demo projects can be found at
 <DocumentsFolder>\XIL API.

How to Prepare a Project for Using the XIL API Library

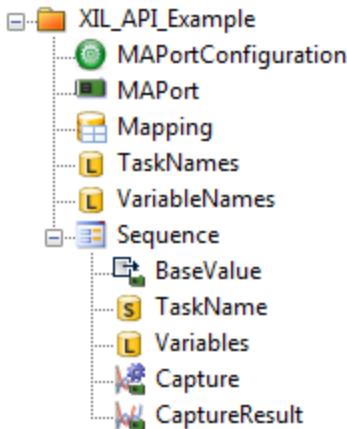
Objective

This [project](#) requires some [data objects](#) which you have to add.

Precondition	<ul style="list-style-type: none">▪ You have added an AutomationDesk sequence to your project.
Method	<p>To prepare a project for using the XIL API library</p> <ol style="list-style-type: none">1 Drag an MAPortConfiguration data object from the Library Browser (MAPort folder) to your project to configure the access to the simulation application.2 Drag an MAPort data object from the Library Browser (MAPort folder) to your project in the Project Manager to access the model that is simulated on the HIL simulator.3 If you work with the XIL API Testbench, drag a Mapping data object from the Library Browser to your project to configure the variable access of the simulated model.4 Drag a List data object from the Library Browser to your project and rename it TaskNames. You need this to hold all available task names of the simulation application.5 Drag a List data object from the Library Browser to your project and rename it VariableNames. You need this to hold all available variable names of the simulation application.6 Drag a BaseValue data object from the Library Browser (Common - ValueContainer folder) to the sequence in the Project Manager to read and write variables.7 Drag a String data object from the Library Browser to the sequence in the Project Manager and rename it TaskName. You need this to specify the name of the task to perform the capturing.8 Drag a List data object from the Library Browser to the sequence in the Project Manager and rename it Variables. You need this to specify the names of the variables to be captured.9 Drag a Capture data object from the Library Browser (Common - Capturing folder) to the sequence in the Project Manager. You need this for getting the application properties.10 Drag a CaptureResult data object from the Library Browser (Common - CaptureResult folder) to the sequence in the Project Manager. This gets the result of a HIL measurement.

Result

You have added data objects to your project to work with in the next steps.



If you work with the XIL API Framework, there is no Mapping data object.

Tip

By default, you can use the project with the dSPACE implementation of the ASAM XIL API standard for the MAPort. If you want to use your project with an XIL API implementation of another vendor, you can use the XIL API Convenience library. Refer to [How to Access Platforms via a Third-Party XIL API Server](#) on page 114.

Next steps

You can now proceed with initializing the MAPort. Refer to [How to Initialize a Model Access Port](#) on page 126.

Related topics**Basics**

[Notes on Using Tuple, List, And Dictionary Data Objects \(AutomationDesk Basic Practices\)](#)

References

BaseValue (Data Object).....	274
Capture (Data Object).....	231
CaptureResult (Data Object).....	223
MAPort (Data Object).....	286

How to Initialize a Model Access Port

Objective

The [model access port \(MAPort\)](#) is the central point for managing access to the model simulated on the HIL simulator. This port provides functionality such as read/write access to the model. You must initialize the MAPort for your AutomationDesk [sequence](#).

Precondition

- You added an AutomationDesk sequence to your project and you prepared its data objects (see [How to Prepare a Project for Using the XIL API Library](#) on page 123).
- The associated [variable description file](#) exists.

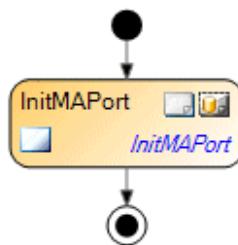
Method

To initialize a model access port (MAPort)

- 1 In the Project Manager, double-click the MAPortConfiguration data object to open it in the Variables pane to configure model access.
- 2 In the Variables pane, enter the platform of your HIL simulator in the Platform name input field.
- 3 Enter the variable description file (SDF) of the simulation application in the Variable file edit field and choose whether the specified path is absolute or relative to the project directory.
- 4 Drag an InitMAPort block from the Library Browser (MAPort folder) to the Sequence Builder. This block initializes the MAPort for your AutomationDesk sequence.
- 5 Set the block's ConfigurationDict data object as a reference to the project-specific MAPortConfiguration data object.
- 6 Set the block's MAPort data object as a reference to the project-specific MAPort data object.

Result

When you execute the sequence, the MAPort is initialized for your AutomationDesk sequence.



Tip

The MAPortConfiguration data object is an alternative to the ConfigurationDict dictionary required by the InitMAPort block. It offers access to AutomationDesk's Variables pane that allows you to specify a variable description file with its variables. For information, refer to [MAPortConfiguration \(Data Object\)](#) on page 289.

Next steps

You can now proceed with reading and writing your variables. Refer to [How to Make Simulator Variables Available to XIL API Library Automation Blocks](#) on page 127.

Related topics**Basics**

[Notes on Using Tuple, List, And Dictionary Data Objects \(AutomationDesk Basic Practices\)](#)

References

BaseValue (Data Object).....	274
Capture (Data Object).....	231
CaptureResult (Data Object).....	223
InitMAPort.....	295
MAPort (Data Object).....	286

How to Make Simulator Variables Available to XIL API Library Automation Blocks

Objective

To change the operating point of a simulation, you have to modify simulator [variables](#). The variables you want to access have to be specified in AutomationDesk. You can use all kind of variables in AutomationDesk:

- Parameter
- System variable
- Block output
- Block input
- Label

Re-usability of XIL API sequences

To make your XIL API [sequence](#) reusable for similar automation tasks, you should use project-specific [data objects](#) containing the simulator variables and reference them to the Variable data objects of the read and write [blocks](#).

Restrictions	<p>Variable names may not contain:</p> <ul style="list-style-type: none">▪ Quotes▪ ESC characters (\n, \t, etc.) <p>You can usually use the corresponding key instead. For example, use the Enter key instead of \n.</p>
Preconditions	<ul style="list-style-type: none">▪ You added an AutomationDesk sequence to your project and you prepared its data objects (see How to Prepare a Project for Using the XIL API Library on page 123).▪ You configured your model access port (see How to Initialize a Model Access Port on page 126).▪ The associated variable description file exists.
Method 1	<p>To make simulator variables available to automation blocks via the XIL API Framework</p> <ol style="list-style-type: none">1 Create a new XIL API Framework configuration file as described in How to Create a Framework Configuration on page 59.2 In the framework configuration, specify the variable mapping as described in How to Specify the Variable Mapping of a Framework Configuration on page 66.3 Use the created framework configuration to initialize the XIL API Framework as described in How to Initialize an XIL API Framework on page 68.
Method 2	<p>To make simulator variables available to automation blocks via the Mapping data object</p> <ol style="list-style-type: none">1 In the Project Manager, double-click the Mapping data object to open the Mapping Viewer.2 In the context menu of the Mapping Viewer, choose Edit to open the Mapping Editor.3 In the Mapping Editor, browse through the variable tree at the left side and drag the desired variable to the mapping table at the right side. A new entry is added to the mapping table which maps a default alias to the variable's model path in the Identifier column.
Method 3	<p>To make simulator variables available to automation blocks via String data objects</p> <ol style="list-style-type: none">1 In the Project Manager, choose Edit from the MAPortConfiguration context menu to open the Variables pane.2 Click the Show button to display the contents of the specified variable description file.

- 3 Browse through the variable tree and drag the desired variable to the project or folder element in the Project Manager. A String data object that contains the model path of the variable is added.

Note

If you drag a variable to an existing variable data object, the variable's name and value are modified.

Result

The project contains a parameterized data object that can be referenced by the XIL API access automation blocks.

Next step

When you have specified the variables to be accessed in your automation project, you should proceed with [How to Write and Read Variables](#) on page 129.

Related topics**References**

Edit (MAPortConfiguration).....	358
---------------------------------	-----

How to Write and Read Variables

Objective

You have created an instance of the [model access port](#), the HIL simulator has been initialized and a simulation model is running. Now you can request all available [model variables](#) in the simulation.

Possible methods

You can access simulator variables by using the following methods:

- You can read and write multiple simulator variables by specifying them as Variant data objects. This is the recommended method to access simulator variables. Refer to Method 1.
- You can access a single simulator variable by specifying its model path in a string. Refer to Method 2.

Precondition

You have initialized the MAPort (see [How to Initialize a Model Access Port](#) on page 126) and all preconditions of the previous steps.

Method 1**To write and read multiple variables**

- 1 On the Home ribbon, click Insert - Insert SetValues.
A SetValues block is added to the sequence.
- 2 In the Variable Viewer, select the variables to be written and drag them to the SetValues block in the Sequence Builder.
Variant data objects whose names are the same as the aliases of the selected variables are added to the block.
- 3 For each Variant data object of the SetValues block: Double-click the data object to open the Value Editor and specify the value to be written.
- 4 If you work with the XIL API Testbench, select Disable Framework Support from the context menu of the SetValues block.
An MAPort data object and a Mapping data object are added to the block. They both automatically reference data objects at a higher hierarchy level.
- 5 On the Home ribbon, click Insert - Insert GetValues. A GetValues block is added to the sequence.
- 6 In the Variable Viewer, select the variables to be read and drag them to the GetValues block in the Sequence Builder.
Variant data objects whose names are the same as the aliases of the selected variables are added to the block.
- 7 If you work with the XIL API Testbench, select Disable Framework Support from the context menu of the GetValues block.
An MAPort data object and a Mapping data object are added to the block. They both automatically reference data objects at a higher hierarchy level.

Method 2**To write and read a single variable**

- 1 Drag a Serial block from the Library Browser to the Sequence Builder.
- 2 Add a data object to the serial that stores the value to be written or read.
- 3 Drag an InitBaseValue block from the Library Browser (Common - ValueContainer folder) to the sequence to initialize a base value data object.
- 4 In the block's DataType data object, select the type of variable to be written or read from the list of supported types.
- 5 Set the block's Value data object as a reference to the data object to be written or read.
- 6 Set the block's BaseValue data object as a reference to the project-specific data object with the same name.
- 7 Drag a Write block from the Library Browser (MAPort folder) to the sequence.
By default, the block's MAPort data object is set as a reference to the project-specific data object.
- 8 In the VariableName data object, specify the variable's model access path.

You can do this in the following ways:

- In the Data Object Selector, select the variable's Identifier element in the variable tree of the project-specific Mapping data object.
- Drag the variable's mapping from the Mapping Viewer to the block's VariableName data object.
- Specify the model path as a string by using the Value Editor. For example, to specify the model path of the battery voltage in the AutomationDesk turn signal demo for VEOS, enter:

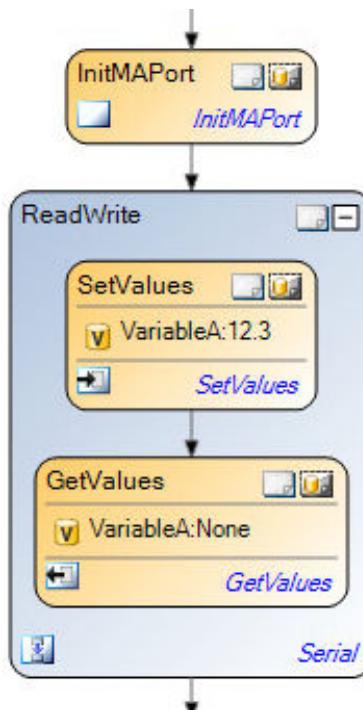
```
(()://Model Root/BatteryVoltage[V]/Value
```

- 9 Set the block's Value data object as a reference to the sequence-specific BaseValue data object.
- 10 Drag a Read block from the Library Browser (MAPort folder) to the sequence to get the value of a variable.
By default, the block's MAPort data object is set as a reference to the project-specific data object.
- 11 Set the VariableName data object as a reference to the variable's model access path.
- 12 Set the block's Value data object as a reference to the sequence-specific BaseValue data object.

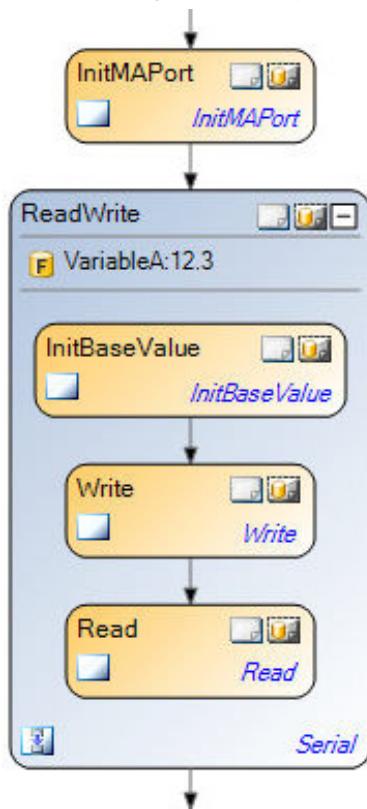
Result

When you execute the sequence, the specified variables are read and written to the connected HIL system.

The following sequence shows an access to multiple variables when you use an XIL API Framework.



Access to a single variable is performed by the following sequence.



Next steps

You can now proceed with getting the application data. Refer to [How to Get Application Properties](#) on page 133.

Related topics

HowTos

[How to Make Simulator Variables Available to XIL API Library Automation Blocks](#)..... 127

References

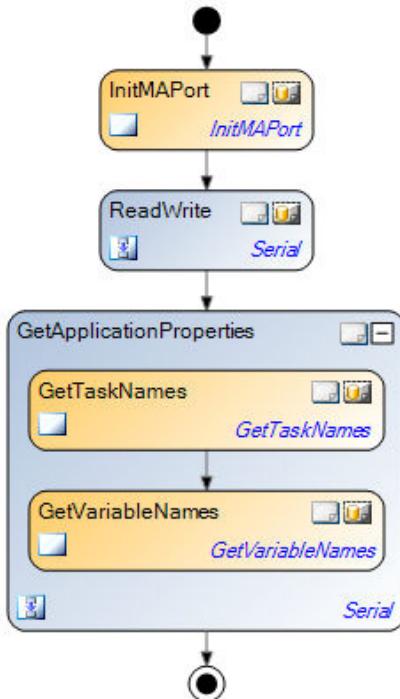
GetValues.....	157
InitBaseValue.....	278
Read.....	299
SetValues.....	158
Write.....	301

How to Get Application Properties

Objective	You can get the simulation application properties (task names and variable names) that you need to configure the data capturing.
Precondition	You have read and written variables (see How to Write and Read Variables on page 129) and all preconditions of the previous steps.
Method	<p>To get application properties</p> <ol style="list-style-type: none">1 Drag a Serial block from the Library Browser to the Sequence Builder.2 Drag a GetTaskNames block from the Library Browser (MAPort folder) to the serial in the Sequence Builder. The block returns the names of all available tasks.3 Set the block's MAPort data object as a reference to the project-specific MAPort data object to access the model that is simulated on the HIL simulator.4 Set the project-specific TaskNames data object as a reference to the block's TaskNames data object. The data object returns the names of all available tasks.5 Drag a GetVariableNames block from the Library Browser (MAPort) to the Serial block in the Sequence Builder. The block returns the List of all available variable names.6 Set the block's MAPort data object as a reference to the project-specific MAPort data object to access the model that is simulated on the HIL simulator.7 Set the block's VariableNames data object as a reference to the project-specific data object that returns the names of all available variables.

Result

When you execute the sequence, you get the data of your application to work with.



Next steps

You can now proceed with capturing data. Refer to [How to Capture Data](#) on page 134.

Related topics

References

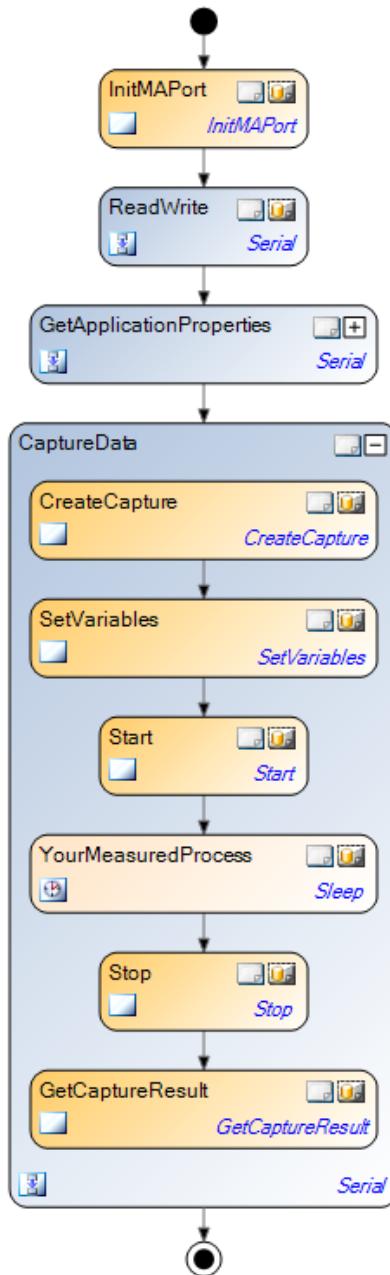
GetTaskNames.....	293
GetVariables.....	242

How to Capture Data

Objective

Capturing is the process of acquiring data from your HIL system in a continuous data stream. You can assign the data to the real-time service or the real-time task [task](#) as soon as it is captured.

Precondition	You have the application properties (see How to Get Application Properties on page 133) and all preconditions of the previous steps.
Method	To capture data <ol style="list-style-type: none">1 In the sequence-specific TaskName String data object, specify the name of the task to use for capturing.2 Drag a CreateCapture block from the Library Browser (MAPort folder) to the Sequence Builder. This block creates a Capture object with the given task.3 Set the block's MAPort, TaskName, and Capture data objects as references to the sequence-specific data objects.4 In the sequence-specific Variables List data object, specify the list of variable to be captured.5 Drag a SetVariables block from the Library Browser (Common - Capturing folder) to the Sequence Builder to set the variables for capturing.6 Set the block's Capture and Variables data objects as references to the sequence-specific data objects.7 Drag a Start block from the Library Browser (Common - Capturing folder) to the Sequence Builder to start data logging and set its Capture data object.8 Add the automation blocks that implement the process you want to measure to the sequence. For demonstration purposes, you can drag a Sleep block from the Library Browser to the Sequence Builder.9 Drag a Fetch block from the Library Browser (Common - Capturing folder) to the Sequence Builder. The block catches the measurement results.10 Set the block's Capture and CaptureResult data objects as references to the sequence-specific data objects.11 Drag a Stop block from the Library Browser (Common - Capturing folder) to stop the data logging and set its Capture data object.12 Drag a GetCaptureResult block from the Library Browser (Common - Capturing folder) to the Sequence Builder to return all measured data.13 Set the block's Capture and CaptureResult data objects as references to the sequence-specific data objects.
Result	When you execute the sequence, the data from your HIL system is captured.



Related topics

References

CreateCapture.....	290
Fetch.....	237
GetCaptureResult.....	239
SetVariables.....	247

Start.....	248
Stop.....	249

Basics on Stimulating Variables via AutomationDesk

Introduction

The XIL API library  provides elements  for stimulus handling via STZ file format.

An STZ file contains a signal description set and optional information about the signal mapping, the description of variables, and the real-time platform. The file is used to generate, download, and control Real-Time Testing sequences, which are executed on the real-time platform to stimulate model variables in real time.

When stimulating simulator variables, Real-Time Testing is used. For information on preconditions and limitations on using Real-Time Testing, refer to [Enabling Real-Time Testing for dSPACE Platforms \(Real-Time Testing Guide !\[\]\(7fc8b7a72794126fa2ec58b7601647d3_img.jpg\)\)](#) and [General Limitations for Real-Time Testing \(Real-Time Testing Guide !\[\]\(a6b72484e2848267f9757dc207861bd7_img.jpg\)\)](#).

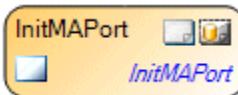
You create and handle STZ files with ControlDesk. For further information, refer to [Basics on Signal Description Sets and Signal Generators \(ControlDesk Signal Editor !\[\]\(850f52cb71ca700ad716ca9250822f03_img.jpg\)\)](#).

With the Signal Editor in AutomationDesk you can also handle STZ files. It allows you to add additional information to a signal description set required for the signal-based testing. For further information, refer to [Implementing Signal-Based Tests \(AutomationDesk Implementing Signal-Based Tests !\[\]\(95981bac9d5c7e8b5884d3a8515b21b0_img.jpg\)\)](#).

Essential automation blocks

There are a few [automation blocks !\[\]\(34a0f644a9833849e47215c8c189f26e_img.jpg\)](#) you have to use in the order as shown in your automation [sequence !\[\]\(63c813c62d5fc369ac0735a860abb07e_img.jpg\)](#) if you want to stimulate variables with AutomationDesk's XIL API library.

- Initialize the MAPort for your AutomationDesk sequence:



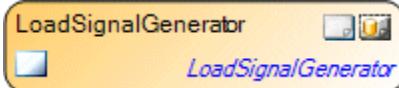
- Create a signal generator:



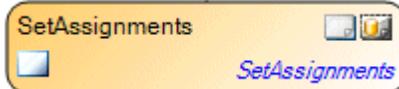
- Create a signal generator STZ reader:



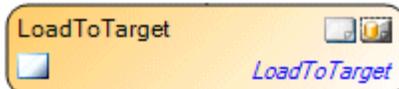
- Load a signal generator:



- Set the assignments to a signal generator (optional):



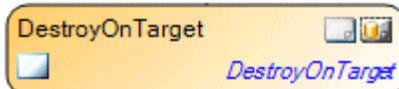
- Download the stimulus to the target:



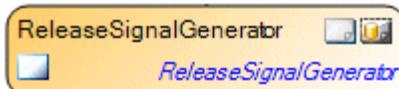
- Start the stimulus:



- Remove the generator from the hardware:



- Release the signal generator instance:



Demo projects

Further AutomationDesk demo projects can be found at <DocumentsFolder>\XIL API. There you find the XILAPIStimulus project that shows you how to use stimulus signals in the TurnSignalTest - BlockExamples - StimulusSignalOnce sequence.

In addition to the demo projects of the XIL API library there is an XILAPICConvenienceExample.zip that uses the automation blocks of the related XIL API Convenience library, see the Examples - SignalGenerator - Signal_and_Capture sequence in the demo project.

How to Migrate a Variable Pool's Data Container to an XIL API Mapping Data Object

Objective

A variable pool is used to configure the access to variables  and provides the following features:

- In a variable pool, you can map aliases for variables to their model paths. By using the aliases in your [sequences](#) and [blocks](#), you decouple the automation from the modeling. If the model path of a variable is modified, you only have to adjust the mapping in the variable pool.
- A short alias is easier to handle than a variable's model path.
- In a variable pool, you can configure the access to multiple variables in one data structure. Thus, you can pass the configuration of all variables to an automation block in one reference.

According to the ASAM standard, AutomationDesk 5.1 introduces the **Mapping** data object, which replaces the variable pools that are implemented as **DataContainer** or **Dictionary** data objects. You are therefore recommended to start migrating the variable pools of your projects according to the following instructions.

Basics

Until AutomationDesk 5.0, there were two different ways to implement a variable pool:

- A data container that contains a String data object for each variable. Each string contains the model path of a variable.
- A dictionary that contains key-value pairs with the aliases as keys and the model paths as values.

The **Mapping** data object is provided by the XIL API library. It contains a mapping table. Each table entry relates an alias in the **Alias** column to the model path of the variable in the **Identifier** column.

To migrate the variable access configuration of your project, you can perform the following steps:

- Export the variable pool to an XML file.
- Import the XML file to the **Mapping** data object.
- Replace the references to the data container that contains the variable pool.
- Replace the references to the strings within the data container.

Restrictions

Your project must not contain inconsistencies.

Preconditions

- The **Mapping** data object must exist in your project. You can add it to the project's top level by dragging it from the XIL API library.
- The **Mapping** data object must be displayed in the **Mapping Viewer**. You can open it by double-clicking the data object.

Method

To migrate a variable pool to a Mapping data object

- 1 In the Project Manager, select **Export as XIL API Mapping** from the context menu of the data container or dictionary that contains the variable pool. This saves the contents of the variable pool to an XML file.

- 2 From the context menu of the Mapping data object, choose Import XIL API Mapping and specify the XML file with the saved variable pool. This writes its contents to the Mapping data object.
The configuration of the variable access can now be referenced from the Mapping data object.
- 3 From the context menu of your project, select Find, and enter the name of the variable pool's data container to be searched. Clear all options except for Reference names and Match whole word, and click Find All.
All references to the variable pool's data container are now migrated to the Mapping data object.
- 4 In the Found items list, enter the name of the Mapping data object and click Replace All to replace the found items.
All references to the variable pool's data container are now migrated to the Mapping data object.
- 5 In the Project Manager, delete the data container that contains the variable pool.
All references to its contents become invalid.
- 6 From the context menu of your project, select Find Inconsistencies, clear all options except for Invalid references. Then select to look in Project, and click Find All.
The Inconsistencies list displays the references to strings that were contained in the variable pool.
- 7 For every entry in the Inconsistency list do the following two steps: Double-click the entry to open the inconsistent block in the Data Object Editor.
8 From the Mapping Viewer, drag the related entry of the mapping list to the inconsistent reference in the Data Object Editor.
The reference to the variable pool is replaced by a reference to the Mapping data object in the format
`<MappingObject>.LabelMapping.<VariableAlias>.Identifier`.
Alternatively, you can click the Browse button in the Reference name field to select the reference via the Data Object Selector.

Result

The variable access within your project is configured in the XIL API Mappings data object.

Reference Information

Where to go from here

Information in this section

Automation Blocks.....	142
Commands And Dialogs.....	302

Automation Blocks

Where to go from here

Information in this section

Platform Management	142
XIL API Framework	155
Description of the data objects and automation blocks for working with an XIL API Framework.	
XIL API Convenience (Model Access)	160
Description of the specific data objects and blocks for accessing your platform using the XIL API Convenience library.	
XIL API (Model Access)	207
Description of the specific data objects and blocks for accessing your platform using the XIL API library.	

Platform Management

Introduction

AutomationDesk's Platform Management library provides automation blocks to load simulation applications to dSPACE simulation platforms and to run them.

Prerequisites

AutomationDesk's Platform Management library is based on the dSPACE *Test Automation APIs* product set. Enhancements of this API immediately take effect on the automation blocks.

Limitation

The Platform Management library does not provide an automation block for registering a platform. To manage a platform via the library's automation blocks, you must first register it, for example, by using AutomationDesk's Platform Manager.

Using Platform Management library features in Python scripts

You can use functions and other definitions of the Platform Management library in Python scripts after you imported the `platformmanagementlib` module to the current namespace.

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

PlatformManagement blocks

[GetPlatformManagement](#)..... 145

To create a PlatformManagement object with the configuration data of the currently registered platforms.

[RefreshPlatformConfiguration](#)..... 151

To refresh your PlatformManagement object with the configuration data of the currently registered platforms.

[GetRegisteredPlatformNames](#)..... 146

To get a list of all the registered platforms.

[GetPlatform](#)..... 144

To get the Platform object of the platform with the specified name.

Platform blocks

[LoadSimulationApplication](#)..... 150

To load a simulation application to a platform.

[GetSimulationApplication](#)..... 147

To get the SimulationApplication object of the simulation application which has been loaded to the specified platform.

SimulationApplication blocks

[StartSimulation](#)..... 152

To restart a simulation that was stopped.

[StopSimulation](#)..... 153

To stop a running simulation.

[GetSimulationState](#)..... 148

To get the current state of the specified simulation.

[UnloadSimulationApplication](#)..... 154

To unload a simulation application from a platform.

Information in other sections

[Managing Platforms and Simulation Applications](#)..... 20

AutomationDesk provides automation capabilities to manage real-time applications on dSPACE real-time hardware and offline simulation applications on VEOS.

[dSPACE Platform Management API Reference](#)

To automate the management of dSPACE platforms, for example registering a platform or loading a real-time application to a platform.

GetPlatform

Graphical representation



Purpose

To get the Platform object of the platform with the specified name.

Description

This block is used to get a Platform object of the platform with the specified unique name. You can use the object, for example, to specify a target platform in the LoadSimulationApplication block.

The necessary information to create the Platform object is retrieved from the PlatformManagement object.

You can get the required PlatformManagement object by first using the GetPlatformManagement block. For a list of the unique names of all the registered platforms, you can use the GetRegisteredPlatformNames block.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
PlatformManagement	In	Variant	None	Specifies the Platform Management object to create the Platform object from.
PlatformUniqueName	In	String	" "	Specifies the unique name of the platform.
Platform	Out	Variant	None	Contains the Platform object.

Related topics

Basics

[Automating Simulation Application Management](#).....48

HowTos

[How to Automate the Loading and Starting of Simulation Applications](#).....51

References

[GetPlatformManagement](#).....145

[GetRegisteredPlatformNames](#).....146

[LoadSimulationApplication](#).....150

GetPlatformManagement

Graphical representation



Purpose

To create a PlatformManagement object with the configuration data of the currently registered platforms.

Description

This block is used to create a PlatformManagement object, which is needed to access platforms. The platforms have to be registered beforehand. For instructions how to do this, refer to [How to Register a dSPACE Platform](#) on page 27.

You can use the provided object to specify the platform management in further operations. For example, the [GetRegisteredPlatformNames](#) block generates a list of unique names of all the registered platforms in the specified PlatformManagement object.

The necessary information to create the PlatformManagement object is retrieved from a platform management server process. Every GetPlatformManagement block starts its own server process. To avoid unnecessary resources, ensure to start only one server process.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
ProgID	In	String	""	Optional: If more than one Platform Management API version is installed, the currently active version of the API is used by default. If you want to use a different API version, assign the program ID of the Platform Management API, followed by the version, to the ProgID parameter, for example, DSPlatformManagementAPI2.1 . Note: With dSPACE Release 2016-B, the Platform Management API 1.0 is discontinued.
PlatformManagement	Out	Variant	None	Contains the PlatformManagement object.

Related topics**Basics**

[Automating Simulation Application Management.....48](#)

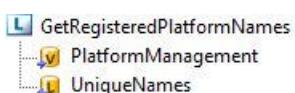
HowTos

[How to Automate the Loading and Starting of Simulation Applications.....51](#)

References

[GetRegisteredPlatformNames.....146](#)

GetRegisteredPlatformNames

Graphical representation**Purpose**

To get a list of all the registered platforms.

Description

This block is used to generate a list of all the currently registered platforms. The unique name of a platform is needed, for example, to specify a platform in the GetPlatform block.

The necessary information to generate the platform name list is retrieved from the PlatformManagement object.

You can get the required PlatformManagement object via the GetPlatformManagement block.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
PlatformManagement	In	Variant	None	Specifies the object of the PlatformManagement
UniqueNames	Out	List	[]	Contains a list of all the registered platforms.

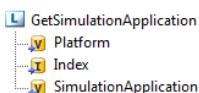
Related topics**Basics**

[Automating Simulation Application Management](#).....48

References

GetPlatform	144
GetPlatformManagement	145

GetSimulationApplication

Graphical representation**Purpose**

To get the **SimulationApplication** object of the simulation application which has been loaded to the specified platform.

Description

If you are working with a DS1007, a MicroLabBox, a MicroAutoBox III, a SCALEXIO platform, or with VEOS, you can use the block's **SimulationApplication** data object to control a simulation application that was loaded to this platform.

You can get the required **Platform** object via the **GetPlatform** block.

With the provided **SimulationApplication** object you can start or stop a simulation by using the **StartSimulation** or the **StopSimulation** block.

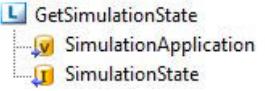
Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
Platform	In	Variant	None	Specifies the Platform object on which the simulation application has been loaded.
Index	In	Int	0	Lets you specify the index of a simulation application if more than one is running on the platform at the same time. The Index data object is only relevant if you work with a SCALEXIO multi-PU system. In this case a PlatformMangementAPI version 2.0 or later is required. The simulation application's index is displayed in the Platform Manager.
SimulationApplication	Out	Variant	None	Contains the SimulationApplication object.

Related topics	Basics
	Automating Simulation Application Management..... 48
	HowTos
	How to Automate the Stopping and Restarting of Simulation Applications..... 53
	References
	GetPlatform..... 144 LoadSimulationApplication..... 150 UnloadSimulationApplication..... 154

GetSimulationState

Graphical representation	
Purpose	To get the current state of the specified simulation.
Description	<p>If you are working with a DS1007, a MicroLabBox, a MicroAutoBox III, a SCALEXIO platform, or with VEOS, you can use this block to determine the current state of a simulation as an integer value with a defined meaning.</p> <p>The returned simulation state is retrieved from the <code>SimulationApplication</code> object. You can get the required <code>SimulationApplication</code> object via the <code>GetSimulationApplication</code> block.</p> <p>The functionality of this block is available for platforms that provide execution control of simulation applications via the Platform Management API. For details, refer to GetSimulationApplication on page 147.</p>

Returned SimulationState Value	State	Description
0	Undefined	The simulation application state is undefined. This temporarily occurs on multicore platforms, if simulation application is stopped or started and not all simultaneous application tasks are already in the same state.

Returned SimulationState Value	State	Description
1	Running	The simulation located on the RAM of the platform has been started. The simulation tasks are being executed.
2	Stopped	The simulation has been stopped. No simulation tasks are running. Simulation can be started again using the StartSimulation block.
3	Terminated	A serious error occurred during the simulation. The simulation tasks finished to a defined state. The simulation can be restarted only by reloading the simulation application.
4	RunningFromFlash	The simulation located at the flash memory of the platform has been started. The simulation tasks are being executed.
5	Paused	This state is only used by VEOS. The simulation has been paused. The simulation tasks are halted and can be continued using the StartSimulation block.
6	Unknown	The simulation application state is unknown.
7	StoppedFromFlash	The simulation located at the flash memory of the platform has been stopped. No simulation tasks are running. Simulation can be started again using the StartSimulation block.
8	Initialized	The simulation on the platform has been initialized.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
SimulationApplication	In	Variant	None	Specifies the SimulationApplication object whose status is retrieved.
SimulationState	Out	String	""	<p>Contains an integer number representing the current state of the simulation.</p> <p>The defined values are:</p> <ul style="list-style-type: none"> ▪ 0 (Undefined) ▪ 1 (Running) ▪ 2 (Stopped)

Name	In / Out	Data Type	Default Value	Description
				<ul style="list-style-type: none"> ▪ 3 (Terminated) ▪ 4 (RunningFromFlash) ▪ 5 (Paused) ▪ 6 (Unknown) ▪ 7 (StoppedFromFlash) ▪ 8 (Initialized)

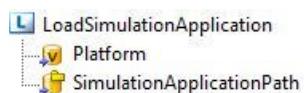
Related topics**Basics**

[Automating Simulation Application Management](#).....48

References

GetPlatform.....	144
GetSimulationApplication.....	147
LoadSimulationApplication.....	150
StartSimulation.....	152
StopSimulation.....	153
UnloadSimulationApplication.....	154

LoadSimulationApplication

Graphical representation**Purpose**

To load a simulation application to a platform.

Description

This block is used to load the specified simulation application to the specified platform. At build time, you can specify whether to start the simulation application automatically after loading.

The target platform is specified by the Platform object, which you can get via the GetPlatform block.

The simulation application to be loaded is defined by either the system description file (SDF), or the platform-specific executable file (PPC, X86, RTA or OSA). For further information, refer to [Basics on Simulation Applications](#) on page 22.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
Platform	In	Variant	None	Specifies the Platform object of the platform to load the simulation application to.
SimulationApplicationPath	In	File	None	Specifies the absolute or relative path and file name of a system description file (SDF) or the platform-specific executable file.

Related topics**Basics**

[Automating Simulation Application Management](#).....48

HowTos

[How to Automate the Loading and Starting of Simulation Applications](#).....51

References

GetPlatform	144
GetSimulationApplication	147
GetSimulationState	148
UnloadSimulationApplication	154

RefreshPlatformConfiguration

Graphical representation**Purpose**

To refresh your PlatformManagement object with the configuration data of the currently registered platforms.

Description

This block is used to refresh the data which is currently stored in your PlatformManagement object. This is necessary if another program, like ControlDesk, has changed the platform configuration since you created or refreshed your PlatformManagement object.

You can create the required PlatformManagement object via the GetPlatformManagement block.

Data objects

This automation block provides the following data object:

Name	In / Out	Data Type	Default Value	Description
PlatformManagement	In/Out	Variant	None	Specifies the PlatformManagement object to be refreshed.

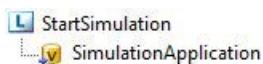
Related topics**Basics**

[Automating Simulation Application Management](#).....48

References

[GetPlatformManagement](#).....145

StartSimulation

Graphical representation**Purpose**

To restart a simulation that was stopped.

Description

If you are working with a DS1007, a MicroLabBox, a MicroAutoBox III, a SCALEXIO platform, or with VEOS, you can use this block to start a simulation that is in **Stopped** state. When you use this block with other platforms, an exception occurs.

You can stop a started simulation by using the **StopSimulation** block. The current state of a simulation can be determined with the **GetSimulationState** block.

The simulation to be started is specified by the **SimulationApplication** object which you can get via the **GetSimulationApplication** block.

Data objects

This automation block provides the following data object:

Name	In / Out	Data Type	Default Value	Description
SimulationApplication	In	Variant	None	Specifies the SimulationApplication object of the simulation to be started.

Related topics**Basics**

[Automating Simulation Application Management](#).....48

HowTos

[How to Automate the Stopping and Restarting of Simulation Applications](#).....53

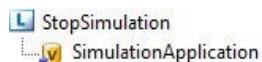
References

[GetSimulationApplication](#).....147

[GetSimulationState](#).....148

[StopSimulation](#).....153

StopSimulation

Graphical representation**Purpose**

To stop a running simulation.

Description

If you are working with a DS1007, a MicroLabBox, a MicroAutoBox III, a SCALEXIO platform, or with VEOS, you can use this block to stop a simulation that is in **Running** state. When you use this block with other platforms, an exception occurs.

You can restart a stopped simulation by using the **StartSimulation** block. The current state of a simulation can be determined with the **GetSimulationState** block.

The simulation to be stopped is specified by the **SimulationApplication** object, which you can get via the **GetSimulationApplication** block.

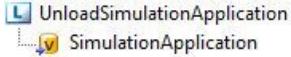
Data objects

This automation block provides the following data object:

Name	In / Out	Data Type	Default Value	Description
SimulationApplication	In	Variant	None	Specifies the SimulationApplication object of the simulation to be stopped.

Related topics	Basics
	Automating Simulation Application Management..... 48
	HowTos
	How to Automate the Stopping and Restarting of Simulation Applications..... 53
	References
	GetSimulationApplication..... 147 GetSimulationState..... 148 StartSimulation..... 152

UnloadSimulationApplication

Graphical representation	
Purpose	To unload a simulation application from a platform.
Description	<p>If you are working with a DS1007, a MicroLabBox, a MicroAutoBox III, a SCALEXIO platform, or with VEOS, you can use this block to unload a specified simulation application from the platform it has been loaded to. It puts the platform in a safe state and marks it as available for new tasks. When you use this block with other platforms, an exception occurs.</p> <p>The simulation application to be unloaded is specified by the <code>SimulationApplication</code> object, which you can get via the <code>GetSimulationApplication</code> block.</p>
Data objects	This automation block provides the following data object:

Name	In / Out	Data Type	Default Value	Description
SimulationApplication	In	Variant	None	Specifies the <code>SimulationApplication</code> object to be unloaded.

Related topics**Basics**

[Automating Simulation Application Management](#).....48

HowTos

[How to Automate the Termination of Simulation Applications](#).....55

References

[GetPlatform](#).....144

[GetSimulationApplication](#).....147

[GetSimulationState](#).....148

[LoadSimulationApplication](#).....150

XIL API Framework

Introduction

The listed data objects and automation blocks for working with an XIL API Framework are available in different libraries. You find:

- Generic data objects in XIL API
- Framework-specific automation blocks in XIL API Convenience

Using XIL API Convenience library features in Python scripts

You can use functions and other definitions of the XIL API Convenience library in Python scripts after you imported the `audxilapiconvenienclib` module to the current namespace.

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

[CheckValues](#).....156

To evaluate a condition that contains variable aliases.

[GetValues](#).....157

To read and report variable values.

[SetValue](#).....158

To write and report variable values.

[TaskName \(Data Object\)](#).....159

To provide the name of the task that executes the simulation application.

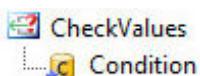
[Variables \(Data Object\)](#).....159

To provide a list of model variables.

Vendor (Data Object).....	160
To provide the vendor of the XIL API implementation.	

CheckValues

Graphical representation



Purpose

To evaluate a condition that contains variable aliases.

Description

The XIL API library provides this block that lets you evaluate a condition that contains variable names, i.e., the aliases. The condition can be specified via the Condition Editor.

By default, this block uses the XIL API Framework for platform access. If you mapped the variable names to model paths in a Mapping data object, you can also use this block after you executed the Disable Framework Support command.

If the evaluated condition is fulfilled, the verdict of the current sequence block is set to Passed, otherwise to Failed.

The condition and the values of the contained variables are added to the result and the report of the executed block.

Tip

You can drag a CheckValues block from the XIL API library to your sequence or you can use the Insert (CheckValues) command on the Home ribbon.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Condition	In	Condition	0	Lets you specify the condition to be evaluated via the Value Editor.

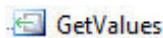
Related topics**HowTos**

[How to Specify Conditions \(AutomationDesk Basic Practices\)](#)

References

Condition (AutomationDesk Basic Practices)	344
Disable Framework Support.....	344
Enable Framework Support.....	346
Insert (CheckValues)....	348

GetValues

Graphical representation**Purpose**

To read and report variable values.

Description

The XIL API library provides this block that lets you read values from variables in a simulation application that is running on a platform.

By default, this block uses the XIL API Framework for platform access. If you mapped the variable names to model paths in a Mapping data object, you can also use this block after you executed the Disable Framework Support command.

You can specify which variables to read by dragging the related entries from the Mapping Viewer to the block. For each variable, a Variant data object is added to the block. The data object name is the same as the alias of the variable. The name of each new data object in the block must be unique.

The read variables and their values are added to the result and the report of the executed block.

Tip

You can drag a GetValues block from the XIL API library to your sequence or you can use the Insert (GetValues) command on the Home ribbon.

Data objects

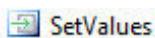
You can add data objects of the following types to the block's parameters:

- Variant
- LabeledValue

Related topics	References								
	<table> <tr> <td>Disable Framework Support.....</td> <td>344</td> </tr> <tr> <td>Enable Framework Support.....</td> <td>346</td> </tr> <tr> <td>Insert (GetValues).....</td> <td>348</td> </tr> <tr> <td>SetValues.....</td> <td>158</td> </tr> </table>	Disable Framework Support.....	344	Enable Framework Support.....	346	Insert (GetValues).....	348	SetValues.....	158
Disable Framework Support.....	344								
Enable Framework Support.....	346								
Insert (GetValues).....	348								
SetValues.....	158								

SetValues

Graphical representation



Purpose

To write and report variable values.

Description

The XIL API library provides this block that lets you write values to variables in a simulation application that is running on a platform.

By default, this block uses the XIL API Framework for platform access. If you mapped the variable names to model paths in a Mapping data object, you can also use this block after you executed the Disable Framework Support command.

You can specify which variables to write by dragging the related entries from the Mapping Viewer to the block. For each variable, a Variant data object is added to the block. The name of the data object is the same as the alias of the variable. The name of each new data object in the block must be unique. In the data objects, you can specify the values to be written by using the Value Editor.

The variable names and the written values are added to the result and the report of the executed block.

Tip

You can drag a SetValues block from the XIL API library to your sequence or you can use the Insert (SetValues) command on the Home ribbon.

Data objects

You can add data objects of the following types to the block parameters:

- Variant
- LabeledValue

Related topics**References**

Disable Framework Support.....	344
Enable Framework Support.....	346
GetValues.....	157
Insert (GetValues).....	348

TaskName (Data Object)

Graphical representation**Purpose**

To provide the name of the task that executes the simulation application.

Description

The XIL API Convenience library provides this string data object with a customized edit dialog. Refer to [Edit \(TaskName\)](#) on page 351.

Related topics**References**

Edit (TaskName).....	351
----------------------	-----

Variables (Data Object)

Graphical representation**Purpose**

To provide a list of model variables.

Description

The XIL API Convenience library provides this List data object with a customized edit dialog. Refer to [Edit \(Variables\)](#) on page 352.

Related topics	Basics
	Notes on Using Tuple, List, And Dictionary Data Objects (AutomationDesk Basic Practices) Using the Value Editor (AutomationDesk Basic Practices)
	References
	Clear Value (AutomationDesk Basic Practices) Edit (Variables) 352

Vendor (Data Object)

Graphical representation	
Purpose	To provide the vendor of the XIL API implementation.
Description	The XIL API Convenience library provides this string data object with a customized edit dialog. Refer to Edit (Vendor) on page 354.

XIL API Convenience (Model Access)

Introduction	The Model Access Port folder of the XIL API Convenience library provides convenient automation blocks to access your platform for reading, writing, capturing and simulating.				
Using XIL API Convenience library features in Python scripts	You can use functions and other definitions of the XIL API Convenience library in Python scripts after you imported the <code>audxilapiconvenienclib</code> module to the current namespace.				
Where to go from here	Information in this section <table border="1"> <tr> <td>Main Elements.....</td> <td>161</td> </tr> <tr> <td>Capture.....</td> <td>171</td> </tr> </table>	Main Elements	161	Capture	171
Main Elements	161				
Capture	171				

StreamToDisk	191
SignalGenerator	194
SimulationApplication	203

Main Elements

Introduction

The main elements of the library are directly available in the Model Access Port folder.

Where to go from here

Information in this section

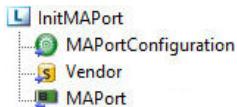
InitMAPort	162
To initialize the model access port data object (MAPort).	
Read	164
To read one variable.	
ReadValues	165
To read several variables.	
ReleaseMAPort	167
To release an MAPort instance.	
Write	168
To write one variable.	
WriteValues	169
To write several variables.	

Information in other sections

Further data objects for working with an XIL API Framework:	
TaskName (Data Object)	159
To provide the name of the task that executes the simulation application.	
Variables (Data Object)	159
To provide a list of model variables.	
Vendor (Data Object)	160
To provide the vendor of the XIL API implementation.	

InitMAPort

Graphical representation



Purpose

To initialize the model access port data object (MAPort).

Description

The InitMAPort automation block is used to prepare the connection to the model that is simulated on the HIL simulator.

The MAPort data object is created at the first execution of this block. Subsequent executions have no effect.

Note

The following preconditions have to be fulfilled before you can use the MAPort for accessing a simulation platform:

- The simulation platform has to be registered, refer to [Registering and Managing dSPACE Platforms](#) on page 26.
- A loaded simulation application must match the MAPort configuration.
- If the real-time application is not already loaded to the platform, it is automatically loaded but not started. You have to explicitly start the simulation before executing model access.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPortConfiguration	In	MAPortConfiguration (Data Object)	{"ApplicationPath": "", "PlatformIdentifier": ""}	<p>The behavior depends on the XIL API layer you work with:</p> <ul style="list-style-type: none"> ▪ XIL API Framework: Lets you select the model access port name from the list of ports that are configured in the XIL API Framework. If the specified simulation application is already running on the platform and the forceConfig parameter is set to True, the simulation is stopped and reloaded. ▪ XIL API Testbench: Sets the following items: <ul style="list-style-type: none"> ▪ ApplicationPath Absolute path and name of variable file (trc or sdf)

Name	In / Out	Type	Default Value	Description
Vendor	In	String	"dSPACE"	<ul style="list-style-type: none"> ▪ <i>PlatformIdentifier</i> ds1006, ds1006_1, ..., ds1401, ds1401_1, ... IP address (for SCALEXIO MC), Name of the MP system (SCALEXIO MP), or VEOS If the specified simulation application is already running, it is not interrupted. <p>This data object auto-references a data object of the same type at a higher hierarchy level.</p> <p>Lets you select the vendor and version of the used XIL API implementation from the list of installed implementations in the Select vendor dialog. If dSPACE is specified as the vendor, the dSPACE XIL API implementation version that matches the active AutomationDesk version is used. To specify another version or another vendor, the following format is used:</p> <pre><VendorName>_<ProductName>_<ProductVersion></pre> <p>Example:</p> <pre>dSPACE GmbH_XIL API_2019-A</pre> <p>It is recommended to use a globally managed Vendor data object as a reference providing the vendor name. For more information, refer to Vendor (Data Object) on page 160.</p>
MAPort	Out	MAPort (Data Object)	None	Returns the initialized MAPort data object. This data object auto-references a data object of the same type at a higher hierarchy level.

Tip

- You can get the list of vendor names for the currently available XIL API implementations via the `xilapi.GetVendorNames()` method.
- If you want to use an XIL API implementation of a vendor other than dSPACE, you must adjust the MAPortConfiguration data object. Refer to [MAPortConfiguration \(Data Object\)](#) on page 289.

Access via Exec block

Alternatively, you can initialize the MAPort data object via Exec block. For further information, refer to [MAPort \(Data Object\)](#) on page 286.

Related topics**Basics**

[Accessing Simulation Platforms via XIL API.....](#) 117

HowTos

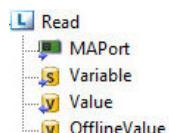
[How to Build a Basic Sequence for Accessing Simulator Variables.....](#) 78

References

[MAPort \(Data Object\).....](#) 286

[MAPortConfiguration \(Data Object\).....](#) 289

Read

Graphical representation**Purpose**

To read one variable.

Description

The Read automation block allows you to read one variable of scalar, vector or matrix type from the platform.

Note

Before this block is used the platform's model access port (MAPort) must have been initialized. To initialize the MAPort, use the `InitMAPort` block. If no XIL API Framework is initialized, release the MAPort by using the `ReleaseMAPort` block at the end of your sequence. If an XIL API Framework is initialized, the port is released automatically when the framework is shut down.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Lets you specify the platform to connect to.
Variable	In	String	""	Lets you specify the model path of the variable to be . Specify the path by referencing a String data object contained in the project's VariablePool data container.
Value	Out	Variant	None	
OfflineValue	In	Variant	None	Lets you specify the value to be used in offline operation mode.

Related topics**Basics**

Accessing Simulation Platforms via XIL API.....	117
Basics of Operation Modes (AutomationDesk Basic Practices 	

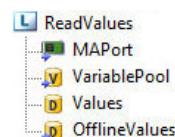
HowTos

How to Write Variable Values.....	87
-----------------------------------	----

References

InitMAPort.....	162
ReadValues.....	165
ReleaseMAPort.....	167
Write.....	168
WriteValues.....	169

ReadValues

Graphical representation**Purpose**

To read several variables from the platform.

Description	The ReadValues automation block lets you specify variables to read from the platform.
--------------------	---

Note

Before this block is used the platform's model access port (MAPort) must have been initialized. To initialize the MAPort, use the InitMAPort block. If no XIL API Framework is initialized, release the MAPort by using the ReleaseMAPort block at the end of your sequence. If an XIL API Framework is initialized, the port is released automatically when the framework is shut down.

Data objects		This automation block provides the following data objects:		
Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Lets you specify the platform to connect to.
VariablePool	In	Variant	None	<p>The behavior depends on the XIL API layer you work with:</p> <ul style="list-style-type: none"> ▪ XIL API Framework: This data object is not needed. The variables that can be used at the block are specified via the variable mapping in the framework configuration. ▪ XIL API Testbench: Lets you specify the set of variables that can be used at the block. Three different methods: <ul style="list-style-type: none"> ▪ Create an XIL API Mapping data object in your project and reference it. The contained Alias values are interpreted as variable names and the associated Identifier values as the model paths. ▪ (For compatibility reasons only) Create one variable pool data container and reference it at the block's VariablePool data object. Specify each variable contained in the pool as a String data object in the data container: <ul style="list-style-type: none"> ▪ <VariableName> the name of the String data object ▪ <ModelPath> the value of the String data object ▪ (For compatibility reasons only) Create a dictionary and reference it. The value of the dictionary is interpreted as <VariableName>:<ModelPath>. It is recommended to use the XIL API Mapping data object.
Values	In/Out	Dictionary	{}	<p>Lets you specify which variables to read by entering their names into the dictionary. If the dictionary is empty, all variables contained in the project's VariablePool are read.</p> <p>After block execution the dictionary contains the values of the model variables that have been read as key-value pairs.</p> <p>Example: {"BatteryVoltage":None,"TurnSignalLever":None}</p>
OfflineValues	In	Dictionary	{}	Lets you specify the value(s) to be used in offline operation mode.

Related topics**Basics**

Accessing Simulation Platforms via XIL API.....	117
Basics of Operation Modes (AutomationDesk Basic Practices)	

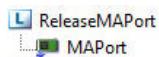
HowTos

How to Write Variable Values.....	87
-----------------------------------	----

References

InitMAPort.....	162
Read.....	164
ReleaseMAPort.....	167
Write.....	168

ReleaseMAPort

Graphical representation**Purpose**

To release an MAPort instance.

Description

This automation block is used to release the currently used MAPort, for example, if you want to switch to another platform or if you want to load another simulation application.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Contains the instantiated MAPort data object to be released.

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

HowTos

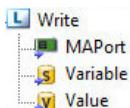
[How to Build a Basic Sequence for Accessing Simulator Variables](#)..... 78

References

[MAPort \(Data Object\)](#)..... 286

[Release MAPort](#)..... 365

Write

Graphical representation**Purpose**

To write one variable.

Description

The Write automation block allows you to write *one* variable of scalar, vector or matrix type to the platform.

Note

Before this block is used the platform's model access port (MAPort) must have been initialized. To initialize the MAPort, use the InitMAPort block. If no XIL API Framework is initialized, release the MAPort by using the ReleaseMAPort block at the end of your sequence. If an XIL API Framework is initialized, the port is released automatically when the framework is shut down.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Lets you specify the platform to connect to.

Name	In / Out	Type	Default Value	Description
Variable	In	String	""	Lets you specify the model path of the variable to be . Specify the path by referencing a String data object contained in the project's VariablePool data container.
Value	In	Variant	None	

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

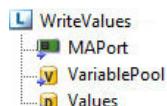
HowTos

[How to Write Variable Values](#)..... 87

References

InitMAPort.....	162
Read.....	164
ReadValues.....	165
ReleaseMAPort.....	167
WriteValues.....	169

WriteValues

Graphical representation**Purpose**

To write several variables to the platform.

Description

The WriteValues automation block lets you specify variables to be written to the platform.

Note

Before this block is used the platform's model access port (MAPort) must have been initialized. To initialize the MAPort, use the InitMAPort block. If no XIL API Framework is initialized, release the MAPort by using the ReleaseMAPort block at the end of your sequence. If an XIL API Framework is initialized, the port is released automatically when the framework is shut down.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Lets you specify the platform to connect to.
VariablePool	In	Variant	None	<p>The behavior depends on the XIL API layer you work with:</p> <ul style="list-style-type: none"> ▪ XIL API Framework: This data object is not needed. The variables that can be used at the block are specified via the variable mapping in the framework configuration. ▪ XIL API Testbench: Lets you specify the set of variables that can be used at the block. Three different methods: <ul style="list-style-type: none"> ▪ Create an XIL API Mapping data object in your project and reference it. The contained Alias values are interpreted as variable names and the associated Identifier values as the model paths. ▪ (For compatibility reasons only) Create <i>one</i> variable pool data container and reference it at the block's VariablePool data object. Specify each variable contained in the pool as a String data object in the data container: <ul style="list-style-type: none"> ▪ <VariableName> the name of the String data object ▪ <ModelPath> the value of the String data object ▪ (For compatibility reasons only) Create a dictionary and reference it. The value of the dictionary is interpreted as <VariableName>:<ModelPath>. It is recommended to use the XIL API Mapping data object.
Values	In	Dictionary	Dictionary	<p>Lets you specify which value to write to each variable. Use key-value pairs with the variable names specified in the project's VariablePool data container as the key.</p> <p>Example: {"BatteryVoltage": 13.4, "TurnSignalLever": -1}</p>

Related topics**Basics**

[Accessing Simulation Platforms via XIL API.....](#) 117

HowTos

[How to Write Variable Values.....](#) 87

References

InitMAPort.....	162
ReadValues.....	165
ReleaseMAPort.....	167
Write.....	168

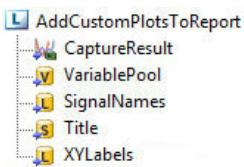
Capture

Introduction	The Capture folder provides convenient automation blocks you can use to access your platform for capturing.
---------------------	---

Where to go from here	Information in this section
	AddCustomPlotsToReport 172 To add selected signals of a capture to the report.
	AddPlotsToReport 173 To add all signals of a capture to the report.
	ConfigureStartCondition 175 To specify a start condition of a capture.
	ConfigureStopCondition 177 To specify a stop condition of a capture.
	ConfigureStopDuration 179 To specify the duration of a capture.
	GetCaptureResult 181 To get data from the capture and set the capture state.
	GetCaptureState 182 To return the status of a capture.
	GetDictionaryFromCaptureResult 184 To get a Dictionary data object filled with the contents of a CaptureResult.
	InitializeCapture 186 To initialize a capture.
	ReleaseCapture 188 To release a capture instance.
	StartCapture 189 To start a capture manually.
	StopCapture 190 To stop a capture manually.

AddCustomPlotsToReport

Graphical representation



Purpose

To add selected signals of a capture to the report.

Description

The AddCustomPlotsToReport automation block lets you specify which results of a capture are to be added to the report.

The CaptureResult data object is returned by, for example, a GetCaptureResult block or a GetRecordedData block earlier in the sequence.

Via an option in the Report Page of the AutomationDesk Options dialog, you can specify whether to add an MDF file (ASAM Common MDF Version 4.1, file name extension: MF4) with the captured signals to the report.

Tip

For an example how to configure plots for capture results, refer to the following sequence in the demo project:

*XILAPIConvenienceExample.Examples.CapturingDemos.Capture_and_Custo
mPlot.*

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
CaptureResult	In	CaptureResult (Data Object)	None	This data object auto-references a data object of the same type at a higher hierarchy level.
VariablePool	In	Variant	None	<p>The behavior depends on the XIL API layer you work with:</p> <ul style="list-style-type: none"> ▪ XIL API Framework: This data object is not needed. The variables that can be used at the block are specified via the variable mapping in the framework configuration. ▪ XIL API Testbench: Lets you specify the set of variables that can be used at the block. Three different methods: <ul style="list-style-type: none"> ▪ Create an XIL API Mapping data object in your project and reference it. The contained Alias values are interpreted as variable names and the associated Identifier values as the model paths. ▪ (For compatibility reasons only) Create one variable pool data container and reference it at the block's VariablePool data object.

Name	In / Out	Type	Default Value	Description
SignalNames	In	List	[]	<p>Specify each variable contained in the pool as a String data object in the data container:</p> <ul style="list-style-type: none"> ▪ <VariableName> the name of the String data object ▪ <ModelPath> the value of the String data object ▪ (For compatibility reasons only) Create a dictionary and reference it. The value of the dictionary is interpreted as <VariableName>:<ModelPath>. It is recommended to use the XIL API Mapping data object.
Title	In	String	None	Lets you specify the title of the plot to be generated.
XYLabels	In	List	[]	Lets you specify labels to be added to the axes of each subplot. The labels have to be consistent with the specification made in the SignalNames data object.

Related topics**Basics**

[Basics of Operation Modes \(AutomationDesk Basic Practices\)](#)

HowTos

[How to Add Plots of the Captured Data to your Report.....](#) 102

References

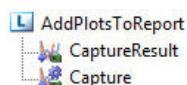
[GetCaptureResult.....](#) 181

[GetRecordedData.....](#) 191

[Options \(AutomationDesk Basic Practices\)](#)

[Report Page \(AutomationDesk Basic Practices\)](#)

AddPlotsToReport

Graphical representation**Purpose**

To add all signals of a capture to the report.

Description The AddPlotsToReport automation block lets you add a plot containing all signals belonging to one signal group to the report.

The CaptureResult data object is returned by, for example, a GetCaptureResult block or a GetRecordedData block earlier in the sequence.

Note

Capture and CaptureResult data objects must match.

If you execute the XIL API built-in library in offline operation mode, results of the capture are not added to the report, because the Capture data object is not allowed to be used in offline operation mode.

Via an option in the Report Page of the AutomationDesk Options dialog, you can specify whether to add an MDF file (ASAM Common MDF Version 4.1, file name extension: MF4) with the captured signals to the report.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
CaptureResult	In	CaptureResult (Data Object)	None	
Capture	In	Capture (Data Object)	None	Represents the configuration of a data capture returned by an InitializeCapture block earlier in the sequence.

Related topics

Basics

Accessing Simulation Platforms via XIL API.....	117
Basics of Operation Modes (AutomationDesk Basic Practices 	

HowTos

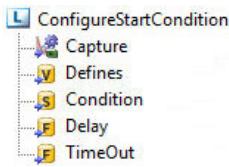
How to Add Plots of the Captured Data to your Report.....	102
---	-----

References

GetCaptureResult.....	181
GetRecordedData.....	191
Options (AutomationDesk Basic Practices 	
Report Page (AutomationDesk Basic Practices 	

ConfigureStartCondition

Graphical representation



Purpose

To specify a start condition of a capture.

Description

The automation block lets you specify a trigger condition to a capture, if required.

Note

Before this block is used a capture has to be initialized by an InitializeCapture block earlier in the sequence.

The capture has to be started by using a StartCapture block later in the sequence.

You can stop the capture by placing a StopCapture block later in the sequence.

Tip

For an example how to configure trigger conditions, refer to the following sequence in the demo project:

XILAPIConvenienceExample.Examples.CapturingDemos.Capture_StartCondition_StopCondition.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	<p>Represents the configuration of a data capture returned by an InitializeCapture block earlier in the sequence.</p> <p>This data object auto-references a data object of the same type at a higher hierarchy level.</p>
Defines	In	Variant	None	<p>The behavior depends on the XIL API layer you work with:</p> <ul style="list-style-type: none"> ▪ XIL API Framework: This data object is not needed. The set of variable aliases that can be used in the block's Condition data object are specified in the variable mapping in the framework configuration. ▪ XIL API Testbench:

Name	In / Out	Type	Default Value	Description
				Lets you specify the set of variable aliases that can be used in the block's Condition data object by referencing a project-specific Mapping data object, a data container, or a Dictionary data object. <ul style="list-style-type: none"> ▪ Using a Mapping data object: You have to add each variable to the Mapping data object via the Mapping Editor. ▪ Using a DataContainer: For each variable, you have to add a String data object to the container, where the name of the String data object is the VariableName, and the value of the String data object is the ModelPath. ▪ Using a Dictionary: For each variable, you have to add a key-value pair as <VariableName>:<ModelPath>.
Condition	In	String	" "	Lets you specify the trigger condition for in ASAM General Expression Syntax. You can use the variables from the block's Defines data object. Example: posedge(BatteryVoltage, 12.0) If you use the Expression Editor to specify the string, the condition can be syntactically checked while you are editing it. For more information, refer to <i>Appendix A. Syntax of Watcher Conditions in ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf</i>
Delay	In	Float	0.0	Sets the trigger delay in seconds. <ul style="list-style-type: none"> ▪ Positive start trigger: If the delay for the start trigger is positive, capturing starts after the specified amount of time passed since the start trigger became true. If no start trigger is specified, capturing starts after the specified amount of time passed since the Start block was executed. ▪ Negative start trigger: If the delay for the start trigger is negative, the capture result buffers the values for the specified time before the start trigger event occurs. A start trigger event is ignored if it occurs before the negative delay is reached for the first time.
TimeOut	In	Float	10.0	Lets you specify the maximum time to wait for the to occur in seconds. To disable the timeout, set the value of this data object to -1.

Related topics**Basics**

Accessing Simulation Platforms via XIL API..... 117

HowTos

How to Capture Data..... 96

References

AddCustomPlotsToReport..... 172

AddPlotsToReport..... 173

ConfigureStopCondition..... 177

ConfigureStopDuration..... 179

Edit (Condition String in ASAM GES) (AutomationDesk Basic Practices)	181
GetCaptureResult.....	181
GetCaptureState.....	182
InitializeCapture.....	186
StartCapture.....	189

ConfigureStopCondition

Graphical representation



Purpose

To specify a stop condition of a capture.

Description

The automation block lets you specify a trigger condition to a capture, if required.

Note

Before this block is used a capture has to be initialized by an InitializeCapture block earlier in the sequence.

The capture has to be started by using a StartCapture block later in the sequence.

You can stop the capture by placing a StopCapture block later in the sequence.

Tip

For an example how to configure trigger conditions, refer to the following sequence in the demo project:

XILAPIConvenienceExample.Examples.CapturingDemos.Capture_StartCondition_StopCondition.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	<p>Represents the configuration of a data capture returned by an InitializeCapture block earlier in the sequence.</p> <p>This data object auto-references a data object of the same type at a higher hierarchy level.</p>
Defines	In	Variant	None	<p>The behavior depends on the XIL API layer you work with:</p> <ul style="list-style-type: none"> ▪ XIL API Framework: This data object is not needed. The set of variable aliases that can be used in the block's Condition data object are specified in the variable mapping in the framework configuration. ▪ XIL API Testbench: Lets you specify the set of variable aliases that can be used in the block's Condition data object by referencing a project-specific Mapping data object, a data container, or a Dictionary data object. <ul style="list-style-type: none"> ▪ Using a Mapping data object: You have to add each variable to the Mapping data object via the Mapping Editor. ▪ Using a DataContainer: For each variable, you have to add a String data object to the container, where the name of the String data object is the VariableName, and the value of the String data object is the ModelPath. ▪ Using a Dictionary: For each variable, you have to add a key-value pair as <VariableName>:<ModelPath>.
Condition	In	String	" "	<p>Lets you specify the trigger condition for in ASAM General Expression Syntax. You can use the variables from the block's Defines data object.</p> <p>Example: <code>posedge(BatteryVoltage, 12.0)</code></p> <p>If you use the Expression Editor to specify the string, the condition can be syntactically checked while you are editing it.</p> <p>For more information, refer to <i>Appendix A. Syntax of Watcher Conditions</i> in ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf</p>
Delay	In	Float	0.0	<p>Sets the trigger delay in seconds.</p> <p>If the delay for the stop trigger is positive, capturing stops after the specified amount of time passed since the stop trigger occurred.</p> <p>A negative value is not supported and causes an error message.</p>
TimeOut	In	Float	10.0	<p>Lets you specify the maximum time to wait for the to occur in seconds.</p> <p>To disable the timeout, set the value of this data object to -1.</p>

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

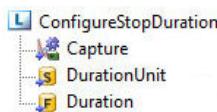
HowTos

[How to Capture Data](#)..... 96

References

AddCustomPlotsToReport	172
AddPlotsToReport	173
ConfigureStartCondition	175
Edit (Condition String in ASAM GES) (AutomationDesk Basic Practices)	176
GetCaptureResult	181
GetCaptureState	182
InitializeCapture	186
StartCapture	189

ConfigureStopDuration

Graphical representation**Purpose**

To specify the duration of a capture.

Description

The ConfigureStopDuration automation block lets you specify how long a capture is to be performed.

You can specify the duration in seconds or in samples.

- If eSAMPLES is set, the Delay parameter of the watcher defined at the ConfigureStartCondition block and the duration of the DurationWatcher are interpreted as A_INT32.
- If eSECONDS is set, these values are interpreted as A_FLOAT64.

Note

Before this block is used a capture has to be initialized by an InitializeCapture block earlier in the sequence.

The capture has to be started by using a StartCapture block later in the sequence.

You can stop the capture by placing a StopCapture block later in the sequence.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Represents the configuration of a data capture returned by an InitializeCapture block earlier in the sequence.
DurationUnit	In	String	eSECONDS	Lets you select the duration's unit: ▪ eSECONDS To specify a time to stop the capture. ▪ eSAMPLE To specify a number of samples to stop the capture after.
Duration	In	Float	10.0	Lets you specify the duration of the capture. The duration you specify at this block overwrites the default duration specified at the InitializeCapture block.

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

HowTos

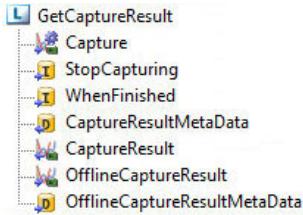
[How to Capture Data](#)..... 96

References

AddCustomPlotsToReport.....	172
AddPlotsToReport.....	173
ConfigureStartCondition.....	175
GetCaptureResult.....	181
GetCaptureState.....	182
InitializeCapture.....	186
StartCapture.....	189

GetCaptureResult

Graphical representation



Purpose

To get data from the capture and set the capture state.

Description

The GetCaptureResult automation block lets you get data from the capture and set the capture state.

Note

Before this block is used a capture has to be initialized by an InitializeCapture block earlier in the sequence.

The capture has to be started by using a StartCapture block later in the sequence.

You can stop the capture by placing a StopCapture block later in the sequence.

Data objects

This automation block provides the following data objects:

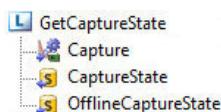
Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Represents the configuration of a data capture returned by an InitializeCapture block earlier in the sequence.
StopCapturing	In	Int	1	Specifies whether to stop the capture after fetching the data. When stopped, the capture is in eCONFIGURED state. Possible values: <ul style="list-style-type: none">▪ 1 = stop▪ 0 = don't stop
WhenFinished	In	Int	0	Lets you specify whether to wait capture's end condition to be met,

Name	In / Out	Type	Default Value	Description
CaptureResultMetaData	Out	Dictionary	{'StartTriggerTimeOut': 'False', 'StopTriggerTimeOut': 'False'}	i.e., whether to wait for the capture to be in eFINISHED state. Possible values: <ul style="list-style-type: none">▪ 1 = wait▪ 0 = don't wait
CaptureResult	Out	CaptureResult (Data Object)	None	Contains the metadata of the CaptureResult, such as the occurrence of a timeout.
OfflineCaptureResult	In	CaptureResult (Data Object)	None	The result to be written to the CaptureResult data object when executed in offline operation mode.
OfflineCaptureResultMetaData	In	Dictionary	{}	The metadata to be used in offline operation mode.

Related topics**Basics**[Basics of Operation Modes \(AutomationDesk Basic Practices\)](#)**HowTos**[How to Capture Data.....](#) 96**References**

GetCaptureState.....	182
InitializeCapture.....	186

GetCaptureState

Graphical representation**Purpose**

To return the status of a capture.

Description

The GetCaptureState automation block lets you return the status of a capture. You can use this information to specify conditions in your sequence to control fetching the data.

Note

Before this block is used a capture has to be initialized by an InitializeCapture block earlier in the sequence.

The capture has to be started by using a StartCapture block later in the sequence.

You can stop the capture by placing a StopCapture block later in the sequence.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Represents the configuration of a data capture returned by an InitializeCapture block earlier in the sequence.
CaptureState	Out	String	" "	<p>At block execution, the current status of the specified Capture data object is written to the block's Status data object.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ▪ eACTIVATED ▪ eCONFIGURED ▪ eRUNNING ▪ eFINISHED
OfflineCaptureState	In	String	" "	<p>Lets you specify the value to be returned in offline operation mode.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ▪ eACTIVATED ▪ eCONFIGURED ▪ eRUNNING ▪ eFINISHED

Related topics**Basics**

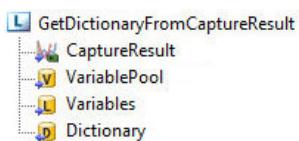
[Basics of Operation Modes \(AutomationDesk Basic Practices\)](#)

References

GetCaptureResult.....	181
InitializeCapture.....	186

GetDictionaryFromCaptureResult

Graphical representation



Purpose

To get a Dictionary data object filled with the contents of a CaptureResult.

Description

This automation block lets you copy and convert a CaptureResult data object that you created with a block from the XIL API library to a Dictionary data object.

The structure of the returned dictionary depends on the number of processors:

- If the result has been captured on a single-processor system, the dictionary contains one item for each captured variable.

The key of each item is the variable's name. The item's value is a List data object that contains all the measured values of the variable in chronological order.

Additionally there is an item with the key `xAxis`. Its value is a List data object that contains all the time stamps of the capture. All List data objects are correlated by their indices.

- If the result has been captured on a multiprocessor system, the dictionary contains one item for each signal group involved.

The key for each item is the name of the application path and the value is a Dictionary data object that is structured like a capture result of a single processor capture.

Note

- After the execution of this block, the same capture result data is stored in the source CaptureResult data object and the target Dictionary data object. When the CaptureResult data object is no longer needed, you should free its allocated memory to reduce memory consumption. To do so, use the `ClearValues` block for the CaptureResult data object.
- At the end of your test sequence, you should free the allocated memory of the Dictionary data object containing the converted capture result to reduce the memory required on the local disk before saving the project. To do so, use the `ClearValues` block for the CaptureResult data object.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
CaptureResult	In	CaptureResult (Data Object)	None	Lets you specify the CaptureResult that was returned by an XIL API library block, for example, a GetCaptureResult, GetRecordedData or Fetch block earlier in the sequence.
VariablePool	In	Variant	None	Lets you specify the variables that are contained in the CaptureResult data object by referencing a project-specific Mapping data object, a data container, or a Dictionary data object. <ul style="list-style-type: none"> ▪ Using a Mapping data object: You have to add each variable to the Mapping data object via the Mapping Editor. ▪ Using a DataContainer: For each variable you have to add a String data object to the container, where the name of the String data object is the VariableName and the value of the String data object is the ModelPath. ▪ Using a Dictionary: For each variable, you have to add a key-value pair as <VariableName>:<ModelPath>.
Variables	In	List	[]	Lets you specify a subset of the variables contained in the CaptureResult data object that are to be copied and converted by referencing a List data object. The subset used depends on the VariablePool data object: <ul style="list-style-type: none"> ▪ If the VariablePool is not empty: For each variable, you have to add a String data object containing the VariableName to the List data object. ▪ If the VariablePool is empty: For each variable, you have to add a String data object containing the variable's ModelPath to the List data object. If the Variables list is empty, all the variables contained in the CaptureResult are used.
Dictionary	Out	Dictionary	{}	Contains the converted contents of the CaptureResult.

Related topics**Basics**

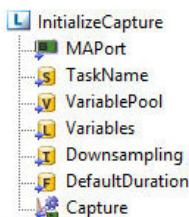
[Basics of Memory Handling \(AutomationDesk Basic Practices\)](#)

References

CaptureResult (Data Object).....	223
Fetch.....	237
GetCaptureResult.....	181
GetRecordedData.....	191

InitializeCapture

Graphical representation



Purpose

To create and initialize an XIL API capture.

Description

The InitializeCapture automation block lets you initialize a capture. It returns a Capture object to be used at other automation blocks.

Note

Before this block is used the platform's model access port (MAPort) must have been initialized. To initialize the MAPort, use the InitMAPort block. If no XIL API Framework is initialized, release the MAPort by using the ReleaseMAPort block at the end of your sequence. If an XIL API Framework is initialized, the port is released automatically when the framework is shut down.

Note

Before this block is used a capture has to be initialized by an InitializeCapture block earlier in the sequence. The capture has to be started by using a StartCapture block later in the sequence. You can stop the capture by placing a StopCapture block later in the sequence.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Lets you specify the platform to connect to.
TaskName	In	String	""	Lets you specify the name of the task If an XIL API Framework is initialized, you can select the task name from the list of tasks that are configured in the XIL API Framework.
VariablePool	In	Variant	None	The behavior depends on the XIL API layer you work with: <ul style="list-style-type: none">▪ XIL API Framework:

Name	In / Out	Type	Default Value	Description
				<p>This data object is not needed. The variables that can be used at the block are specified via the variable mapping in the framework configuration.</p> <ul style="list-style-type: none"> ▪ XIL API Testbench: Lets you specify the set of variables that can be used at the block. Three different methods: <ul style="list-style-type: none"> ▪ Create an XIL API Mapping data object in your project and reference it. The contained Alias values are interpreted as variable names and the associated Identifier values as the model paths. ▪ (For compatibility reasons only) Create one variable pool data container and reference it at the block's VariablePool data object. Specify each variable contained in the pool as a String data object in the data container: <ul style="list-style-type: none"> ▪ <VariableName> the name of the String data object ▪ <ModelPath> the value of the String data object ▪ (For compatibility reasons only) Create a dictionary and reference it. The value of the dictionary is interpreted as <VariableName>:<ModelPath>. <p>It is recommended to use the XIL API Mapping data object.</p>
Variables	In	List	[]	<p>The behavior depends on the XIL API layer you work with:</p> <ul style="list-style-type: none"> ▪ XIL API Framework: Lets you specify the variables that are captured by selecting their aliases from the list of variables that are configured in the framework in the Select Variables dialog. ▪ XIL API Testbench: Lets you specify a subset of variables that are captured at the block by listing variable names specified in the project's VariablePool data container.
Downsampling	In	Int	1	Contains the downsampling factor. At a factor of n, a set of data is stored every n-th sampling period. The sampling period is defined in the model as the increment of sampling steps. The downsampling factor must be a positive integer value.
DefaultDuration	In	Float	10.0	<p>Lets you specify a default duration of the capture in seconds. If you want to define a duration in samples, use the ConfigureStopDuration block later in your sequence.</p> <p>If you want to define a condition to start or stop the capture, use the ConfigureStartCondition block or ConfigureStopCondition block later in your sequence.</p>
Capture	Out	Capture (Data Object)	None	Returns the initialized Capture object.

Tip

To add a variable name to the Variables data object, you can drag an item from the VariablePool while pressing the **Ctrl** key and drop it to the list.

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

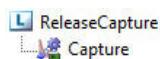
HowTos

[How to Capture Data](#)..... 96

References

AddCustomPlotsToReport.....	172
ConfigureStartCondition.....	175
ConfigureStopCondition.....	177
ConfigureStopDuration.....	179
Edit (Variables).....	352
GetCaptureResult.....	181
GetCaptureState.....	182
InitMAPort.....	162
ReleaseMAPort.....	167
StartCapture.....	189
StopCapture.....	190

ReleaseCapture

Graphical representation**Purpose**

To release a capture instance.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Contains the instantiated Capture data object.

Access via Exec block

The Capture data object provides the possibility to access the `Release` method within an Exec block.

```
_AD_.Capture.Release()
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API.....](#) 117

HowTos

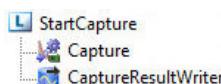
[How to Release Resources After Data Capturing.....](#) 105

References

[Capture \(Data Object\).....](#) 231

[Release Capture.....](#) 364

StartCapture

Graphical representation**Purpose**

To start a capture manually.

Description

The StartCapture automation block lets you start a capture independently of a configured start condition. The capture result is stored in the referenced CaptureResultWriter object.

You can manually stop the capture by placing a StopCapture block later in the sequence.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Represents the configuration of a data capture returned by an InitializeCapture block earlier in the sequence.
CaptureResultWriter	In	CaptureResultWriter (Data Object)	None	Specifies where and in which format the capture result is written. <ul style="list-style-type: none"> ▪ None: Capture result is written to the local memory of your PC. ▪ Instantiated CaptureResultWriter: Capture result is written to the file that is specified in the CaptureResultWriter data object.

Access via Exec block

Alternatively, you can start a capture via script within an Exec block.

```
_AD_.Capture.Start()
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

HowTos

[How to Capture Data](#)..... 96

References

Capture (Data Object).....	231
CaptureResultWriter (Data Object).....	233
ConfigureStartCondition.....	175
ConfigureStopCondition.....	177
ConfigureStopDuration.....	179
GetCaptureResult.....	181
GetCaptureState.....	182
StopCapture.....	190

StopCapture

Graphical representation**Purpose**

To stop a capture manually.

Description

The StopCapture automation block lets you stop a running capture independently of the configured stop conditions of the referenced Capture instance. The state of the capture is set to eCONFIGURED, refer to [CaptureState \(Data Object\)](#) on page 234.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Represents the configuration of a data capture returned by an InitializeCapture block earlier in the sequence.

Access via Exec block

Alternatively, you can stop a capture via script within an Exec block.

```
_AD_.Capture.Stop()
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

Capture (Data Object).....	231
ConfigureStopCondition.....	177
ConfigureStopDuration.....	179
GetCaptureResult.....	181
GetCaptureState.....	182
StartCapture.....	189

StreamToDisk

Introduction

The StreamToDisk folder of the XIL API Convenience library provides convenient automation blocks you can use to start streaming the data and to get the streamed data later on.

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

[GetRecordedData](#)..... 191

To read a capture result from file.

[StartStreamToDisk](#)..... 193

To specify a file to stream a capture to and start the streaming process.

GetRecordedData

Graphical representation

Purpose	To read a capture result from file.
Description	<p>The GetRecordedData automation block lets you read a captured result from an IDF or MDF file created, for example, by the StartStreamToDisk block. It returns a CaptureResult object to be used, for example, at the AddPlotsToReport or AddCustomPlotsToReport blocks. The CaptureResult object can also be evaluated with the Evaluation library.</p> <p>Tip</p> <p>For an example how to use the GetRecordedData block, refer to the <i>XILAPIConvenienceExample</i> demo project. In the <i>Examples.StreamToDisk</i> project folder, you find the <i>StreamToDisk</i> sequence that uses the GetRecordedData block.</p>

Note

- The only IDF file format that can be read by the XIL API that is used by AutomationDesk is IDF version 5. Thus, you cannot read IDF version 6 files, which are written by ControlDesk 6.4 or earlier or by automation blocks that use ControlDesk 6.4 or earlier.
- For migration purposes, you can read IDF files that were generated with a dSPACE XIL API implementation up to and including version 2019-B.

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 GetSignalGroupNames block to dynamically get the value of the **SignalGroupName**.

Data objects

This automation block provides the following data objects:

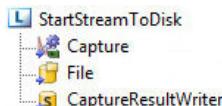
Name	In / Out	Type	Default Value	Description
File	In	File	None	
Vendor	In	String	"dSPACE"	<p>Specifies the vendor and version of the used XIL API implementation. With dSPACE specified as the vendor, the current version of the dSPACE XIL API implementation is used.</p> <p>To specify another version or another vendor, use the following format:</p> <pre><VendorName>_<ProductName>_<ProductVersion></pre> <p>Example:</p> <pre>dSPACE GmbH_XIL API_2019-A</pre>

Name	In / Out	Type	Default Value	Description
CaptureResultReader	In	String	MDF	<p>It is recommended to use a globally managed Vendor data object as a reference providing the vendor name. For more information, refer to Vendor (Data Object) on page 160.</p> <p>Lets you select the file type. The following file types are supported:</p> <ul style="list-style-type: none"> ▪ IDF (only for migration purposes; will be discontinued in later AutomationDesk versions) ▪ MDF (represents MDF version 4 (MF4) files) ▪ MDF40 (only HIL API; using XIL API blocks, the <i>MDF40</i> format is automatically mapped to the <i>MDF</i> format)
CaptureResult	Out	CaptureResult (Data Object)	None	

Related topics**References**

AddCustomPlotsToReport.....	172
AddPlotsToReport.....	173
Evaluation (AutomationDesk Basic Practices 	
StartStreamToDisk.....	193

StartStreamToDisk

Graphical representation**Purpose**

To write the fetched data directly to a specified file.

Description

The StartStreamToDisk automation block lets you specify an MDF file to stream a capture to. The streaming process is started during the block's execution.

Note

Before this block is used a capture has to be initialized by an InitializeCapture block earlier in the sequence.

The capture has to be started by using a StartCapture block later in the sequence.

You can stop the capture by placing a StopCapture block later in the sequence.

Tip

For an example how to use the StartStreamToDisk block, refer to the following sequence in the demo project:
XILAPIConvenienceExample.Examples.StreamToDisk.StreamToDisk.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Represents the configuration of a data capture returned by an InitializeCapture block earlier in the sequence.
File	In	File	None	
CaptureResultWriter	In	String	MDF	Lets you select the file type. The following file types are supported: <ul style="list-style-type: none"> ▪ MDF (represents MDF version 4 (MF4) files) ▪ MDF40 (only HIL API; using XIL API blocks, the <i>MDF40</i> format is automatically mapped to the <i>MDF</i> format)

Related topics**Basics**

[Accessing Simulation Platforms via XIL API.....](#) 117

References

InitializeCapture.....	186
StopCapture.....	190

SignalGenerator

Introduction

The SignalGenerator folder of the XIL API Convenience library provides convenient automation blocks you can use to configure and generate stimulus signals.

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

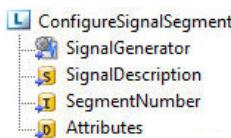
[ConfigureSignalSegment.....](#) 195

To modify attributes of a signal segment contained in a signal description.

DestroyOnTarget	196
To remove the generator from hardware.	
InitializeSignalGenerator	197
To initialize a signal generator to stimulate model variables of the real-time application running on the platform.	
LoadToTarget	200
To download the stimulus to the target.	
SaveSignalGenerator	201
To save a modified signal description set to file.	
StartSignalGenerator	202
To start a stimulus which has already been downloaded.	
ReleaseSignalGenerator	203
To release a signal generator instance.	

ConfigureSignalSegment

Graphical representation



Purpose

To modify attributes of a signal segment contained in a signal description.

Description

The ConfigureSignalSegment automation block lets you modify attributes of a signal segment contained in a signal description. You can specify signal segments via their index in the signal description collection.

Note

Before using this block, you must have initialized a signal generator via an InitializeSignalGenerator block.

Note

The instantiated SignalGenerator object and automation blocks using it have to be executed in the same execution thread.

Tip

For an example how to configure signal segments, refer to the following sequence in the demo project:
XILAPICConvenienceExample.Examples.SignalGenerator.SignalGenerator_ConfigureSegment.

Data objects

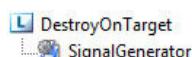
This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
SignalGenerator	In	SignalGenerator (Data Object)	None	Holds the configured and initialized signal generator.
SignalDescription	In	String	None	Lets you specify the name of the signal description the signal segment is contained in.
SegmentNumber	In	Int	0	Lets you specify by index which segment of the signal description you want to modify.
Attributes	In	Dictionary	{}	Lets you specify which attribute(s) of the segment (specified in the SegmentNumber data object) you want to modify. Specify each modification as a key-value pair. For example, to modify the attributes of a RAMP segment you have to specify: {'Start': 10.0, 'Stop': 15.0}.

Related topics**References**

DestroyOnTarget.....	196
InitializeSignalGenerator.....	197
SaveSignalGenerator.....	201

DestroyOnTarget

Graphical representation**Purpose**

To remove the generator from hardware.

Description

This block is used to remove the signal generator from the simulator platform.

If you remove a signal generator that is already initialized but not downloaded yet, the exception that occurs due to the ASAM standard is handled internally.

The execution is not interrupted, and so the block behavior stays the same as in previous library versions. The message is not displayed in the Output Viewer or in the Message pane. However, it is logged in dSPACE Log.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
SignalGenerator	In	SignalGenerator (Data Object)	None	Contains the instantiated SignalGenerator data object.

Access via Exec block

The SignalGenerator data object provides the possibility to use the block's functionality within an Exec block.

```
_AD_.SignalGenerator.DestroyOnTarget()
```

Related topics**Basics**

Accessing Simulation Platforms via XIL API	117
--	-----

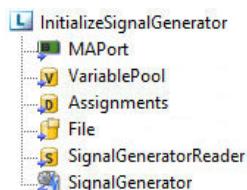
HowTos

How to Release Resources after Variable Stimulation	112
---	-----

References

SignalGenerator (Data Object)	254
---	-----

InitializeSignalGenerator

Graphical representation**Purpose**

To initialize a signal generator to stimulate model variables of the real-time application running on the platform.

Description

The InitSignalGenerator automation block lets you initialize a signal generator by loading a signal description set from an STZ or STI file created by programs such as the Signal Editor in AutomationDesk or ControlDesk.

The STZ file is a ZIP file that contains the signal descriptions in STI format. The STZ file can also contain additional MAT files to describe numerical signal data. You can assign the symbol names contained in the STZ file to variable names in AutomationDesk.

Note

Before this block is used the platform's model access port (MAPort) must have been initialized. To initialize the MAPort, use the InitMAPort block. If no XIL API Framework is initialized, release the MAPort by using the ReleaseMAPort block at the end of your sequence. If an XIL API Framework is initialized, the port is released automatically when the framework is shut down.

Note

The instantiated SignalGenerator object and automation blocks using it have to be executed in the same execution thread.

Tip

For an example how to use a signal generator, refer to the following sequence in the demo project:
XILAPIConvenienceExample.Examples.SignalGenerator.SignalGenerator_and_Capture.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Lets you specify the platform to connect to. This data object auto-references a data object of the same type at a higher hierarchy level.
VariablePool	In	Variant	None	The behavior depends on the XIL API layer you work with: <ul style="list-style-type: none"> ▪ XIL API Framework: This data object is not needed. The variables that can be used at the block are specified via the variable mapping in the framework configuration. ▪ XIL API Testbench: Lets you specify the set of variables that can be used at the block.

Name	In / Out	Type	Default Value	Description
				<p>Three different methods:</p> <ul style="list-style-type: none"> ▪ Create an XIL API Mapping data object in your project and reference it. The contained Alias values are interpreted as variable names and the associated Identifier values as the model paths. ▪ (For compatibility reasons only) Create <i>one</i> variable pool data container and reference it at the block's VariablePool data object. <p>Specify each variable contained in the pool as a String data object in the data container:</p> <ul style="list-style-type: none"> ▪ <VariableName> the name of the String data object ▪ <ModelPath> the value of the String data object ▪ (For compatibility reasons only) Create a dictionary and reference it. The value of the dictionary is interpreted as <VariableName>:<ModelPath>. <p>It is recommended to use the XIL API Mapping data object.</p>
Assignments	In	Dictionary	Ø	<p>The behavior depends on the XIL API layer you work with:</p> <ul style="list-style-type: none"> ▪ XIL API Framework: Lets you assign the SignalDescription name from the STZ or STI file to aliases from the variable mapping of the framework. ▪ XIL API Testbench: Lets you assign the SignalDescription name from the STZ or STI file to aliases specified in the project's variable pool. <p>Specify each assignment as a key-value pair, i.e., <SignalDescriptionName>:<VariableName>. Assignments from the STZ or STI file are ignored.</p>
File	In	File	None	
SignalGeneratorReader	In	String	STZ	Lets you select the file type. The following file types are supported: <ul style="list-style-type: none"> ▪ STZ ▪ STI
SignalGenerator	In	SignalGenerator (Data Object)	None	Returns the initialized SignalGenerator object.

Related topics

Basics

[Accessing Simulation Platforms via XIL API.....](#) 117

HowTos

[How to Stimulate Simulator Variables.....](#) 108

References

DestroyOnTarget.....	196
InitMAPort.....	162
LoadToTarget.....	200
ReleaseMAPort.....	167
StartSignalGenerator.....	202

LoadToTarget

Graphical representation

Purpose

To download the stimulus to the target.

Description

Prepares the HIL System for a stimulus run (comparable to a declaration of a stimulus on the HIL system). In general, this methods will load the stimulus down to the target. After downloading, the stimulus can be started.

Note

It is not possible to handle more than one LoadToTarget block within one thread. The DestroyOnTarget block must be executed before you can use the LoadToTarget block again.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
SignalGenerator	In	SignalGenerator (Data Object)	None	Contains the instantiated SignalGenerator data object.

Access via Exec block

Additionally, you can load a SignalGenerator data object to the target via Exec block.

```
_AD_.SignalGenerator.LoadToTarget()
```

Related topics
Basics

[Accessing Simulation Platforms via XIL API](#)..... 117

HowTos

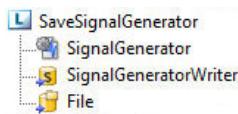
[How to Stimulate Simulator Variables](#)..... 108

References

DestroyOnTarget	196
SignalGenerator (Data Object)	254

SaveSignalGenerator

Graphical representation



Purpose

To save a modified signal description set to file.

Description

The SaveSignalGenerator automation block lets you save a signal generator to an STZ or STI file, for example, after modifying it via a ConfigureSignalSegment block.

The STZ file is a ZIP file that contains the signal descriptions in STI format. The STZ file can also contain additional MAT files to describe numerical signal data. This allows you to export signal generators modified in automation sequences to other programs such as the Signal Editor in AutomationDesk or ControlDesk.

Note

Before using this block, you must have initialized a signal generator via an InitializeSignalGenerator block.

Note

The instantiated SignalGenerator object and automation blocks using it have to be executed in the same execution thread.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
SignalGenerator	In	SignalGenerator (Data Object)	{}	Holds the configured and initialized signal generator.
SignalGeneratorWriter	In	String	STZ	Lets you specify the type of the file used for the signal description set. You can specify STI or STZ.
File	In/Out	File	None	Lets you specify the path and file name of an STZ or STI file to write a signal description set to.

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

ConfigureSignalSegment	195
InitializeSignalGenerator	197

StartSignalGenerator

Graphical representation**Purpose**

To start a stimulus which has already been downloaded.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
SignalGenerator	In	SignalGenerator (Data Object)	None	Contains the instantiated SignalGenerator data object.

Access via Exec block

Additionally, you start a signal generator via Exec block.

```
_AD_.SignalGenerator.Start()
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

HowTos

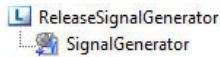
[How to Stimulate Simulator Variables](#)..... 108

References

[SignalGenerator \(Data Object\)](#)..... 254

ReleaseSignalGenerator

Graphical representation



Purpose

To release a signal generator instance.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
SignalGenerator	In	SignalGenerator (Data Object)	None	Contains the instantiated SignalGenerator data object.

Access via Exec block

Additionally, you can release a SignalGenerator data object via Exec block.

```
_AD_.SignalGenerator.Release()
```

Related topics

Basics

[Accessing Simulation Platforms via XIL API.....](#) 117

HowTos

[How to Release Resources after Variable Stimulation.....](#) 112

References

[SignalGenerator \(Data Object\).....](#) 254

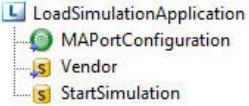
SimulationApplication

Introduction

The SimulationApplication folder of the XIL API Convenience library provides convenient automation blocks you can use to manage the simulation application running on your platform.

Where to go from here	Information in this section
	<p>LoadSimulationApplication..... 204 To load a simulation application to a platform.</p>
	<p>StartSimulation..... 205 To restart a simulation that was stopped.</p>
	<p>StopSimulation..... 206 To stop a running simulation.</p>

LoadSimulationApplication

Graphical representation	
Purpose	To load a simulation application to a platform.
Description	<p>This block is used to load a simulation application. The application file and the platform are specified in the MAPortConfiguration data object.</p> <p>If you use the dSPACE MAPortConfiguration, the currently running application is stopped and the configured application is loaded. If you use a vendor-specific MAPortConfiguration, the behavior of loading an application depends on the specified forceConfig attribute.</p> <p>The simulation application to be loaded is defined by either the system description file (SDF), or the platform-specific executable file (PPC, X86, RTA or OSA). For further information, refer to Basics on Simulation Applications on page 22.</p>
Data objects	This automation block provides the following data objects:

Name	In/Out	Data Type	Default Value	Description
MAPortConfiguration	In		{'ApplicationPath': '', 'PlatformIdentifier': ''}	<p>The behavior depends on the XIL API layer you work with:</p> <ul style="list-style-type: none"> ▪ XIL API Framework: Lets you select the model access port name from the list of ports that are configured in the XIL API Framework.

Name	In/Out	Data Type	Default Value	Description
Vendor	In	String	"dSPACE"	<ul style="list-style-type: none"> XIL API Testbench: Specifies the platform to load the simulation application to. This data object auto-references a data object of the same type at a higher hierarchy level. <p>Lets you select the vendor and version of the used XIL API implementation from the list of installed implementations in the Select vendor dialog.</p> <p>If dSPACE is specified as the vendor, the dSPACE XIL API implementation version that matches the active AutomationDesk version is used.</p> <p>To specify another version or another vendor, the following format is used:</p> <pre><VendorName>_<ProductName>_<ProductVersion></pre> <p>Example:</p> <pre>dSPACE_GmbH_XIL_API_2019-A</pre> <p>It is recommended to use a globally managed Vendor data object as a reference providing the vendor name. For more information, refer to Vendor (Data Object) on page 160.</p>
StartSimulation	In/Out	String	"TRUE"	<p>Specifies whether the application is to be started after the download.</p> <ul style="list-style-type: none"> TRUE: The application is automatically started after download. FALSE: The application is in stopped state after download. To start the application, use the StartSimulation block.

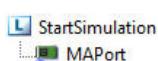
Related topics**HowTos**

How to Download and Start a Simulation Application	81
--	----

References

MAPortConfiguration (Data Object)	289
StartSimulation	205

StartSimulation

Graphical representation**Purpose**

To restart a simulation that was stopped.

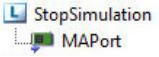
Description	If you are working with a DS1007, a MicroLabBox, a MicroAutoBox III, a SCALEXIO platform, or with VEOS, you can use this block to start a simulation that is in Stopped state. You can stop a started simulation by using the StopSimulation block. The simulation to be started is specified by the MAPort object configured and initialized via the InitMAPort block. If you start a running application, the exception that is raised due to the ASAM standard is handled internally. The execution is not interrupted and so the block behavior stays the same as in previous library versions.
--------------------	--

Data objects	This automation block provides the following data object:
---------------------	---

Name	In/Out	Data Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Specifies the platform to connect to.

Related topics	References								
	<table> <tr> <td>InitMAPort.....</td> <td>162</td> </tr> <tr> <td>LoadSimulationApplication.....</td> <td>204</td> </tr> <tr> <td>MAPort (Data Object).....</td> <td>286</td> </tr> <tr> <td>StopSimulation.....</td> <td>206</td> </tr> </table>	InitMAPort	162	LoadSimulationApplication	204	MAPort (Data Object)	286	StopSimulation	206
InitMAPort	162								
LoadSimulationApplication	204								
MAPort (Data Object)	286								
StopSimulation	206								

StopSimulation

Graphical representation	
Purpose	To stop a running simulation.
Description	If you are working with a DS1007, a MicroLabBox, a MicroAutoBox III, a SCALEXIO platform, or with VEOS, you can use this block to stop a simulation that is in Running state. You can restart a stopped simulation by using the StartSimulation block. The simulation to be stopped is specified by the MAPort object configured and initialized via the InitMAPort block.

If you stop a stopped application, the exception that is raised due to the ASAM standard is handled internally. The execution is not interrupted and so the block behavior stays the same as in previous library versions.

Data objects

This automation block provides the following data object:

Name	In/Out	Data Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Specifies the platform to connect to.

Related topics**References**

InitMAPort.....	162
LoadSimulationApplication.....	204
MAPort (Data Object).....	286
StartSimulation.....	205

XIL API (Model Access)

Introduction**Note**

For executing automation blocks of the XIL API library, the *Platform API Package* is required.

For further information, refer to [Packaging of AutomationDesk \(AutomationDesk Introduction And Overview\)](#).

The XIL API library provides a subset of the functionality of the *ASAM Generic Simulator Interface* to AutomationDesk required for the platform access and electrical error simulation. It supports ASAM AE XIL API version 2.1.0.

Since dSPACE Release 2016-B, the ASAM AE HIL API version 1.x is no longer supported.

For information on which parts of the ASAM AE XIL you can work in AutomationDesk's XIL API library, refer to [Accessing Simulation Platforms via XIL API](#) on page 117.

You can enhance a block's functionality by using the ASAM XIL API methods and properties in an Exec block.

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>.

Where to go from here

Information in this section

[Main Elements](#)..... 208

Provides data objects that represent the base objects of the ASAM XIL API.

[Common](#)..... 213

Provides selected subpackages of the ASAM XIL API common package.

[MAPort](#)..... 285

Provides data objects and automation blocks to access the model access port (MAPort).

Information in other sections

[XIL API \(Electrical Error Simulation\) \(AutomationDesk Simulating Electrical Errors\)](#)

Description of the specific blocks and data objects for electrical error simulation using the XIL library.

Main Elements

Introduction

The top level of the XIL API library provides data objects that represent the base objects of the ASAM XIL API to initialize and configure a Testbench instance. If you use the automation blocks from the XIL API library, these data objects are not required.

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

[Mapping \(Data Object\)](#)..... 209

To map aliases to the model paths of variables.

[PortConfig \(Data Object\)](#)..... 211

To provide the base configuration for all the XIL API ports.

[Testbench \(Data Object\)](#)..... 211

To instantiate a test bench as the common object for the ports used.

[TestbenchFactory \(Data Object\)](#)..... 212

To create a vendor-specific testbench.

Information in other sections

Further automation blocks for working with an XIL API Framework:

[CheckValues](#)..... 156

To evaluate a condition that contains variable aliases.

[GetValues](#)..... 157

To read and report variable values.

[SetValues](#)..... 158

To write and report variable values.

Mapping (Data Object)

Graphical representation

- If you work with the XIL API Framework, i.e., the framework is initialized:



- If you work with the XIL API Testbench, i.e., no framework is initialized:

**Purpose**

To map aliases to the model paths of variables.

Description

If you work with the XIL API Framework, the variable mapping is configured centrally across all projects in the framework configuration and the project-specific Mapping data objects are obsolete. However, they can remain in the projects that are migrated to use the XIL API Framework. Refer to [How to Migrate Projects to Use XIL API Framework](#) on page 70.

If you work with the XIL API Testbench, the Mapping data object lets you use aliases instead of model paths to increase the readability of variable handling.

You can edit single entries of a Mapping data object via the **Mapping Editor**, and you can change its contents as a whole via **Export XIL API Mapping** and **Import XIL API Mapping**.

Double-clicking the Mapping data object displays its contents in the **Mapping Viewer** pane for reading. From here you can drag the entries, for example, to add signals to a signal description set.

Note

- You can create only one Mapping data object per AutomationDesk project.
- The Mapping data object must reside at the project hierarchy level.

Access via Exec block

The Mapping data object can be accessed via an Exec block. The member listing feature of the Python Editor provides the available methods and properties.

The following methods are provided:

- Export (<Path>, <ConfirmOverwrite=True|False>)
- Import(Path)

The **LabelMapping** property provides read access to the contents of the Mapping data object.

The following example shows how to access a Mapping data object named **Mapping** to get the **Identifier** via a script:

```
# Show the value of a specified column of this entry:  
# _AD_.<DOBName>.LabelMapping.<Alias>.<Alias|Identifier|  
#   VariableType|Description>  
print (_AD_.Mapping.LabelMapping.BatteryVoltage.Identifier)
```

Related topics

Basics

[Accessing Simulation Platforms via XIL API](#)..... 117

HowTos

[How to Make Simulator Variables Available](#)..... 84

References

Export XIL API Mapping	361
Import XIL API Mapping	362
Mapping Editor (AutomationDesk Basic Practices)	

PortConfig (Data Object)

Graphical representation



Purpose

To provide the base configuration for all the XIL API ports.

Description

The PortConfig data object provides the base configuration for a formerly instantiated MAPort or EESPort object.

You must use this data object only, if you want to create and configure an MAPort or EESPort via a Testbench instance. The PortConfig data object provides the port-specific configurations after using the **LoadConfiguration** method of the corresponding port.

Access via Exec block

The PortConfig data object can be accessed within an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

The following example shows how to set the values of a PortConfig instance via script.

```
VendorSpecificConfiguration = "MyVendorConfigName"
# MAPortConfig
_AD_.PortConfig = \
    _AD_.MAPort.LoadConfiguration(r"C:\MAPortConfiguration.xml")
print (_AD_.PortConfig.ModelFile)
_AD_.MAPort.Configure(_AD_.PortConfig, True)
# EESPortConfig
_AD_.PortConfig = \
    _AD_.EESPort.LoadConfiguration(r"C:\EESPortConfiguration.xml")
_AD_.EESPort.Configure(_AD_.PortConfig, True)
```

Related topics

Basics

[Accessing Simulation Platforms via XIL API.....](#) 117

Testbench (Data Object)

Graphical representation



Purpose	To instantiate a testbench as the common object for the ports used.
Description	The Testbench data object is used as a wrapper for the Testbench class. It lets you read information given in the testbench configuration, such as the ASAM XIL API version, the name of the vendor-specific implementation and the available port types. If you do not use the default XIL API implementation by dSPACE, the vendor-specific information must be specified when you initialize the TestbenchFactory data object.
Access via Exec block	The Testbench data object can be accessed within an Exec block. The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf .
Related topics	<p>Basics</p> <p>Accessing Simulation Platforms via XIL API..... 117</p> <p>References</p> <p>PortConfig (Data Object)..... 211 TestbenchFactory (Data Object)..... 212</p>

TestbenchFactory (Data Object)

Graphical representation	 TestbenchFactory
Purpose	To create a vendor-specific testbench.
Description	The TestbenchFactory data object is used to instantiate a testbench based on the vendor-specific port configuration.

Access via Exec block

The following example shows how to initialize a TestbenchFactory data object and instantiate a Testbench data object.

```
_AD_.TestbenchFactory.InitTestbenchFactory()
_AD_.Testbench = \
    _AD_.TestbenchFactory.CreateVendorSpecificTestbench(
        "dSPACE GmbH", "XIL", "2.1.0")
```

Related topics**Basics**

Accessing Simulation Platforms via XIL API	117
--	-----

References

PortConfig (Data Object)	211
Testbench (Data Object)	211

Common

Introduction

The Common folder provides selected subpackages of the ASAM XIL API common package.

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

Duration	214
The Duration folder provides data objects for handling durations that differ in unit and data type.	
MetalInfo	216
The MetalInfo folder provides data objects to access the meta information of the instantiated testbench.	
Symbol	219
Provides data objects for using symbols.	
CaptureResult	222
Provides a data object and automation blocks to obtain the measured data as results.	
Capturing	230
Provides data objects and automation blocks to acquire data in a continuous data stream.	

Script	250
Provides data objects for handling scripts that are executed on real-time hardware or VEOS.	
SignalGenerator	253
Provides data objects and automation blocks for generating stimulus signals.	
Signal	266
Provides data objects for stimulus signal handling.	
ValueContainer	273
Provides data objects and an automation block to gain and deliver data.	
WatcherHandling	279
Provides data objects and automation blocks, for example, to define the trigger definitions of captures.	

Duration

Introduction

The Duration folder provides data objects for handling durations.

Where to go from here

Information in this section

Duration (Data Object)	214
To provide a duration as a time span or a number of cycles.	
DurationFactory (Data Object)	216
To instantiate a factory object to create a duration.	

Duration (Data Object)

Graphical representation



Duration

Purpose

To provide a duration as a time span or a number of cycles.

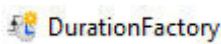
Description	<p>The Duration data object is used to specify durations as a time span or a number of simulation cycles.</p> <p>The instantiated Duration data object can be accessed by using the methods and properties from the ASAM AE XIL standard in an Exec block. For example, you can use the Value property to set the value of a Duration data object.</p> <p>The Duration data object must be initialized by using the InitDuration method, see below. To release the data object, you have to initialize it with None or use the Release method.</p> <p>This element is supported only by XIL API 2.1.0.</p>
Access via Exec block	<p>The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf.</p> <p>The xilapi Python module lets you work with the Duration data object within an Exec block. The following example shows how to initialize Duration objects:</p> <pre>_AD_.DurationInSeconds = xilapi.InitDuration(_AD_.Vendor, # vendor = "dSPACE" 'TimeSpanDuration', # duration class name 12.3) # time span in seconds) _AD_.DurationInNumOfCycles = xilapi.InitDuration(_AD_.Vendor, # vendor = "dSPACE" 'CycleNumberDuration', # duration class name 3) # time span in cycles)</pre> <p>Alternatively, you can initialize the duration according to the ASAM standard:</p> <pre># instantiate the duration factory object _Ad_.DurationFactory = _AD_.Testbench.GetDurationFactory() # create a time span duration of 12.3 seconds _Ad_.DurationInSeconds = _AD_.DurationFactory.CreateTimeSpanDuration(12.3) # create a duration of three simulation cycles _Ad_.DurationInCycles = _AD_.DurationFactory.CreateCycleNumberDuration(3)</pre>

Related topics**References**

DurationFactory (Data Object)	216
---	-----

DurationFactory (Data Object)

Graphical representation



DurationFactory

Purpose

To instantiate a factory object to create a duration.

Description

The DurationFactory data object is used to instantiate a Duration data object that lets you specify durations as time spans or as a number of simulation cycles.

This element is supported only by XIL API 2.1.0.

Access via Exec block

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

The following example shows how to initialize a DurationFactory data object.

```
_AD_.DurationFactory = _AD_.Testbench.GetDurationFactory()
```

Related topics

References

Duration (Data Object)	214
--	-----

MetaInfo

Introduction

The MetaInfo folder provides data objects to access the meta information of the instantiated testbench.

Where to go from here

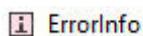
Information in this section

ErrorInfo (Data Object)	217
To provide error information.	
TaskInfo (Data Object)	217
To get information on a testbench's port task.	

VariableInfo (Data Object).....	218
To get information on a testbench's port variable.	

ErrorInfo (Data Object)

Graphical representation



Purpose

To provide error information.

Description

The ErrorInfo data object is used to provide testbench error information in cases where exceptions are not applicable.

This element is supported only by XIL API 2.1.0.

Access via Exec block

The ErrorInfo data object can be accessed within an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

If you set the XIL API library to offline operation mode, you can provide the required values via the related `Get` methods. Otherwise, the default values are used.

The following example shows how to get the task name from a running simulation application.

```
_AD_.FirstErrorInfo = _AD_.MAPort.ErrorInfos[0]
_AD_.FirstTaskName = _AD_.FirstErrorInfo.Name
```

The following example shows how to get the task name when you run the test in offline operation mode.

```
_AD_FirstErrorInfo = _AD_.MAPort.ErrorInfos[0]
_AD_.FirstTaskName = _AD_.ErrorInfo.GetName("Task5ms")
```

TaskInfo (Data Object)

Graphical representation



Purpose	To get information on a testbench's port task.
Description	The TaskInfo data object is used to read the available information on a task that is assigned to a testbench's port, for example, a capture task on a model access port.
Access via Exec block	<p>The TaskInfo data object can be accessed within an Exec block.</p> <p>The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf.</p> <p>If you set the XIL API library to offline operation mode, you can provide the required values via the related Get methods. Otherwise, the default values are used.</p> <p>The following example shows how to get the task name from a running simulation application.</p> <pre>_AD_.FirstTaskInfo = _AD_.MAPort.TaskInfos[0] _AD_.FirstTaskName = _AD_.FirstTaskInfo.Name</pre> <p>The following example shows how to get the task name when you run the test in offline operation mode.</p> <pre>_AD_FirstTaskInfo = _AD_.MAPort.TaskInfos[0] _AD_.FirstTaskName = _AD_.TaskInfo.GetName("Task5ms")</pre>

Related topics	Basics Accessing Simulation Platforms via XIL API..... 117 References VariableInfo (Data Object)..... 218
-----------------------	--

VariableInfo (Data Object)

Graphical representation  VariableInfo

Purpose	To get information on a testbench's port variable.
----------------	--

Description	The VariableInfo data object is used to read the available information that is assigned to a testbench's port, for example, an MAPortVariableInfo value.
--------------------	--

Access via Exec block	<p>The VariableInfo data object can be accessed within an Exec block.</p> <p>The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf.</p> <p>If you set the XIL API library to offline operation mode, you can provide the required values via the related <code>Get</code> methods. Otherwise, the default values are used.</p> <p>The following example shows how to get the VariableInfo data type from a running simulation application.</p>
------------------------------	---

```
_AD_.VariableInfo = \
    _AD_.MAPort.GetVariableInfo(
        r"Model Root/BatteryVoltage[V]/Value")
print(xilapi.Common.ValueContainer.Enum.DataType.eFLOAT == \
    _AD_.VariableInfo.DataType)
```

The following example shows how to get the VariableInfo data type when you run the test in offline operation mode.

```
OfflineDataType = \
    xilapi.Common.ValueContainer.Enum.DataType.eFLOAT
print(xilapi.Common.ValueContainer.Enum.DataType.eFLOAT == \
    _AD_.VariableInfo.Get(DataType(OfflineDataType)))
```

Related topics	<p>Basics</p> <table> <tr> <td>Accessing Simulation Platforms via XIL API.....</td><td>117</td></tr> </table> <p>References</p> <table> <tr> <td>TaskInfo (Data Object).....</td><td>217</td></tr> </table>	Accessing Simulation Platforms via XIL API	117	TaskInfo (Data Object)	217
Accessing Simulation Platforms via XIL API	117				
TaskInfo (Data Object)	217				

Symbol

Introduction	The Symbol folder provides data objects for instantiating a Symbol data object.
---------------------	---

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

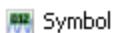
[Symbol \(Data Object\)](#)..... 220

To provide a symbol.

[SymbolFactory \(Data Object\)](#)..... 221

To instantiate a factory object to create a symbol.

Symbol (Data Object)

Graphical representation**Purpose**

To provide a symbol.

Description

The Symbol data object is used to provide a placeholder for a string or a constant value. For example string symbols are usually used for model paths, const symbols are usually used for parameters or segment durations.

The instantiated Symbol data object can be accessed by using the methods and properties from the ASAM AE XIL standard in an Exec block. For example, you can use the **Value** property to set the value of a Symbol data object.

The Symbol data object must be initialized by using the **InitSymbol** method, see below. To release the data object you have to initialize it with **None** or use the **Release** method.

Access via Exec block

The Python module **xilapi** provides the possibility to work with the Symbol data object within an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

The following example shows how to instantiate a Symbol data object as a const symbol with the value 4.5.

```
_AD_.ConstSymbol = xilapi.InitSymbol(
    _AD_.Vendor,      # vendor, for example "dSPACE"
    "ConstSymbol",   # SymbolType: StringSymbol, SignalSymbol, ConstSymbol
    4.5)             # value
```

Related topics

Basics

Accessing Simulation Platforms via XIL API.....	117
---	-----

SymbolFactory (Data Object)

Graphical representation**Purpose**

To instantiate a factory object to create a symbol.

Description

The SymbolFactory data object is used to instantiate a Symbol data object. It must only be used if you have created your testbench starting with the TestbenchFactory object according to the ASAM XIL API standard. By using the `xilapi` module or the corresponding automation blocks, this data object is not required.

Access via Exec block

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

The following example shows how to initialize a SymbolFactory data object and instantiate a Symbol data object.

```
_AD_.SymbolFactory = _AD_.Testbench.SymbolFactory
_AD_.MyConstSymbol = _AD_.SymbolFactory.CreateConstSymbol()
```

Related topics

Basics

Accessing Simulation Platforms via XIL API.....	117
---	-----

References

Symbol (Data Object).....	220
---	-----

CaptureResult

Introduction	Provides a data object and automation blocks to obtain the measured data as results.
---------------------	--

Where to go from here	Information in this section
	CaptureEvent (Data Object) 222 To provide an event during data capturing.
	CaptureResult (Data Object) 223 To get the result of an XIL API measurement.
	ExtractSignalValue 225 To return a SignalValue specified by Variable and SignalGroupName.
	GetMetaData 226 To return the meta data information of the capture result.
	GetSignalGroupNames 227 Returns the list of identifiers (task/raster names) which are used to access the SignalGroupValue instances.
	GetSignalGroupValue 228 To return a SignalGroupValue specified by its identifier.
	SetMetaData 229 To set the meta data.

CaptureEvent (Data Object)

Graphical representation



Purpose

To provide an event during data capturing.

Description

CaptureEvent data objects are used to get information on the events that occurred during a data capture on a platform, i.e., that are contained in a capture result. You can get the time stamp and event type of each event.

Some types of events are triggered by the XIL API, for example, at frame start and at frame stop.

You can also trigger custom events explicitly via the `TriggerClientEvent` method that is provided by `Capture` data objects.

The instantiated `CaptureEvent` data object can be accessed by using the methods and properties from the ASAM AE XIL standard in an Exec block.

This element is supported only by XIL API 2.1.0.

Access via Exec block

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

The following example shows how to get the time stamp of the first event in a capture result:

```
_AD_.FirstCaptureEvent = _AD_.CaptureResult.GetEvents()[0]
_AD_.FirstTimeStamp = _AD_.FirstCaptureEvent.TimeStamp
```

The following example shows how to get the task name when you run the test in offline operation mode.

```
_AD_.FirstCaptureEvent = _AD_.CaptureResult.GetEvents()[0]
_AD_.FirstTimeStamp = _AD_.FirstCaptureEvent.getTimeStamp()
```

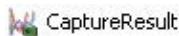
Related topics

References

Capture (Data Object).....	231
----------------------------	-----

CaptureResult (Data Object)

Graphical representation



Purpose

To get the result of an XIL API measurement.

Description	Contains one time axis and at least one measurement of a variable. The complete structure can be stored in a file.
--------------------	--

Note

You must initialize the CaptureResult data object with `None` or use the `Release` method in an Exec block to free the allocated memory of the CaptureResult instance (XIL API class instance) which is stored in the CaptureResult data object.

`_AD_.CaptureResult = None` or `_AD_CaptureResult.Release()`

Alternatively, you can use a ClearValues block to minimize the required memory.

Access via Exec block

The data object can be accessed within an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

The following example shows how to initialize an empty CaptureResult object.

```
_AD_.CaptureResult = xilapi.InitCaptureResult(_AD_.Vendor)
```

A CaptureResult object is automatically initialized by using the `GetCaptureResult` method of a Capture object, refer to [GetCaptureResult](#) on page 239.

Related topics**HowTos**

[How to Capture Data.....](#) 96

References

[Clear Value \(AutomationDesk Basic Practices\)](#)

[ClearValues \(AutomationDesk Basic Practices\)](#)

ExtractSignalValue

Graphical representation



Purpose

To return a SignalValue specified by Variable and SignalGroupName.

Description

The SignalValue contains the data of one measured variable. To get a list of the signal group names contained in a CaptureResult data object, use the GetSignalGroupNames block. To get the data of a complete signal group use the GetSignalGroupValue block.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
CaptureResult	In	CaptureResult (Data Object)	None	Contains the instantiated CaptureResult data object.
SignalGroupName	In	String	""	Contains the name of the task/raster, e.g. "5ms". On multicore or multiprocessor systems, the task/raster is prefixed by the application name, for example, masterappl/HostService .
Variable	In	String	""	Contains the name of the variable, e.g. 'Model Root/Gain/nMot'. On multicore or multiprocessor systems, the variable begins with the application name, for example, masterappl/Model Root/.... The variable must be contained in the signal group specified in SignalGroupName.
SignalValue	Out	SignalValue (Data Object)	None	Contains the instantiated SignalValue data object.

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 GetSignalGroupNames block to dynamically get the value of the **SignalGroupName**.

Access via Exec block

The block's functionality can be used via script in an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object.

```
_AD_.SignalValue = _AD_.CaptureResult.ExtractSignalValue(  
    SignalGroupName, VariableName)
```

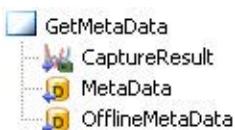
Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

CaptureResult (Data Object).....	223
GetSignalGroupNames.....	227
SignalValue (Data Object).....	277

GetMetaData

Graphical representation**Purpose**

To return the meta data information of the capture result.

Description

The meta data contains additional information on, such as, true or false trigger conditions. Measurement data is not part of the meta data.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
CaptureResult	In	CaptureResult (Data Object)	None	Contains the instantiated CaptureResult data object.
MetaData	Out	Dictionary	{}	Returns the meta data as an associative list.
OfflineMetaData	In	Dictionary	{}	Lets you specify the meta data to be used in offline operation mode.

Access via Exec block

The block's functionality can be used via script in an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object.

```
_AD_.Dictionary = _AD_.CaptureResult.GetMetaData()
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

CaptureResult (Data Object)	223
SetMetaData	229

GetSignalGroupNames

Graphical representation**Purpose**

Returns the list of identifiers (task/raster names) which are used to access the SignalGroupName instances.

Description

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 GetSignalGroupNames block to dynamically get the value of the **SignalGroupName**.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
CaptureResult	In	CaptureResult (Data Object)	None	Contains the instantiated CaptureResult data object.
SignalGroup\ Names	Out	List	[]	Returns a list of signal group names.

Access via Exec block

The block's functionality can be used via script in an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object.

```
_AD_.List = _AD_.CaptureResult.GetSignalGroupNames()
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

[CaptureResult \(Data Object\)](#)..... 223

GetSignalGroupValue

Graphical representation**Purpose**

To return a SignalGroupValue specified by its identifier.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
CaptureResult	In	CaptureResult (Data Object)	None	Contains the instantiated CaptureResult data object.
Identifier	In	String	""	Contains the task name (e.g., 5 ms). You can get available task names with the GetSignalGroupNames on page 227 block.
SignalGroup\\Value	Out	SignalGroupValue (Data Object)	None	Contains the instantiated SignalGroupValue data object.

Access via Exec block

The block's functionality can be used via script in an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object.

```
FirstSignalGroupName = \
    _AD_.CaptureResult.GetSignalGroupNames()[0]
.SignalGroupValue = \
    _AD_.CaptureResult.GetSignalGroupValue(FirstSignalGroupName)
```

Related topics

Basics

[Accessing Simulation Platforms via XIL API](#)..... 117

References

CaptureResult (Data Object)	223
SignalGroupValue (Data Object)	276

SetMetaData

Graphical representation



Purpose

To set the meta data.

Description

The meta data contains additional information on, such as, true or false trigger conditions. Measurement data is not part of the meta data.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
CaptureResult	In	CaptureResult (Data Object)	None	Contains the instantiated CaptureResult data object.
MetaData	In	Dictionary	{}	Sets the meta data as an associative dictionary.

Access via Exec block

The block's functionality can be used via script in an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object.

```
_AD_.CaptureResult.SetMetaData(_AD_.Dictionary)
```

Related topics

Basics

[Accessing Simulation Platforms via XIL API](#)..... 117

References

[CaptureResult \(Data Object\)](#)..... 223
[GetMetaData](#)..... 226

Capturing

Introduction

The Capturing folder provides data objects and automation blocks to acquire data in a continuous data stream.

Where to go from here

Information in this section

[Capture \(Data Object\)](#)..... 231

Represents the configuration of a measurement, for example, data capturing.

[CaptureResultReader \(Data Object\)](#)..... 232

To provide a file reader for a capture result.

[CaptureResultWriter \(Data Object\)](#)..... 233

To provide a file writer for a capture result.

[CaptureState \(Data Object\)](#)..... 234

To provide a group of states.

[CapturingFactory \(Data Object\)](#)..... 235

To instantiate a factory object to create capture-related objects.

[ClearConfiguration](#)..... 236

To delete every set configuration.

[Fetch](#)..... 237

To catch measurement results.

[GetCaptureResult](#)..... 239

To return all measured data.

GetMinBufferSize	240
To return the minimum buffer size (in byte) of the real-time service.	
GetState	241
To return the current state of the CaptureObject.	
GetVariables	242
To return the variables selected for capturing.	
ReleaseCapture	243
To release a capture instance.	
SetMinBufferSize	244
To set the buffer size (in byte) of the real-time service.	
SetStartTriggerCondition	245
To set the start trigger condition.	
SetStopTriggerCondition	246
To set the stop trigger condition.	
SetVariables	247
To set the variables for capturing.	
Start	248
To start data logging.	
Stop	249
To stop data logging.	

Capture (Data Object)

Graphical representation



Purpose

Represents the configuration of a measurement, for example, data capturing.

Description

The configuration contains the variables to be captured, the trigger conditions, the duration of the measurement and the real-time model task.

Note

You must free the allocated memory of the Capture instance (XIL API class instance) which is stored in the Capture data object. Use the following release method:

```
_AD_.Capture.Release()
```

Access via Exec block	The Capture data object can be accessed via script within an Exec block. The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf . The following example shows how to create a Capture object.
	<pre>_AD_.CaptureDataObject = self.MAPort.CreateCapture(TaskName)</pre>

Related topics	Basics
	Accessing Simulation Platforms via XIL API..... 117
	HowTos
	How to Capture Data..... 96

CaptureResultReader (Data Object)

Graphical representation	 CaptureResultReader
Purpose	To provide a file reader for a capture result.
Description	<p>The CaptureResultReader data object is used to offer an instantiated CaptureResultReader object to access a file that contains measured data, for example, an MDF file with the file name suffix MF4. To access other formats, you have to implement a reader by using the methods for a CaptureResultReader object specified in the ASAM standard.</p> <p>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.</p>

You can use the `GetSignalGroupNames` block to dynamically get the value of the `SignalGroupName`.

Note

- The only IDF file format that can be read by the XIL API that is used by AutomationDesk is IDF version 5. Thus, you cannot read IDF version 6 files, which are written by ControlDesk 6.4 or earlier or by automation blocks that use ControlDesk 6.4 or earlier.
- For migration purposes, you can read IDF files that were generated with a dSPACE XIL API implementation up to and including version 2019-B.

Access via Exec block

The Python module `xilapi` provides the possibility to work with the `CaptureResultReader` data object within an Exec block.

The following example shows how to instantiate a `CaptureResultReader` to read MDF files via script.

```
_AD_.CaptureResultReader = xilapi.InitCaptureResultReader(
    _AD_.Vendor, # vendor = "dSPACE"
    "CaptureResultMDFReader", # CaptureResultReaderType
    r"C:\CaptureResult.mf4") # the instance-specific
                            # capture result MDF file
```

For migration purposes, you can instantiate a `CaptureResultReader` to read IDF files (will be discontinued in later AutomationDesk versions).

```
_AD_.CaptureIDFResultReader = xilapi.InitCaptureResultReader(
    _AD_.Vendor, # vendor = "dSPACE"
    "CaptureResultIDFReader", # CaptureResultReaderType
    r"C:\CaptureResult.idf") # the instance-specific
                            # capture result IDF file
```

Related topics

Basics

[Accessing Simulation Platforms via XIL API](#)..... 117

References

[File Types \(ControlDesk User Interface Handling\)](#)

CaptureResultWriter (Data Object)

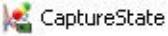
Graphical representation



Purpose	To provide a file writer for a capture result.
Description	If you want to use the CaptureResultWriter data object to save your capture result to an external file. Use the Start block to start capturing. To access specific formats, you have to implement a writer by using the methods for a CaptureResultWriter object specified in the ASAM standard.
Access via Exec block	<p>The Python module <code>xilapi</code> provides the possibility to work with the CaptureResultWriter data object within an Exec block.</p> <p>The following example shows how to instantiate a CaptureResultWriter to write to an MDF file with the file name suffix MF4 via script.</p> <pre>_AD_.CaptureResultWriter = xilapi.InitCaptureResultWriter(_AD_.Vendor, # vendor = "dSPACE" "CaptureResultMDFWriter", # CaptureResultWriterType r"C:\CaptureResult.mf4") # the instance-specific # capture result MDF file</pre>

Related topics	Basics
	Accessing Simulation Platforms via XIL API..... 117

CaptureState (Data Object)

Graphical representation	 CaptureState
Purpose	To provide a group of states.
Description	The CaptureState object can take any of the following enumeration types (Enums).

CaptureState	Description
eCONFIGURED	Initial state before capturing is started and after it is stopped.
eACTIVATED	Capturing is started and waits for a start trigger.

CaptureState	Description
eRUNNING	Measurement is running.
eFINISHED	Measurement is stopped via a stop trigger.

Tip

You can find a capturing state diagram ("state diagram of capturing") in the ASAM documentation [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

You can get information on the state also via:

```
print(_AD_.Capture.GetState().__doc__)
```

Tip

You have to import the `xilapi` module if you want to access the Enums by an Exec block.

The following example shows how to access the `eCONFIGURED` state.

```
print(_AD_.Capture.GetState() == \
xilapi.Common.Capturing.Enum.CaptureState.eCONFIGURED)
```

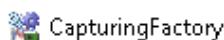
Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

[Edit \(CaptureState\)](#)..... 356
[Exec \(AutomationDesk Basic Practices\)](#)

CapturingFactory (Data Object)

Graphical representation**Purpose**

To instantiate a factory object to create capture-related objects.

Description

The CapturingFactory data object is used to instantiate a Capture object and further objects required for capturing.

Access via Exec block	This data object can be accessed via script within an Exec block. The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf .
------------------------------	---

The following example shows how to create a CapturingFactory object.

```
_AD_.CapturingFactory = _AD_.Testbench.CapturingFactory
```

Related topics	Basics <table border="1"> <tr> <td>Accessing Simulation Platforms via XIL API</td><td>117</td></tr> </table> References <table border="1"> <tr> <td>CaptureResult (Data Object)</td><td>223</td></tr> </table>	Accessing Simulation Platforms via XIL API	117	CaptureResult (Data Object)	223
Accessing Simulation Platforms via XIL API	117				
CaptureResult (Data Object)	223				

ClearConfiguration

Graphical representation	 ClearConfiguration  Capture
Purpose	To delete every set configuration.
Description	The ClearConfiguration automation block deletes all stored data and releases all resources.
Data objects	This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Contains the instantiated Capture data object.

Access via Exec block	This block can be accessed via script within an Exec block. The name completion feature of the Python Editor provides the available properties and methods of the current object.
	<pre>_AD_.Capture.ClearConfiguration()</pre>

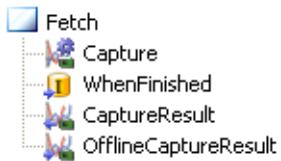
Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

[Capture \(Data Object\)](#)..... 231

Fetch

Graphical representation**Purpose**

To catch the measurement results.

Description

The Fetch automation block returns the measurement results that have been acquired for the last call of the block if no overflow occurred.

Note

- The Fetch automation block only delivers valid values if the CaptureState (Data Object) is either eACTIVATED or eRUNNING.
- Executing the Fetch automation block is not allowed if you have configured to write the capture result to a file.

Tip

You can find a capturing state diagram ("state diagram of capturing") in the ASAM documentation [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

You can get information on the state also via:

```
print(_AD_.Capture.GetState().__doc__)
```

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Contains the instantiated Capture data object.
WhenFinished	In	Int	0	<ul style="list-style-type: none"> ▪ True (nonzero) means that the further execution of the block will be blocked until the capture state has changed to eFinished (Data Object) on page 234). ▪ False (=0) means that the acquired data will be returned immediately after the block execution is finished.
				<p>Note</p> <p>If you set the WhenFinished parameter to True and ConditionWatchers are used whose trigger conditions has not been fulfilled, a Deadlock can occur. To avoid this set the timeout parameter of the ConditionWatcher (see InitConditionWatcher on page 281).</p>
CaptureResult	Out	CaptureResult (Data Object)	None	Returns the measurement result.
OfflineCaptureResult	In	CaptureResult (Data Object)	None	Lets you specify the OfflineCaptureResult to be used in offline operation mode.

Access via Exec block

The Python module xilapi provides the possibility to access the Fetch method within an Exec block.

```
_AD_.CaptureResult = _AD_.Capture.Fetch()
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

Capture (Data Object)	231
CaptureResult (Data Object)	223
CaptureState (Data Object)	234
GetCaptureResult	239

GetCaptureResult

Graphical representation



Purpose

To return all measured data.

Description

If no data is available an empty object is returned.

To get the capture result the capture state must be eCONFIGURED. Therefore a Stop block must be executed before you can get the capture result.

Tip

You can find a capturing state diagram ("state diagram of capturing") in the ASAM documentation [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

You can get information on the state also via:

```
print(_AD_.Capture.GetState().__doc__)
```

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Contains the instantiated Capture data object.
CaptureResult	Out	CaptureResult (Data Object)	None	Returns the complete data.
OfflineCapture\Result	In	CaptureResult (Data Object)	None	Lets you specify the CaptureResult to be used in offline operation mode.

Access via Exec block

Additionally, you can get a capture result via Exec block.

```
_AD_.CaptureResult = \
    _AD_.Capture.GetCaptureResult(OfflineCaptureResult)
# OfflineCaptureResult is optional
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

Capture (Data Object).....	231
CaptureResult (Data Object).....	223
CaptureState (Data Object).....	234
Stop.....	249

GetMinBufferSize

Graphical representation**Purpose**

To return the minimum buffer size (in byte) of the real-time service.

Note

This method is not supported by dSPACE.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Contains the instantiated Capture data object.
MinBufferSize	Out	Int	0	Returns the buffer size (in byte) of the real-time service.
OfflineMinBufferSize	In	Int	0	Lets you specify the MinBufferSize to be used in offline operation mode.

Access via Exec block

Additionally, you can get the specified buffer size via Exec block.

`_AD_.Capture.GetMinBufferSize()`

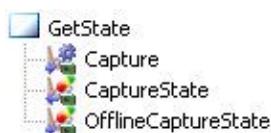
Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

Capture (Data Object)	231
SetMinBufferSize	244

GetState

Graphical representation**Purpose**

To return the current state of the CaptureObject.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Contains the instantiated Capture data object.
CaptureState	Out	CaptureState (Data Object)	eCONFIGURED	Returns the current state of the CaptureObject.
OfflineCapture\State	In	CaptureState (Data Object)	eCONFIGURED	Lets you specify the CaptureState to be used in offline operation mode.

Access via Exec block

Additionally, you can get the state via Exec block.

```

_AD_.CaptureState = _AD_.Capture.GetState(OfflineCaptureState)
# OfflineCaptureState is optional
print(_AD_.Capture.GetState() == \
      xilapi.Common.Capturing.Enum.CaptureState.eCONFIGURED)
  
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

Capture (Data Object)	231
CaptureState (Data Object)	234

GetVariables

Graphical representation**Purpose**

To return the variables which are selected for capturing.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Contains the instantiated Capture data object.
Variables	Out	List	[]	Returns the variables which are selected for capturing.
OfflineVariables	In	List	[]	Lets you specify the Variables to be used in offline operation mode.

Access via Exec block

The Python module xilapi provides the possibility to access the GetVariables method within an Exec block.

```

_AD_.VariableList = _AD_.Capture.GetVariables(OfflineVariables)
# OfflineVariables is optional
    
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API.....](#) 117

References

[Capture \(Data Object\).....](#) 231

ReleaseCapture

Graphical representation**Purpose**

To release a capture instance.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Contains the instantiated Capture data object.

Access via Exec block

The block's functionality can be used via script in an Exec block.

```
_AD_.Capture.Release()
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API.....](#) 117

References

[Capture \(Data Object\).....](#) 231

SetMinBufferSize

Graphical representation



Purpose

To set the buffer size (in byte) of the real-time service.

Note

This method is not supported by dSPACE.

Description

The buffer size has to ensure continuous measurement for the specified duration. By default, the complete available buffer is reserved.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Contains the instantiated Capture data object.
MinBufferSize	In	Int	0	Sets the buffer size (in byte) of the real-time service.

Access via Exec block

Additionally, you can set the specified buffer size via Exec block.

`_AD_.Capture.SetMinBufferSize()`

Related topics

Basics

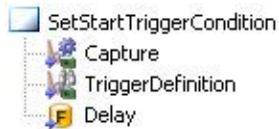
[Accessing Simulation Platforms via XIL API.....](#) 117

References

Capture (Data Object).....	231
GetMinBufferSize.....	240

SetStartTriggerCondition

Graphical representation



Purpose

To set the start trigger condition.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Contains the instantiated Capture data object.
TriggerDefinition	In	Watcher (Data Object)	None	Contains the instantiated Watcher data object. The watcher data object must be initialized before. The start trigger condition is defined via a ConditionWatcher object (see InitConditionWatcher on page 281).
Delay	In	Float	0.0	Sets the trigger delay in seconds. <ul style="list-style-type: none"> ▪ Positive start trigger: If the delay for the start trigger is positive, capturing starts after the specified amount of time passed since the start trigger became true. If no start trigger is specified, capturing starts after the specified amount of time passed since the Start block was executed. ▪ Negative start trigger: If the delay for the start trigger is negative, the capture result buffers the values for the specified time before the start trigger event occurs. A start trigger event is ignored if it occurs before the negative delay is reached for the first time.

Access via Exec block

Additionally, you can set the start trigger via Exec block.

```
_AD_.Capture.SetStartTriggerCondition(_AD_.ConditionWatcher, 0.0)
```

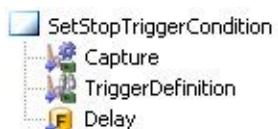
Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

Capture (Data Object)	231
Watcher (Data Object)	279

SetStopTriggerCondition

Graphical representation**Purpose**

To set the stop trigger condition.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Contains the instantiated Capture data object.
TriggerDefinition	In	Watcher (Data Object)	None	Contains the instantiated Watcher data object. The watcher data object must be initialized before. The stop trigger condition can be either via a ConditionWatcher or a DurationWatcher object (see InitConditionWatcher on page 281 and InitDurationWatcher on page 284).
Delay	In	Float	0.0	Sets the trigger delay in seconds. If the delay for the stop trigger is positive, capturing stops after the specified amount of time passed since the stop trigger occurred. A negative value is not supported and causes an error message.

Access via Exec block

Additionally, you can set the stop trigger via Exec block.

```
_AD_.Capture.SetStopTriggerCondition(_AD_.Watcher, 0.0)
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

Capture (Data Object)	231
Watcher (Data Object)	279

SetVariables

Graphical representation**Purpose**

Sets the variables for capturing.

Description

Only complete sets of variables can be set. Every time the SetVariables automation block is executed the variable list is cleared and reconfigured.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Contains the instantiated Capture data object.
Variables	In	List	[]	List of variables to be measured.

Access via Exec block

The Python module xilapi provides the possibility to access the SetVariables method within an Exec block.

```
_AD_.Capture.SetVariables(_AD_.VariableList)
```

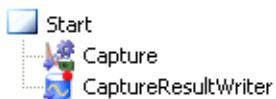
Related topics**Basics**

[Accessing Simulation Platforms via XIL API.....](#) 117

References

[Capture \(Data Object\).....](#) 231

Start

Graphical representation**Purpose**

To start data logging.

Description

This block is used to start capturing. The capture result is either written to a CaptureResult data object or to a file specified by the CaptureResultWriter data object. In the first case, the CaptureResultWriter data object must be set to *None* or to a CaptureResultMemoryWriter.

For an example of capturing and writing to a file, refer to the XIL API demo (TurnSignalTest-Blocks sequence in the TurnSignal project).

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Contains the instantiated Capture data object.
CaptureResultWriter	In	CaptureResultWriter (Data Object)	None	<p>Specifies where and in which format the capture result is written.</p> <ul style="list-style-type: none"> ▪ None: Capture result is written to the local memory of your PC. ▪ Instantiated CaptureResultWriter: Capture result is written to the file that is specified in the CaptureResultWriter data object.

Access via Exec block

The Capture data object provides the possibility to work with the Start method within an Exec block. The following example shows how to start a capture and writing the results to an MDF file via script.

```
# Initialize CaptureResultWriter
_AD_.CaptureResultWriter = xilapi.InitCaptureResultWriter(
    "dSPACE",                                # vendor
    "CaptureResultMDFWriter",    # CaptureResultWriterType
    r"C:\CaptureResult.mf4")      # capture result MDF file
# Start capturing to file
_AD_.Capture.Start(_AD_.CaptureResultWriter)
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

Capture (Data Object).....	231
SetStartTriggerCondition.....	245
SetStopTriggerCondition.....	246
Stop.....	249

Stop

Graphical representation**Purpose**

To stop data logging.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Capture	In	Capture (Data Object)	None	Contains the instantiated Capture data object.

Access via Exec block

The Capture data object provides the possibility to access the Stop method within an Exec block.

```
_AD_.Capture.Stop()
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

[Capture \(Data Object\)](#)..... 231
[Start](#)..... 248

Script

Introduction

The Script folder provides data objects for handling scripts that are executed on real-time hardware or VEOS synchronous to the real-time applications.

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

[Script \(Data Object\)](#)..... 250

To provide a script that can be executed on real-time hardware and VEOS.

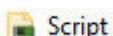
[TargetScriptFactory \(Data Object\)](#)..... 252

To instantiate a factory object for creating a reader for a target script.

[TargetScriptsFileReader \(Data Object\)](#) 252

To provide a file reader for a target script.

Script (Data Object)

Graphical representation**Purpose**

To provide a script that can be executed on real-time hardware and VEOS.

Description

The Script data object is used to provide a configuration and execution control interface for user-defined scripts that are executed on real-time hardware or VEOS. Scripts that run on dSPACE platforms are RTT sequences, i.e., Python code that uses dSPACE *Real-Time Testing* components. You can configure each RTT

sequence via an XML file with the same name as the sequence. You can use RTT sequences, for example, to monitor or stimulate variables synchronously to the real-time application.

The instantiated Script data object can be accessed by using the methods and properties from the ASAM AE XIL standard in an Exec block.

This element is supported only by XIL API 2.1.0.

Access via Exec block

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

The following example shows how to instantiate a Script data object, download the script, and start it.

```
# initialize a TargetScriptFileReader
_AD_.TargetScriptFactory = _AD_.Testbench.GetTargetScriptFactory()
_AD_.TargetScriptFileReader = \
    _AD_.TargetScriptFactory.CreateTargetScriptFileReaderByFileName(
        File)
# instantiate an empty script data object
_AD_Script = _AD_.MAPort.CreateTargetScript()
# initialize it with the file reader object
_AD_.Script.Load(_AD_.TargetScriptFileReader)
# load it to simulator and start it
_AD_.Script.LoadToTarget()
_AD_.Script.Start()
```

Tip

For an example of how to use a target script, refer to the following sequence in the XIL API demo project:
TurnSignal.TurnSignalTest.TargetScript_Examples.TargetScriptExample.

Related topics

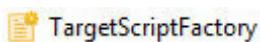
Basics

[Real-Time Testing Guide](#)

References

Real-Time Testing Library Reference	
TargetScriptFactory (Data Object).....	252
TargetScriptsFileReader (Data Object)	252

TargetScriptFactory (Data Object)

Graphical representation**Purpose**

To instantiate a factory object for creating a reader for a target script.

Description

The TargetScriptFactory data object is used to instantiate a reader data object for a script to be executed on a target, i.e., on real-time hardware or VEOS.

The factory data object must only be used if you have created a testbench starting with the TestbenchFactory object according to the ASAM XIL API standard.

By using the `xilapi` module, this data object is not required.

This element is supported only by XIL API 2.1.0.

Access via Exec block

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

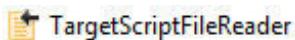
The following example shows how to initialize a TargetScriptFactory data object.

```
_AD_.TargetScriptFactory = _AD_.Testbench.GetTargetScriptFactory()
```

Related topics**References**

Script (Data Object).....	250
TargetScriptsFileReader (Data Object)	252

TargetScriptsFileReader (Data Object)

Graphical representation**Purpose**

To provide a file reader for a target script.

Description	<p>The TargetScriptsFileReader data object is used to offer an instantiated TargetScriptsFileReader object to access a file that contains a script that can be executed on real-time hardware or VEOS. The supported language of the script is vendor-specific. The dSPACE XIL API implementation supports Python scripts.</p> <p>To access other formats, you have to implement a reader by using the methods for a TargetScriptsFileReader object specified in the ASAM standard.</p> <p>This element is supported only by XIL API 2.1.0.</p>
Access via Exec block	<p>The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf.</p> <p>The <code>xilapi</code> Python module lets you work with the TargetScriptsFileReader data object within an Exec block. The following example shows how to instantiate a TargetScriptsFileReader:</p> <pre>_AD_.TargetScriptsFileReader = xilapi.InitTargetScriptsFileReaderByFileName(_AD_.Vendor, # vendor = "dSPACE" "C:\TargetScript.py") # the Python target script)</pre> <p>You can also instantiate the reader according to the ASAM standard:</p> <pre>_AD_.TargetScriptFactory = _AD_.Testbench.GetTargetScriptFactory() _AD_.TargetScriptFileReader = \ _AD_.TargetScriptFactory.CreateTargetScriptFileReaderByFileName(File)</pre>

Related topics**References**

Script (Data Object).....	250
TargetScriptFactory (Data Object).....	252

SignalGenerator

Introduction

The SignalGenerator folder provides data objects and automation blocks for generating stimulus signals.

Tip

For an example how to use the data objects and automation blocks, see also the XILAPISimulus demo project.

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

SignalGenerator (Data Object)	254
To get a signal description set for stimulating variables of a real-time application.	
SignalGeneratorFactory (Data Object)	256
To instantiate a factory object to create a signal generator reader or writer.	
SignalGeneratorReader (Data Object)	257
To get all Reader implementations in the SignalGenerator context.	
SignalGeneratorWriter (Data Object)	257
To get a writer implementation in the SignalGenerator context.	
DestroyOnTarget	258
To remove the generator from hardware.	
InitSignalGeneratorSTZReader	259
To create a signal generator STZ reader.	
InitSignalGeneratorSTZWriter	260
To create a signal generator STZ writer.	
LoadSignalGenerator	261
To load the signal generator reader.	
LoadToTarget	262
To download the stimulus to the target.	
ReleaseSignalGenerator	263
To release a capture instance.	
SaveSignalGenerator	264
To save the signal generator writer.	
SetAssignments	264
To set assignments to the signal generator.	
StartSignalGenerator	265
To start a stimulus which has already been downloaded.	

SignalGenerator (Data Object)

Graphical representation**Purpose**

To get a signal description set for stimulating variables of a real-time application.

Description	<p>The SignalGenerator data object is used as a wrapper for the SignalGenerator class and contains the SignalGenerator instance. You can use a SignalGenerator data object to stimulate the model variables of a real-time application running on connected real-time hardware. The STZ file of a SignalGenerator data object contains the signal descriptions and optional information about the signal mapping, variable descriptions, and the assigned real-time hardware.</p> <p>The instantiated SignalGenerator data object can be accessed by using the methods and properties from the XIL API standard in an Exec block.</p> <p>In offline execution mode, the SignalGenerator data object is only passed as a parameter.</p> <p>To free the allocated memory of the SignalGenerator instance (XIL API class instance) which is stored in the SignalGenerator data object, you have to use the ReleaseSignalGenerator block.</p> <p>Using stimulus signal on multiprocessor and multicore systems When you use a multiprocessor or multicore system, you have to specify the name of the member application on which the stimulus is to be generated within a Dictionary data object. The key that represents the member application is ApplicationProcessor. The contents of the Dictionary data object can be applied to the SignalGenerator via the CustomProperties property. It is not possible to access only one key-value pair of the dictionary.</p> <p>For example, the following script is to be used in an Exec block.</p> <pre>customDict = _AD_.SignalGenerator.CustomProperties customDict["ApplicationProcessor"] = "masterAppl" _AD_.SignalGenerator.CustomProperties = customDict</pre>
Access via Exec block	<p>The SignalGenerator data object can be accessed within an Exec block.</p> <p>The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf.</p> <p>The following example shows how to initialize a SignalGenerator instance via script. The vendor argument is optional.</p> <pre>_AD_.SignalGenerator = _AD_.MAPort.CreateSignalGenerator() ... # To release a SignalGenerator _AD_.SignalGenerator.Release()</pre>

Related topics**Basics**

[Accessing Simulation Platforms via XIL API.....](#) 117

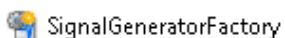
HowTos

[How to Stimulate Simulator Variables.....](#) 108

References

[CreateSignalGenerator.....](#) 291

SignalGeneratorFactory (Data Object)

Graphical representation**Purpose**

To instantiate a factory object to create a signal generator reader or writer.

Description

The SignalGeneratorFactory data object is used to instantiate a SignalGeneratorReader or SignalGeneratorWriter object.

Access via Exec block

This data object can be accessed via script within an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

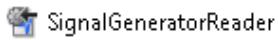
The following example shows how to initialize a SignalGeneratorReader to read an STZ file created by AutomationDesk's Signal Editor.

```
_AD_.SignalGeneratorFactory = \
    _AD_.Testbench.SignalGeneratorFactory
    _AD_.SignalGeneratorReader = \
        _AD_.SignalGeneratorFactory.CreateSignalGeneratorSTZReader()
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API.....](#) 117

SignalGeneratorReader (Data Object)

Graphical representation

Purpose

To get a reader implementation in the SignalGenerator context to load a SignalGenerator signal.

Description

If you use the `InitSignalGeneratorSTZReader` block, you instantiates a signal generator reader that reads its stimulus signal from an STZ file. For accessing an STI file, you have to instantiate the `SignalGeneratorReader` data object via script in an Exec block.

Access via Exec block

The Python module `xilapi` provides the possibility to create a `SignalGeneratorReader` data object within an Exec block.

The following example shows how to initialize a `SignalGeneratorReader` for reading an STI file.

```
_AD_.SignalGeneratorReader = xilapi.InitSignalGeneratorReader(
    _AD_.Vendor,           # vendor
    # SignalGeneratorReaderType: SignalGeneratorSTIReader|SignalGeneratorSTZReader
    "SignalGeneratorSTIReader",
    r"C:\Signal.sti")      # STI file to write to
```

Related topics
Basics

[Accessing Simulation Platforms via XIL API.....](#) 117

SignalGeneratorWriter (Data Object)

Graphical representation

Purpose

To get a writer implementation in the SignalGenerator context to save the SignalGenerator signal to a file.

Description

If you use the `InitSignalGeneratorSTZWriter` block, you instantiates a signal generator writer that writes its stimulus signal to an STZ file. For accessing an STI

file, you have to instantiate the SignalGeneratorWriter data object via script in an Exec block.

Access via Exec block

The Python module `xilapi` provides the possibility to create a SignalGeneratorWriter data object within an Exec block. The following example shows how to initialize a SignalGeneratorWriter for writing to an STI file.

```
_AD_.SignalGeneratorWriter = xilapi.InitSignalGeneratorWriter(
    _AD_.Vendor,           # vendor
    "SignalGeneratorSTIWriter", # SignalGeneratorWriterType
    r"C:\Signal.sti")       # STZ file to write to
```

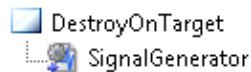
Related topics

Basics

[Accessing Simulation Platforms via XIL API.....](#) 117

DestroyOnTarget

Graphical representation



Purpose

To remove the generator from hardware.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
SignalGenerator	In	SignalGenerator (Data Object)	None	Contains the instantiated SignalGenerator data object.

Access via Exec block

The SignalGenerator data object provides the possibility to use the block's functionality within an Exec block.

```
_AD_.SignalGenerator.DestroyOnTarget()
```

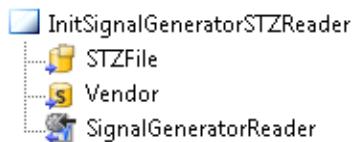
Related topics**Basics**

[Accessing Simulation Platforms via XIL API.....](#) 117

References

[SignalGenerator \(Data Object\).....](#) 254

InitSignalGeneratorSTZReader

Graphical representation**Purpose**

To create a signal generator STZ reader.

Description

An STZ file contains a signal description set. It can be created, for example, by using the Signal Editor from AutomationDesk or ControlDesk.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
STZFile	In	File	None	Contains the path and the file name of the STZ file to be loaded.
Vendor	In	String	"dSPACE"	<p>Specifies the vendor and version of the used XIL API implementation.</p> <p>With dSPACE specified as the vendor, the current version of the dSPACE XIL API implementation is used.</p> <p>To specify another version or another vendor, use the following format:</p> <pre><VendorName>_<ProductName>_<ProductVersion></pre> <p>Example:</p> <pre>dSPACE_GmbH_XIL_API_2019-A</pre> <p>It is recommended to use a globally managed Vendor data object as a reference providing the vendor name. For more information, refer to Vendor (Data Object) on page 160.</p>
SignalGeneratorReader	Out	SignalGeneratorReader (Data Object)	None	Contains the created SignalGeneratorReader object.

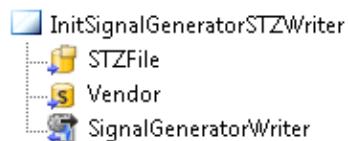
Access via Exec block

Alternatively, you can initialize the SignalGeneratorReader data object via Exec block. For further information, refer to [SignalGeneratorReader \(Data Object\)](#) on page 257.

Related topics**References**

File (AutomationDesk Basic Practices)	257
SignalGeneratorReader (Data Object)	

InitSignalGeneratorSTZWriter

Graphical representation**Purpose**

To create a signal generator STZ writer.

Description

An STZ file contains a signal description set. It can be created, for example, by using the Signal Editor from AutomationDesk or ControlDesk.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
STZFile	In	File	None	Contains the path and the file name of the STZ file to be saved.
Vendor	In	String	"dSPACE"	<p>Specifies the vendor and version of the used XIL API implementation.</p> <p>With dSPACE specified as the vendor, the current version of the dSPACE XIL API implementation is used.</p> <p>To specify another version or another vendor, use the following format:</p> <pre><VendorName>_<ProductName>_<ProductVersion></pre> <p>Example:</p> <pre>dSPACE GmbH_XIL API_2019-A</pre>

Name	In / Out	Type	Default Value	Description
SignalGeneratorWriter	Out	SignalGeneratorWriter (Data Object)	None	<p>It is recommended to use a globally managed Vendor data object as a reference providing the vendor name. For more information, refer to Vendor (Data Object) on page 160.</p> <p>Contains the created SignalGeneratorWriter object.</p>

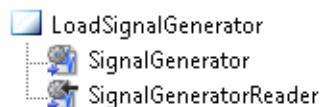
Access via Exec block

Alternatively, you can initialize the SignalGeneratorWriter data object via Exec block. For further information, refer to [SignalGeneratorWriter \(Data Object\)](#) on page 257.

Related topics**References**

- | | | |
|--|-------|-----|
| File (AutomationDesk Basic Practices  | | 257 |
| SignalGeneratorWriter (Data Object) | | |

LoadSignalGenerator

Graphical representation**Purpose**

To load the signal generator reader.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
SignalGenerator	In	SignalGenerator (Data Object)	None	Contains the instantiated SignalGenerator data object.
SignalGeneratorReader	In	SignalGeneratorReader (Data Object)	None	Contains the created SignalGeneratorReader object.

Access via Exec block

Additionally, you can load a SignalGenerator data object via Exec block.

<code>_AD_.SignalGenerator.Load(_AD_.SignalGeneratorReader)</code>
--

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

SignalGenerator (Data Object)	254
SignalGeneratorReader (Data Object)	257

LoadToTarget

Graphical representation**Purpose**

To download the stimulus to the target.

Description

Prepares the HIL System for a stimulus run (comparable to a declaration of a stimulus on the HIL system). In general, this methods will load the stimulus down to the target. After downloading, the stimulus can be started.

Note

It is not possible to handle more than one LoadToTarget block within one thread. The DestroyOnTarget block must be executed before you can use the LoadToTarget block again.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
SignalGenerator	In	SignalGenerator (Data Object)	None	Contains the instantiated SignalGenerator data object.

Access via Exec block

Additionally, you can load a SignalGenerator data object to the target via Exec block.

```
_AD_.SignalGenerator.LoadToTarget()
```

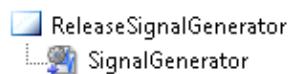
Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

DestroyOnTarget	258
SignalGenerator (Data Object)	254

ReleaseSignalGenerator

Graphical representation**Purpose**

To release a signal generator instance.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
SignalGenerator	In	SignalGenerator (Data Object)	None	Contains the instantiated SignalGenerator data object.

Access via Exec block

Additionally, you can release a SignalGenerator data object via Exec block.

```
_AD_.SignalGenerator.Release()
```

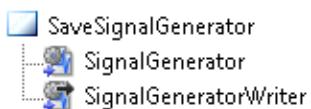
Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

SignalGenerator (Data Object)	254
---	-----

SaveSignalGenerator

Graphical representation

Purpose

To save the signal generator writer.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
SignalGenerator	In	SignalGenerator (Data Object)	None	Contains the instantiated SignalGenerator data object.
SignalGeneratorWriter	In	SignalGeneratorWriter (Data Object)	None	Contains the created SignalGeneratorWriter object.

Access via Exec block

Additionally, you can save the signal generator writer via Exec block.

```
_AD_.SignalGenerator.Save(_AD_.SignalGeneratorWriter)
```

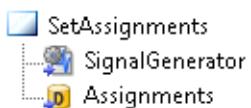
Related topics
Basics

[Accessing Simulation Platforms via XIL API](#)..... 117

References

SignalGenerator (Data Object)	254
SignalGeneratorWriter (Data Object)	257

SetAssignments

Graphical representation

Purpose

To set assignments to the signal generator.

Description	This automation block is used to assign model variables and specific platform information to the signal description set.
--------------------	--

Data objects	This automation block provides the following data objects:
---------------------	--

Name	In / Out	Type	Default Value	Description
SignalGenerator	In	SignalGenerator (Data Object)	None	Contains the instantiated SignalGenerator data object.
Assignments	Out	Dictionary	{}	Contains the assignments to be set in a Dictionary data object. {<Keys>:<Value>} ▪ Keys: Signal name out of the used signal description set. ▪ Value: Path to the variable.

Access via Exec block	Additionally, you can set the assignments via Exec block.
------------------------------	---

```
_AD_.SignalGenerator.SetAssignments(
    {"Signal1": "ModelRoot\TurnSignal\Param"})
```

Tip

See also the XILAPISimulus demo project.

Related topics

References

Dictionary (AutomationDesk Basic Practices 	254
SignalGenerator (Data Object).....		

StartSignalGenerator

Graphical representation



Purpose

To start a stimulus which has already been downloaded.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
SignalGenerator	In	SignalGenerator (Data Object)	None	Contains the instantiated SignalGenerator data object.

Access via Exec block

Additionally, you start a signal generator via Exec block.

```
_AD_.SignalGenerator.Start()
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

[SignalGenerator \(Data Object\)](#)..... 254

Signal

Introduction

The Signal folder provides data objects for the stimulus signal handling.

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

[ScriptParameterInfo \(Data Object\)](#)..... 267

To get information on a target script's parameters.

[SignalDescriptionSet \(Data Object\)](#)..... 267

To provide a signal description set.

[SignalFactory \(Data Object\)](#)..... 268

To instantiate a factory object to create a signal or segment.

[SignalSegment \(Data Object\)](#)..... 269

To provide a signal segment.

[SignalDescription \(Data Object\)](#)..... 270

To provide a signal description.

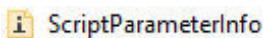
[SignalDescriptionSetReader \(Data Object\)](#)..... 271

To provide a signal description reader.

SignalDescriptionSetWriter (Data Object)	272
To provide a signal description writer.	

ScriptParameterInfo (Data Object)

Graphical representation



Purpose

To get information on a target script's parameters.

Description

The ScriptParameterInfo data object is used to get the available information on the parameters of a script that is assigned to a testbench's port.

This element is supported only by XIL API 2.1.0.

Access via Exec block

The ScriptParameterInfo data object can be accessed within an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

The following example shows how to get the information of the first parameter of a target script.

```
_AD_.FirstScriptParameterInfo = _AD_.Script.ScriptParameterInfos()[0]
_AD_.FirstTaskName = _AD_.FirstScriptParameterInfo.Name
```

Related topics

References

Script (Data Object)	250
--------------------------------------	-----

SignalDescriptionSet (Data Object)

Graphical representation



Purpose	To provide a signal description set.
Description	<p>Signal description sets are used to edit and arrange signals. The signal description set is a container for one or more signals. When you save the project, each signal description set is saved to an STZ file. The STZ file is a ZIP file that contains the signal descriptions in STI format. The STZ file can also contain additional MAT files to describe numerical signal data.</p> <p>A signal description set that you want to use for stimulating variables of a real-time application, must be configured as a SignalGenerator data object beforehand.</p> <p>You can also use a SignalDescriptionSet data object as the input for the Evaluation library's GetSignalFromXILAPISignalValue block.</p> <p>In offline execution mode, the SignalDescriptionSet data object is only passed as a parameter.</p> <p>The SignalDescriptionSet data object can be released by initializing it with None or calling the Release method.</p>
Access via Exec block	<p>This data object can be accessed via script within an Exec block.</p> <p>The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf.</p> <pre>_AD_.SignalDescriptionSet1 = xilapi.InitSignalDescriptionSet(_AD_.Vendor, # vendor = "dSPACE" _AD_.SignalDescriptionSetReader) # SignalDescriptionSetReader object prints(_AD_.SignalDescriptionSet1.Count())</pre>

Related topics**References**

SignalGenerator (Data Object).....	254
------------------------------------	-----

SignalFactory (Data Object)

Graphical representation**Purpose**

To instantiate a factory object to create a signal or segment.

Description	The SignalFactory data object is used to instantiate a Signal object. You can build a signal with the same segment types as available in AutomationDesk's Signal Editor. In addition, you can create signal description sets and reader and writer for signal description sets.
--------------------	---

Access via Exec block	This data object can be accessed via script within an Exec block.
------------------------------	---

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

For further information on the available signal types, refer to [Signal Editor \(AutomationDesk Implementing Signal-Based Tests\)](#).

The following example shows how to create a Const signal via script.

```
_AD_.SignalFactory = _AD_.Testbench.SignalFactory
_AD_.ConstSignal = _AD_.SignalFactory.CreateConstSignal()
```

Related topics	Basics
	Accessing Simulation Platforms via XIL API 117

SignalSegment (Data Object)

Graphical representation	 SignalSegment
Purpose	To provide a signal segment.
Description	<p>A SignalSegment data object is used to provide a specific segment of a signal. Each segment must be configured with a duration in seconds, other properties depend on the selected segment type. The segment properties are based on the symbolic mapping. They can be therefore specified by a numerical value or another signal, for example, for specifying the amplitude of a sine segment.</p> <p>In offline execution mode, the SignalSegment data object is only passed as a parameter.</p> <p>The SignalSegment data object must be initialized by using the <code>InitSignalSegment</code> method, see below. You have to initialize the data object with <code>None</code> to release it or use the <code>Release</code> method.</p>

For limitations, refer to [Limitations When Using the XIL API Library](#) on page 371.

Access via Exec block

This data object can be accessed via script within an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

The following example shows how to instantiate a SignalSegment data object as a constant signal segment.

```
_AD_.ConstSegment = xilapi.InitSignalSegment(  
    _AD_.Vendor,          # vendor = "dSPACE"  
    "ConstSegment",       # SignalSegmentClassName  
    _AD_.DurationSymbol, # instance-specific arguments  
    _AD_.ValueSymbol)
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

[Symbol \(Data Object\)](#)..... 220

SignalDescription (Data Object)

Graphical representation**Purpose**

To provide a signal description.

Description

The SignalDescription data object is used to provide a signal description object for reading or writing. It contains the description of one signal (consisting of signal segments) without information on the hardware or the model to be used.

A SignalDescription data object can be configured as SegmentSignalDescription or OperationSignalDescription.

The SignalDescription data object must be initialized by using the **InitSignalDescription** method, see below. You have to initialize the data object with None to release it or use the **Release** method.

Access via Exec block

This data object can be accessed via script within an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

The following example shows how to instantiate a SignalDescription data object and to access a segment by index.

```
_AD_.SegmentSignalDescription = xilapi.InitSignalDescription(
    _AD_.Vendor,                      # vendor = "dSPACE"
    "SegmentSignalDescription")       # SignalDescriptionClassName
    _AD_.SignalSegment = _AD_.SegmentSignalDescription.GetByIndex(0)
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API.....](#) 117

SignalDescriptionSetReader (Data Object)

Graphical representation**Purpose**

To provide a signal description reader.

Description

The SignalDescriptionSetReader data object is used to provide an object to read from a signal description.

The instantiated SignalDescriptionSetReader data object can be accessed by using the methods and properties from the ASAM AE HIL/XIL standard in an Exec block. For example, you can use the **Load** method to load a SignalDescriptionSet object.

The SignalDescriptionSetReader data object must be initialized by using the **InitSignalDescriptionSetReader** method, see below. You have to initialize the data object with None to release it or use the **Release** method.

Access via Exec block

This data object can be accessed via script within an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

The following example shows how to instantiate a SignalDescriptionSetReader data object to read from the specified STZ file.

```
_AD_.SignalDescriptionSetReader = xilapi.InitSignalDescriptionSetReader(
    _AD_.Vendor,                                # vendor = "dSPACE"
    "SignalDescriptionSetSTZReader", # SignalDescriptionSetReaderType (STI or STZ)
    r"C:\Signal.stz")                         # instance-specific argument
```

Related topics**Basics**

Accessing Simulation Platforms via XIL API	117
--	-----

References

SignalDescriptionSet (Data Object)	267
SignalDescriptionSetWriter (Data Object)	272

SignalDescriptionSetWriter (Data Object)

Graphical representation**Purpose**

To provide a signal description writer.

Description

The SignalDescriptionSetWriter data object is used to write to a signal description.

The instantiated SignalDescriptionSetWriter data object can be accessed by using the methods and properties from the ASAM AE HIL/XIL standard in an Exec block. For example, you can use the **Save** method to save a modified SignalDescriptionSet object.

The SignalDescriptionSetWriter data object must be initialized by using the **InitSignalDescriptionSetWriter** method, see below. You have to initialize the data object with **None** to release it or use the **Release** method.

Access via Exec block

The Python module **xilapi** provides the possibility to work with the SignalDescriptionSetWriter data object within an Exec block. The following example shows how to instantiate a SignalDescriptionSetWriter data object to write to the specified STZ file.

```
_AD_.SignalDescriptionSetWriter = xilapi.InitSignalDescriptionSetWriter(
    _AD_.Vendor,                                # vendor = "dSPACE"
    "SignalDescriptionSetSTZWriter", # SignalDescriptionSetWriterType (STI or STZ)
    r"C:\Signal.stz")                         # instance-specific argument
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

SignalDescriptionSet (Data Object)	267
SignalDescriptionSetReader (Data Object)	271

ValueContainer

Introduction

The ValueContainer folder provides data objects and an automation block to gain and deliver data.

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

[Attributes \(Data Object\)](#)..... 273

To provide a value's attributes.

[BaseValue \(Data Object\)](#)..... 274

Represents the base class for all value containers.

[DataType \(Data Object\)](#)..... 275

To set a data type instance to initialize automation blocks.

[SignalGroupValue \(Data Object\)](#)..... 276

Represents a collection of all values of a signal group.

[SignalValue \(Data Object\)](#)..... 277

Represents the value of a signal with its associated time stamp.

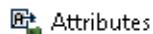
[ValueFactory \(Data Object\)](#)..... 277

To instantiate a factory object to create a value.

[InitBaseValue](#)..... 278

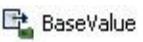
To initialize a base value data object.

Attributes (Data Object)

Graphical representation

Purpose	To provide a value's attributes.
Description	The Attributes data object is used to provide additional information for an instantiated value. Each information consists of an attribute name and its value. Some attributes are predefined, for example, the Unit attribute for the unit used. You can add user-defined attributes. For each attribute, you can read and write its value.
Access via Exec block	This data object can be accessed via script within an Exec block. The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf .
	The following example shows how to access an Attributes instance via script. <pre>_AD_.Float = xilapi.InitBaseValue(_AD_.Vendor, "eFLOAT", 1.1) _Ad_.Attributes = _AD_.Float.GetAttributes() print(_AD_.Attributes.GetUnit())</pre>
Related topics	Basics Accessing Simulation Platforms via XIL API 117

BaseValue (Data Object)

Graphical representation	 BaseValue
Purpose	Represents the base class for all value containers.
Description	A BaseValue data object is able to hold physical values, for example, for a parameter. Several interfaces are derived from the BaseValue data object for the representation of scalar, array, matrix values and others.

Note

You must initialize the BaseValue data object with *None* in an Exec block to free the allocated memory of the BaseValue instance (HIL/XIL API class instance) which is stored in the BaseValue data object.

```
_AD_.BaseValue = None or _AD_.BaseValue.Release()
```

Access via Exec block

This data object can be accessed via script within an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

The following example shows how to instantiate an integer and a float value.

```
_AD_.IntBaseValue = xilapi.InitBaseValue(
    _AD_.Vendor,      # vendor = "dSPACE"
    "eINT",          # DataType enumeration
    5)                # value
_Ad_.FloatBaseValue = xilapi.InitBaseValue(
    _AD_.Vendor,      # vendor = "dSPACE"
    "eFLOAT",         # DataType enumeration
    5.5)              # value
...
# To release the BaseValue objects
_Ad_.IntBaseValue.Release()
_Ad_.FloatBaseValue.Release()
```

Related topics**Basics**

Accessing Simulation Platforms via XIL API.....	117
---	-----

Data Type (Data Object)

Graphical representation**Purpose**

To set a data type instance to initialize automation blocks.

Description	You can set the data type to different enumeration types (Enums) via a drop down list.
--------------------	--

Tip

You have to import the `xilapi` module if you want to access the Enums by an Exec block.

Example:

```
print(_AD_.Float.Type == \
      xilapi.Common.ValueContainer.Enum.DataType.eFLOAT)
```

Related topics**Basics**

Accessing Simulation Platforms via XIL API.....	117
---	-----

References

Edit (DataType).....	357
--------------------------------------	-----

SignalGroupValue (Data Object)

Graphical representation**Purpose**

Represents a collection of all values of a signal group.

Description

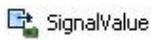
The SignalGroupValue represents all captured data of a capture. The timestamps are included too. It is composed by an X vector (which represents, for example, the time axis), an array of Y vectors (which holds, for example, FloatVectorValues). The X vector and all Y vectors must have the same length.

To free the allocated memory set the data object to `None` or use the `Release` method.

Related topics**Basics**

Accessing Simulation Platforms via XIL API.....	117
---	-----

SignalValue (Data Object)

Graphical representation

Purpose

Represents the value of a signal with its associated time stamp.

Description

To free the allocated memory set the data object to **None** or use the **Release** method.

Related topics

Basics

[Accessing Simulation Platforms via XIL API](#)..... 117

ValueFactory (Data Object)

Graphical representation

Purpose

To instantiate a factory object to create a value.

Description

The ValueFactory data object is used to instantiate and configure a Value object.

Access via Exec block

This data object can be accessed via script within an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

```
_AD_.ValueFactory = _AD_.Testbench.ValueFactory
```

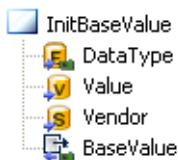
Related topics

Basics

[Accessing Simulation Platforms via XIL API](#)..... 117

InitBaseValue

Graphical representation



Purpose

To initialize a base value data object.

Description

BaseValue data objects are used in different automation blocks, for example:

- [Read](#) on page 299
- [Write](#) on page 301

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
(DataType)	In	Float	eINT	Selects the data type of the base value data object.
(Value)	In	Variant	None	Sets the value of the base value data object.
(Vendor)	In	String	"dSPACE"	Specifies the vendor and version of the used XIL API implementation. With dSPACE specified as the vendor, the current version of the dSPACE XIL API implementation is used. To specify another version or another vendor, use the following format: <code><VendorName>_<ProductName>_<ProductVersion></code> Example: <code>dSPACE GmbH_XIL_API_2019-A</code> It is recommended to use a globally managed Vendor data object as a reference providing the vendor name. For more information, refer to Vendor (Data Object) on page 160.
(BaseValue)	Out	BaseValue (Data Object)	None	Contains the instantiated BaseValue data object.

Access via Exec block

Alternatively, you can initialize a base value data object via Python using an Exec block. For further information, refer to [BaseValue \(Data Object\)](#) on page 274.

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

[BaseValue \(Data Object\)](#)..... 274

WatcherHandling

Introduction

The WatcherHandling folder provides data objects and automation blocks to create and configure watcher objects.

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

[Watcher \(Data Object\)](#)..... 279

Represents the base class for all types of Watcher data objects.

[WatcherFactory \(Data Object\)](#)..... 280

To instantiate a factory object to create a watcher.

[InitConditionWatcher](#)..... 281

To initialize a watcher data object to a specific condition.

[InitDurationWatcher](#)..... 284

To initialize a watcher data object to a specific duration.

Watcher (Data Object)

Graphical representation**Purpose**

Represents the base class for all types of Watcher data objects.

Description

Watcher data objects are used in the following automation blocks:

- [InitConditionWatcher](#) on page 281
- [InitDurationWatcher](#) on page 284

- [SetStartTriggerCondition](#) on page 245
- [SetStopTriggerCondition](#) on page 246

The Watcher data object can be instantiated as a ConditionWatcher by using the InitConditionWatcher block or as a DurationWatcher by using the InitDurationWatcher block. You have to initialize the data object with **None** to release it or use the **Release** method.

Access via Exec block

The Python module `xilapi` provides the possibility to work with the Watcher data object within an Exec block. The following examples show how to instantiate a Watcher data object as ConditionWatcher or DurationWatcher.

```
# Instantiating a ConditionWatcher
_AD_.ConditionWatcher = xilapi.InitWatcher(
    _AD_.Vendor,           # vendor = "dSPACE"
    "ConditionWatcher")   # WatcherType
# Instantiating a DurationWatcher
_AD_.DurationWatcher = xilapi.InitWatcher(
    _AD_.Vendor,           # vendor = "dSPACE"
    "DurationWatcher",     # WatcherType
    3.4)                  # duration value
```

Related topics

Basics

[Accessing Simulation Platforms via XIL API](#)..... 117

WatcherFactory (Data Object)

Graphical representation



Purpose

To instantiate a factory object to create a watcher.

Description

The WatcherFactory data object is used to instantiate and configure a Watcher object.

Access via Exec block

This data object can be accessed via script within an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

The following example shows how to create a DurationWatcher.

```
_AD_.WatcherFactory = \
    _AD_.Testbench.WatcherFactory
_AD_.DurationWatcher = \
    _AD_.WatcherFactory.CreateDurationWatcher(4.0)
```

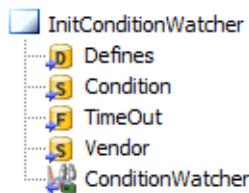
Related topics

Basics

[Accessing Simulation Platforms via XIL API.....](#) 117

InitConditionWatcher

Graphical representation



Purpose

To initialize a Watcher data object to a specific condition.

Description

You can use a Watcher data object to implement an event generator mechanism. For example, an initialized ConditionWatcher can be used to trigger the start and stop of a capture, refer to [SetStartTriggerCondition](#) on page 245 and [SetStopTriggerCondition](#) on page 246.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Defines	In	Dictionary	{}	Specifies the key-value pair: <ul style="list-style-type: none">▪ Key = symbol name▪ Value = variable path to the trace file
Condition	In	String	" "	Specifies the trigger condition, refer to Specifying the condition.
TimeOut	In	Float	-1.0	Specifies a timeout value in seconds for the selected trigger condition. In case the trigger condition never becomes true, you can set a timeout. So, the InitConditionWatcher block stops evaluating its trigger condition and fires its

Name	In / Out	Type	Default Value	Description
Vendor	In	String	"dSPACE"	<p>event as soon as the time of the TimeOut parameter is elapsed. This prevents that the InitConditionWatcher block is used endlessly.</p> <ul style="list-style-type: none"> ▪ -1.0 = infinite timeout (timeout is disabled) ▪ 0.0 = the InitConditionWatcher block fires its event immediately and the specified trigger condition is ignored <p>The CaptureResult's MataData dictionary provides the information whether a timeout occurred.</p> <p>Specifies the vendor and version of the used XIL API implementation. With dSPACE specified as the vendor, the current version of the dSPACE XIL API implementation is used.</p> <p>To specify another version or another vendor, use the following format:</p> <pre><VendorName>_<ProductName>_<ProductVersion></pre> <p>Example:</p> <pre>dSPACE GmbH_XIL API_2019-A</pre> <p>It is recommended to use a globally managed Vendor data object as a reference providing the vendor name. For more information, refer to Vendor (Data Object) on page 160.</p>
Condition\Watcher	Out	Watcher (Data Object)	None	Contains the instantiated Watcher data object.

Specifying the condition

You can set a trigger condition as a string.

If you use the Expression Editor to specify the string, the condition can be syntactically checked while you are editing it.

The syntax must be HIL/XIL API conform (regular expressions, subset of the ASAM General Expression Syntax)

Syntax and Arguments	Description
Bool &> Bool	Sequential evaluation of several trigger conditions: if the condition on the left side is true, the evaluation of the condition on the right side starts (and continues even if the condition on the left side does not remain true).
Bool Bool	Logical or
Bool ^^ Bool	Logical XOR
Bool && Bool	Logical AND
Bool == Bool Number == Number	Equality (the comparison of floating-point numbers is implementation-specific)
Bool != Bool Number != Number	Non-equality
Number < Number	Less than
Number > Number	Greater than
Number <= Number	Less or equal

Syntax and Arguments	Description
Number >= Number	Greater or equal
Number * Number	Multiplication
Number / Number	Division
Number + Number	Addition
Number - Number	Subtraction
- Number	Negation
+ Number	Positive sign: Has no effect, just indicates a positive number
! Bool	Logical NOT
sin (Number)	Sine (argument in radians)
cos (Number)	Cosine (argument in radians)
pow (Number)	Power (pow(a,b) -> a ^b)
abs (Number)	Absolute value
min (Number, Number)	Minimum
max (Number, Number)	Maximum
posedge (Variable, Number(Threshold))	Detection of positive edge: Returns true if the value of the signal defined by the variable changes from a value lower than the threshold to a value greater than or equal to the threshold.
negedge (Variable, Number(Threshold))	Detection of negative edge: Returns true if the value of the signal defined by the variable changes from a value higher than the threshold to a value less than or equal to the threshold.
changed (Variable, Number(Delta))	Detection of value change: Change is detected if the difference between the current number and its direct successor (number in the last evaluation step) is greater than or equal to the given delta.

Example The following expression returns true if the Rate_Limiter_Out1 signal changes from a negative value to zero or to a positive value:

```
posedge(Rate_Limiter_Out1,0)
```

Examples for combined trigger conditions If the trigger is assigned to a SCALEXIO Processing Unit platform, you can combine trigger conditions using the syntax and arguments shown in the table above:

```
posedge(Rate_Limiter_Out1,0) + posedge(Rate_Limiter_Out1,1)
```

or

```
(Var1 >= 7 && Var2 < 8) || posedge(Var3,1)
```

For more information on the ASAM XIL API standard, refer to:

- [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#)
- [ASAM_AE_XIL_Generic-Simulator-Interface_BS-2-4_CSharp-API-Technology-Reference-Mapping-Rules_V2-1-0.pdf](#)

Access via Exec block

For an example, refer to [Watcher \(Data Object\)](#) on page 279.

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

Edit (Condition String in ASAM GES) (AutomationDesk Basic Practices)	245
SetStartTriggerCondition	246
SetStopTriggerCondition	246
Watcher (Data Object)	279

InitDurationWatcher

Graphical representation**Purpose**

To initialize a Watcher data object to a specific duration.

Description

You can use an initialized DurationWatcher with SetStopTriggerCondition automation blocks (TriggerDefinition).

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
Duration	In	Float	0.0	Sets the duration in seconds.
Vendor	In	String	"dSPACE"	<p>Specifies the vendor and version of the used XIL API implementation. With dSPACE specified as the vendor, the current version of the dSPACE XIL API implementation is used.</p> <p>To specify another version or another vendor, use the following format:</p> <p style="background-color: #f0f0f0; padding: 2px;"><code><VendorName>_<ProductName>_<ProductVersion></code></p> <p>Example:</p> <p style="background-color: #f0f0f0; padding: 2px;"><code>dSPACE GmbH_XIL API_2019-A</code></p> <p>It is recommended to use a globally managed Vendor data object as a reference providing the vendor name. For more information, refer to Vendor (Data Object) on page 160.</p>

Name	In / Out	Type	Default Value	Description
Duration\Watcher	Out	Watcher (Data Object)	None	Contains the instantiated Watcher data object.

Access via Exec blockFor an example, refer to [Watcher \(Data Object\)](#) on page 279.**Related topics****Basics**[Accessing Simulation Platforms via XIL API](#)..... 117**References**[SetStopTriggerCondition](#)..... 246[Watcher \(Data Object\)](#)..... 279

MAPort

Introduction

The MAPort folder provides data objects and automation blocks to access the model access port of the XIL API. It lets you write variables to and read variables from the running simulation application.

Where to go from here**Information in this section**[MAPort \(Data Object\)](#)..... 286

To access the model that is simulated on the HIL simulator.

[MAPortFactory \(Data Object\)](#)..... 288

To instantiate a factory object to create an MAPort object.

[MAPortConfiguration \(Data Object\)](#)..... 289

To configure the application path and the platform identifier of the MAPort.

[CreateCapture](#)..... 290

To initialize a Capture data object with the given task.

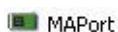
[CreateSignalGenerator](#)..... 291

To initialize a SignalGenerator data object.

GetDataType	292
To return the data type for the specified variable name.	
GetTaskNames	293
To return the names of all available tasks.	
GetVariableNames	294
To return the names of all available variables.	
InitMAPort	295
To initialize the model access port data object (MAPort).	
IsReadable	297
To check if a specified variable (VariableName) is readable.	
IsWritable	298
To check if a specified variable (VariableName) is writable.	
Read	299
To return the value of a variable.	
ReleaseMAPort	300
To release an MAPort instance.	
Write	301
To set the value of a variable.	

MAPort (Data Object)

Graphical representation



Purpose

To access the model that is simulated on the HIL simulator.

Description

The MAPort data object is used as a wrapper for the MAPort class or contains the MAPort instance which is created when using the InitMAPort block. The Model Access port is the central point for managing access to the model, simulated on the HIL simulator. This port provides functionality for read and write to the model, to set up captures and stimuli, and to manage model variables.

An MAPort instance can only be used for one simulation application. If you want to load another simulation application, you must release the MAPort instance beforehand.

An MAPort instance also blocks external access to the same platform. For example, you cannot load a simulation application by using the Platform Management library or the Platform Manager in AutomationDesk or ControlDesk, while an MAPort instance is accessing the platform.

To free the allocated memory of the MAPort instance (HIL/XIL API class instance) which is stored in the MAPort data object, you have to use the `ReleaseMAPort` block or use the `Release` method.

Access via Exec block

This data object can be accessed via script within an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object.

The available methods and properties depend on the standard used.

The following methods are available:

- `InitMAPort`
- `Release`
- `GetAttachState`
- `GetDataType`
- `GetTaskNames`
- `GetVariableNames`
- `CreateCapture`
- `CreateSignalGenerator`
- `IsReadable`
- `IsWritable`
- `Read`
- `Write`
- `Configure`
- `LoadConfiguration`
- `StartSimulation`
- `StopSimulation`
- `Dispose`
- `Disconnect`
- `GetVariableInfo`
- `CheckVariableNames` (XIL API 2.1.0 only)
- `CreateTargetScript` (XIL API 2.1.0 only)
- `DownloadParameterSets` (XIL API 2.1.0 only)
- `PauseSimulation` (XIL API 2.1.0 only)

The following properties are available:

- `DAQClock`
- `Configuration`
- `Name`
- `TaskInfos`
- `TaskNames`
- `State`

In offline operation mode, you have to use the related `Get` methods, for example, `GetDAQClock(OfflineValue)`.

The following example shows how to initialize an MAPort instance via script.

```
try:
    _AD_.MAPort.InitMAPort(
        _AD_.Vendor           # vendor = "dSPACE"
        _AD_.ConfigurationDict) # MAPortConfiguration data object
    ...
finally:
    # To release the MAPort instance
    _AD_.MAPort.Release()
```

Note

If you use the `StartSimulation` method to start the application on the simulation platform, the MAPort must be in the `eSIMULATION_PAUSED` state or in the `eSIMULATION_STOPPED` state. Otherwise, an exception occurs.

If you also want to accept the `eSIMULATION_RUNNING` state, use the `StartSimulation` block of the XIL API Convenience library.

If you use the `StopSimulation` method to stop the application on the platform, the MAPort must be in the `eSIMULATION_PAUSED` state or in the `eSIMULATION_RUNNING` state. Otherwise, an exception occurs.

If you also want to accept the `eSIMULATION_STOPPED` state, use the `StopSimulation` block of the XIL API Convenience library.

Related topics

Basics

Accessing Simulation Platforms via XIL API	117
--	-----

HowTos

How to Build a Basic Sequence for Accessing Simulator Variables	78
---	----

References

InitMAPort	295
ReleaseMAPort	300
StartSimulation	205
StopSimulation	206

MAPortFactory (Data Object)

Graphical representation



Purpose

To instantiate a factory object to create an MAPort object.

Description

The MAPortFactory data object is used to create and configure an MAPort object.

Access via Exec block

The MAPortFactory data object can be accessed within an Exec block.

The name completion feature of the Python Editor provides the available properties and methods of the current object. For more information on the properties and methods, refer to [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

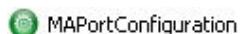
The following example shows how to initialize an MAPort instance via script.

```
_AD_.MAPortFactory = _AD_.Testbench.MAPortFactory
_AD_.MAPort = _AD_.MAPortFactory.CreateMAPort("MyMAPort")
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API.....](#) 117

MAPortConfiguration (Data Object)

Graphical representation**Purpose**

To configure the application path and the platform identifier of the MAPort.

Description

If you use an implementation of the ASAM XIL API standard, this data object is required to specify the model access ports that are used to access platforms.

To configure a model access port for a dSPACE XIL API implementation, you can use the **Edit (MAPortConfiguration)** command. For more information, refer to [How to Make Simulator Variables Available to XIL API Library Automation Blocks](#) on page 127.

To initialize the MAPort for another vendor than dSPACE, you have to use the **Edit as Dictionary** command from the data object's context menu. Delete the **PlatformIdentifier** and the **ApplicationPath** item from the dictionary and enter the following new items:

Key	Value Type	Value
name	String	Specifies the name of the model access port instance.
filepath	String	Specifies the absolute path of the model access port configuration file.
forceConfig	Boolean	<p>Specifies the downloading behavior in case the specified simulation application is already running on the platform.</p> <ul style="list-style-type: none"> ▪ Specify True to force a new downloading when the model access port is initialized. ▪ Specify False to avoid a new downloading when the model access port is initialized. <p>To specify a Boolean value, enter True or False without quotation marks in the Value Editor.</p>

For further information, refer to chapter 5.2.2.1 in [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

Related topics

Basics

[Accessing Simulation Platforms via XIL API.....](#) 117

HowTos

[How to Build a Basic Sequence for Accessing Simulator Variables.....](#) 78

References

[Edit as Dictionary.....](#) 360

[InitMAPort.....](#) 295

[MAPort \(Data Object\).....](#) 286

CreateCapture

Graphical representation



Purpose

To initialize a Capture data object with the given task.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Contains the instantiated MAPort data object.

Name	In / Out	Type	Default Value	Description
TaskName	In	String	" "	Sets the task name of the measurement.
Capture	Out	Capture (Data Object)	None	Contains the created Capture object.

Access via Exec block

The Python module `xilapi` provides the possibility to work with the `CreateCapture` method within an Exec block.

```
_AD_.Capture = _AD_.MAPort.CreateCapture(_AD_.TaskName)
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

Capture (Data Object)	231
MAPort (Data Object)	286

CreateSignalGenerator

Graphical representation**Purpose**

To initialize a SignalGenerator data object.

Description

If you want to use a SignalGenerator data object in your sequence, you must initialize it by using this automation block.

After the initialization of the SignalGenerator and the SignalDescriptionSet, you can use the properties and methods from the HIL/XIL API standard in an Exec block for working with these data structures.

In offline operation mode, the SignalGenerator data object is only passed as a parameter.

Data objects		This automation block provides the following data objects:		
Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Contains the instantiated MAPort data object.
SignalGenerator	Out	SignalGenerator (Data Object)	None	Contains the signal description set (STZ file) for stimulating variables.

Access via Exec block The Python module `xilapi` provides the possibility to work with a signal generator within an Exec block. The following example shows how to access a stimulus signal and start it.

```
_AD_.SignalGenerator = _AD_.MAPort.CreateSignalGenerator()
```

Tip

See also the XILAPISimulus demo project.

Related topics

Basics

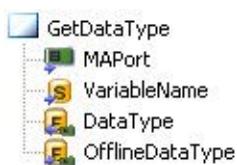
[Accessing Simulation Platforms via XIL API](#)..... 117

References

MAPort (Data Object)	286
SignalGenerator (Data Object)	254

GetDataType

Graphical representation



Purpose

To return the data type for the specified variable name.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Contains the instantiated MAPort data object.
VariableName	In	String	""	Sets the VariableName to be checked.
DataType	Out	Float	eINT	Returns the data type for the specified variable name. The value is contained in <code>xilapi.Common.ValueContainer.Enum.DataType</code> .
OfflineDataType	In	Float	eINT	Lets you specify the OfflineDataType to be used in offline operation mode.

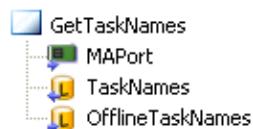
Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

[MAPort \(Data Object\)](#)..... 286

GetTaskNames

Graphical representation**Purpose**

To return the names of all available tasks.

Description	Note
	<p>With AutomationDesk 3.6, the data type of the TaskNames data object has been changed from Dictionary to List. The data objects are automatically migrated if you load an AutomationDesk project created with an earlier version, but the references which you have specified for parameterizing this data object remain unchanged.</p> <ul style="list-style-type: none"> ▪ Use the Find Inconsistencies dialog to search for invalid references. The search result will contain the references to the TaskNames data objects to be manually migrated. <p>This has to be applied also to the OfflineTaskNames data object.</p>

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Contains the instantiated MAPort data object.
TaskNames	Out	List	[]	Returns the names of all available tasks as a list of strings.
OfflineTaskNames	In	List	[]	Lets you specify the TaskNames to be used in offline operation mode.

Access via Exec block

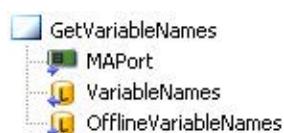
Additionally, you can return the names of all available tasks via Exec block.

```
TaskNames = _AD_.MAPort.GetTaskNames(OfflineTaskNames)
# OfflineTaskNames is optional
```

Related topics**References**

Find Inconsistencies (AutomationDesk Basic Practices 	
MAPort (Data Object).....	286

GetVariableNames

Graphical representation

Purpose	To return the names of all available variables.
----------------	---

Data objects	This automation block provides the following data objects:
---------------------	--

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Contains the instantiated MAPort data object.
VariableNames	Out	List	[]	Returns the names of all available variables.
OfflineVariableNames	In	List	[]	Lets you specify the VariableNames to be used in offline operation mode.

Access via Exec block

Additionally, you can return the names of all available variables via Exec block.

```
VariableNames = _AD_.MAPort.GetVariableNames(OfflineVariableNames)
# OfflineVariableNames list is optional
```

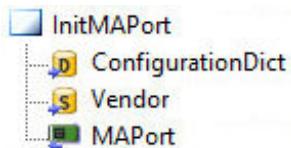
Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

[MAPort \(Data Object\)](#)..... 286

InitMAPort

Graphical representation**Purpose**

To initialize the model access port data object (MAPort).

Description

The InitMAPort automation block is used to prepare the connection to the model that is simulated on the HIL simulator.

The MAPort data object is created at the first execution of this block. Subsequent executions have no effect.

Note

The following preconditions have to be fulfilled before you can use the MAPort for accessing a simulation platform:

- The simulation platform has to be registered, refer to [Registering and Managing dSPACE Platforms](#) on page 26.
- A loaded simulation application must match the MAPort configuration.
- If the real-time application is not already loaded to the platform, it is automatically loaded but not started. You have to explicitly start the simulation before executing model access.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
ConfigurationDict	In	Dictionary	{"ApplicationPath": "", "PlatformIdentifier": ""}	<p>Sets the following items:</p> <ul style="list-style-type: none"> ▪ <i>ApplicationPath</i> Absolute path and name of variable file (trc or sdf) ▪ <i>PlatformIdentifier</i> ds1006, ds1006_1, ..., ds1401, ds1401_1, ..., IP address (for SCALEXIO MC), Name of the MP system (SCALEXIO MP), or VEOS <p>Alternatively, you can reference to an MAPortConfiguration data object that specifies the MAPort configuration.</p>
Vendor	In	String	"dSPACE"	<p>Specifies the vendor and version of the used XIL API implementation.</p> <p>With dSPACE specified as the vendor, the current version of the dSPACE XIL API implementation is used.</p> <p>To specify another version or another vendor, use the following format:</p> <pre><VendorName>_<ProductName>_<ProductVersion></pre> <p>Example:</p> <pre>dSPACE GmbH_XIL API_2019-A</pre> <p>It is recommended to use a globally managed Vendor data object as a reference providing the vendor name. For more information, refer to Vendor (Data Object) on page 160.</p>
MAPort	Out	MAPort (Data Object)	None	Returns the initialized MAPort data object.

Tip

- You can get the list of vendor names for the currently available XIL API implementations via the `xilapi.GetVendorNames()` method.
- If you want to use an XIL API implementation of a vendor other than dSPACE, you must adjust the MAPortConfiguration data object. Refer to [MAPortConfiguration \(Data Object\)](#) on page 289.

Access via Exec block

Alternatively, you can initialize the MAPort data object via Exec block. For further information, refer to [MAPort \(Data Object\)](#) on page 286.

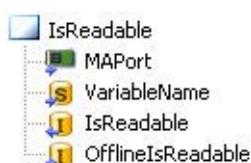
Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

[MAPort \(Data Object\)](#)..... 286
[MAPortConfiguration \(Data Object\)](#)..... 289

IsReadable

Graphical representation**Purpose**

To check if a specified variable (VariableName) is readable.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Contains the instantiated MAPort data object.
VariableName	In	String	" "	Sets the VariableName to be checked.
IsReadable	Out	Int	0	Returns whether the variable is readable or not. <ul style="list-style-type: none"> ▪ Nonzero means: the variable is readable

Name	In / Out	Type	Default Value	Description
OfflineIsReadable	In	Int	0	<ul style="list-style-type: none"> ▪ 0 means: variable is not readable <p>Lets you specify the IsReadable to be used in offline operation mode.</p>

Access via Exec block

The MAPort data object provides the possibility to use the block's functionality within an Exec block.

```
ReadableFlag = _AD_.MAPort.IsReadable(_AD_.VariableName, _AD_.OfflineValue)
# OfflineValue (Typ Int) is optional
```

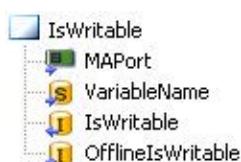
Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

[MAPort \(Data Object\)](#)..... 286

IsWritable

Graphical representation**Purpose**

To check if a specified variable (VariableName) is writable.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Contains the instantiated MAPort data object.
VariableName	In	String	" "	Sets the VariableName to be checked.
IsWritable	Out	Int	0	Returns whether the variable is writeable or not. <ul style="list-style-type: none"> ▪ Unequal 0 means: the variable is writeable

Name	In / Out	Type	Default Value	Description
OfflineIsWriteable	In	Int	0	<ul style="list-style-type: none"> ▪ 0 means: variable is not writeable <p>Lets you specify the IsWriteable to be used in offline operation mode.</p>

Access via Exec block

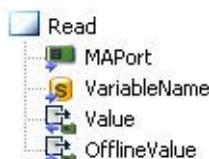
The MAPort data object provides the possibility to use the block's functionality within an Exec block.

```
WritableFlag = _AD_.MAPort.IsWritable(_AD_.VariableName, _AD_.OfflineValue)
# OfflineValue (Typ Int) is optional
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API.....](#) 117

Read

Graphical representation**Purpose**

To return the value of a variable.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Contains the initialized MAPort data object.
VariableName	In	String	""	Sets the VariableName to be read.
Value	Out	BaseValue (Data Object)	None	Returns the value of the specified variable.
OfflineValue	In	BaseValue (Data Object)	None	Lets you specify the Value to be used in offline operation mode.

Access via Exec block

The MAPort data object provides the possibility to use the block's functionality within an Exec block.

```
_AD_.BaseValue = _AD_.MAPort.Read(
    _AD_.VariableName,
    _AD_.OfflineValue) # OfflineValue (Typ Basevalue) is optional
```

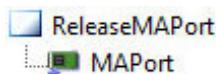
Related topics**Basics**

Accessing Simulation Platforms via XIL API.....	117
---	-----

References

BaseValue (Data Object).....	274
MAPort (Data Object).....	286

ReleaseMAPort

Graphical representation**Purpose**

To release an MAPort instance.

Description

This automation block is used to release the currently used MAPort, for example, if you want to switch to another platform or if you want to load another simulation application.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Contains the instantiated MAPort data object to be released.

Access via Exec block

Alternatively, you can release the instance via script within an Exec block.

```
_AD_.MAPort.Release()
```

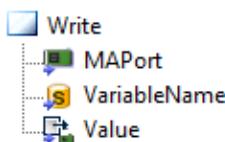
Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

MAPort (Data Object)	286
Release MAPort	365

Write

Graphical representation**Purpose**

To set the value of a variable.

Data objects

This automation block provides the following data objects:

Name	In / Out	Type	Default Value	Description
MAPort	In	MAPort (Data Object)	None	Contains the initialized MAPort data object.
VariableName	In	String	""	Sets the name of the variable to write to.
Value	In	BaseValue (Data Object)	None	Sets the value of the specified variable.

Access via Exec block

The MAPort data object provides the possibility to use the block's functionality within an Exec block.

```
_AD_.MAPort.Write(_AD_.VariableName, _AD_.BaseValue)
```

Related topics**Basics**

[Accessing Simulation Platforms via XIL API](#)..... 117

References

BaseValue (Data Object)	274
MAPort (Data Object)	286

Commands And Dialogs

Where to go from here

Information in this section

Platform Manager	302
Commands, properties, and dialogs for managing platforms in an AutomationDesk project.	
XIL API Framework	342
Description of the commands for working with an XIL API Framework.	
XIL API Convenience	350
Description of the specific commands of the XIL API Convenience library.	
XIL API	355
Description of the specific commands of the XIL API library.	

Platform Manager

Introduction

AutomationDesk provides the following properties, commands and dialogs for managing platforms.

Where to go from here

Information in this section

Assembly View	305
To display a component-based view of the registered hardware in the Platform Manager.	
Clear Flash	306
To clear the flash memory of the selected multiprocessor platform, in whole or in part.	
Clear Flash Options - Clear Complete Flash Memory	307
To clear the complete flash memory of the selected hardware.	
Clear Flash Options - Clear Flash Application	308
To clear the application loaded to the flash memory of the selected hardware.	
Clear Flash Options - Clear Flight Recorder Data	308
To clear the flight recorder data from the flash memory of the selected hardware.	

Clear Flash Options - Clear Nonvolatile Data	309
To clear nonvolatile data from the flash memory of the selected hardware.	
Clear System	310
To clear the entire system you are currently working with.	
Collapse	310
To collapse the platforms and subnodes of the node selected in the Platform Manager.	
Create Support Info	311
To generate an XML file containing textual information on platforms/devices that are currently detected by the Platform Manager.	
Expand	312
To expand the collapsed platforms and subnodes of the node selected in the Platform Manager.	
Explore Logged Data	312
To save the logged data currently available in a USB mass storage device connected to the platform hardware.	
Manage Platforms	314
To display and manage the platforms that were registered in your system.	
Network View	316
To display a network-based view of the registered hardware in the Platform Manager.	
Pause	317
To pause an offline simulation.	
Platform Manager	317
To display the hardware components of all hardware systems connected to your host PC and accessible via AutomationDesk.	
Properties (Platform/Device)	319
To view the properties of the selected platform.	
Real-Time Application - Load	319
To load a real-time application to the RAM of the selected hardware, and start it automatically.	
Real-Time Application / Offline Simulation Application - Load	320
To load an application to the RAM of the selected hardware or to VEOS. After downloading, the application is not started.	
Real-Time Application / Offline Simulation Application - Load and Start	321
To load an application to the RAM of the selected hardware or to VEOS, and start it automatically.	
Real-Time Application - Load to Flash (DS1006/DS1104/MicroAutoBox II)	322
To load an application to the flash memory of the selected hardware, and start it automatically.	

Real-Time Application - Load to Flash (DS1007/MicroLabBox/MicroAutoBox III/SCALEXIO)	323
To load an application to the flash memory of the selected hardware.	
After loading, the application is not started.	
Real-Time Application - Load to Flash and Start (DS1007/MicroLabBox/MicroAutoBox III/SCALEXIO)	324
To load an application to the flash memory of the selected hardware, and start it automatically.	
Real-Time Application - Reload	325
To reload the currently loaded application to the RAM.	
Refresh Interface Connections	325
To refresh the interface connections between AutomationDesk and the hardware.	
Refresh Platform Configuration	326
To refresh the hardware configuration.	
Register Platforms	326
To register dSPACE real-time hardware and VEOS.	
Reload	331
To reload the currently loaded application to the RAM. After reloading, the application is not started.	
Reload and Start	332
To reload the currently loaded application to the RAM. After reloading, the application is started.	
Real-Time Application - Reload to Flash	332
To reload the currently loaded application to the flash memory of the selected hardware.	
Reload to Flash	333
To reload the currently loaded application to the flash memory. After reloading, the application is not started.	
Reload to Flash and Start	334
To reload the currently loaded application to the flash memory. After reloading, the application is started.	
Show Connected Clients	335
To display the clients to which the selected SCALEXIO platform or SCALEXIO Processing Unit is connected.	
Single Step	335
To run a pausing or stopped offline simulation stepwise.	
Start	336
To start the selected application.	
Stop	337
To stop the selected application.	

[Stop RTP.....](#) 337

To stop the application running on the selected platform.

[Stop RTPs.....](#) 338

To stop the applications running on the selected Multiprocessor System platform.

[Unload.....](#) 338

To unload the selected application.

[Update Firmware.....](#) 339

To update the firmware of the selected platform.

Assembly View

Access

You can access this command via:

Ribbon	Platforms - Views
Context menu of	Platform Manager
Shortcut key	None
Icon	

Purpose

To display a component-based view of the registered hardware in the Platform Manager.

Result

The Platform Manager displays the physical assembly of the registered hardware.

Related topics**References**[Network View.....](#) 316

Clear Flash

Access

The hardware must be connected to the host PC. You can access the command via:

Ribbon	None
Context menu of	Platform Manager – platforms
Shortcut key	None
Icon	None

This command is available for multiprocessor platforms only.

Purpose

To clear the flash memory of the selected multiprocessor platform, in whole or in part.

Note

Relevant for Multiprocessor System platforms: To clear the flash memory of a single processor that belongs to the DS1006-based multiprocessor system, you can alternatively call the Clear Flash Options - Clear Complete Flash Memory or Clear Flash Options - Clear Flash Application command from the context menu of the processor.

Result

AutomationDesk opens the Clear Flash dialog, which lets you completely or partly clear the flash memory of the selected multiprocessor platform. The real-time processors must be reset for this. If an application is running on the selected platform when the Clear Flash dialog opens, you are asked to stop the running RTPs or unload the running applications, respectively.

The dialog displays all connected processing units and/or processor boards that belong to the selected platform and the applications currently loaded to their flash memories. If more than one clearing option is possible for the multiprocessor platform, you can choose the appropriate option. Finally you can select the processing units or processor boards for which the clear flash option is to be applied and start the clearing.

Clear Flash dialog

To completely or partly clear the flash memory of the selected multiprocessor platform.

Clear options Lets you select the clearing option to be applied to the multiprocessor platform. The available clearing options are offered for selection. If only one clearing option is possible, it is selected automatically and you cannot change it.

Processor Board/Processing Unit Displays the processor boards and/or processing units belonging to the multiprocessor system.

Application in Flash Displays for each processor board or processing unit the path of the application that is loaded to the flash memory. If there is no application in the flash memory or after clearing the flash memory, 'No application loaded.' is displayed.

For DS1006 boards of a Multiprocessor System platform, the value 'Currently not available.' might be displayed. This value means that it is not clear whether the application is running on the flash or RAM. The value is set after registering a platform or after the Refresh Interface Connections command is executed.

Select Lets you select the individual processor boards or processing units to which the selected clearing option is to be applied.

Clear Lets you execute the selected clearing option for the selected processing units/processor boards.

Clear Flash Options - Clear Complete Flash Memory

Access

The hardware must be connected to the host PC. You can access the command via:

Ribbon	None
Context menu of	Platform Manager – platforms
Shortcut key	None
Icon	None

Purpose

To clear the complete flash memory of the selected hardware.

Result

AutomationDesk completely clears the flash memory of the selected platform, i.e., the application loaded to the flash memory, the nonvolatile data, and the flight recorder data are cleared. To do this, the real-time processor must be reset. If an application is running on the selected platform when you call the command, you are asked to stop the running RTP.

Related topics

References

Clear Flash.....	306
Clear Flash Options - Clear Flash Application.....	308
Clear Flash Options - Clear Flight Recorder Data.....	308
Clear Flash Options - Clear Nonvolatile Data.....	309

Clear Flash Options - Clear Flash Application

Access	The hardware must be connected to the host PC. You can access the command via:								
Ribbon	None								
Context menu of	Platform Manager – platforms								
Shortcut key	None								
Icon	None								
Purpose	To clear the application loaded to the flash memory of the selected hardware.								
Result	AutomationDesk clears the application loaded to the flash memory of the selected platform. The real-time processor must be reset for this. If the application is running on the selected platform when you call the command, you are asked to stop the running RTP.								
Related topics	References								
	<table><tr><td>Clear Flash.....</td><td>306</td></tr><tr><td>Clear Flash Options - Clear Complete Flash Memory.....</td><td>307</td></tr><tr><td>Clear Flash Options - Clear Flight Recorder Data.....</td><td>308</td></tr><tr><td>Clear Flash Options - Clear Nonvolatile Data.....</td><td>309</td></tr></table>	Clear Flash.....	306	Clear Flash Options - Clear Complete Flash Memory.....	307	Clear Flash Options - Clear Flight Recorder Data.....	308	Clear Flash Options - Clear Nonvolatile Data.....	309
Clear Flash.....	306								
Clear Flash Options - Clear Complete Flash Memory.....	307								
Clear Flash Options - Clear Flight Recorder Data.....	308								
Clear Flash Options - Clear Nonvolatile Data.....	309								

Clear Flash Options - Clear Flight Recorder Data

Access	The hardware must be connected to the host PC. You can access the command via:
Ribbon	None
Context menu of	Platform Manager – platforms
Shortcut key	None
Icon	None
Purpose	To clear the flight recorder data from the flash memory of the selected hardware.

Result	AutomationDesk clears the flight recorder data from the flash memory of the selected platform. The real-time processor must be reset for this. If the application is running on the selected platform when you call the command, you are asked to stop the running RTP.
---------------	---

Related topics	References
	Clear Flash..... 306
	Clear Flash Options - Clear Complete Flash Memory..... 307
	Clear Flash Options - Clear Flash Application..... 308
	Clear Flash Options - Clear Nonvolatile Data..... 309

Clear Flash Options - Clear Nonvolatile Data

Access	The hardware must be connected to the host PC. You can access the command via:
	Ribbon
	Context menu of
	Shortcut key
	Icon
	None
	Platform Manager – platforms
	None
	None

Purpose	To clear nonvolatile data from the flash memory of the selected hardware.
----------------	---

Result	AutomationDesk clears the nonvolatile data from the flash memory of the selected platform. The real-time processor must be reset for this. If the application is running on the selected platform when you call the command, you are asked to stop the running RTP.
---------------	---

Related topics	References
	Clear Flash..... 306
	Clear Flash Options - Clear Complete Flash Memory..... 307
	Clear Flash Options - Clear Flash Application..... 308
	Clear Flash Options - Clear Flight Recorder Data..... 308

Clear System

Access

You can access this command via:

Ribbon	Platforms – Platform Management
Context menu of	None
Shortcut key	None
Icon	

Purpose

To clear the entire system you are currently working with.

Result

AutomationDesk clears the system by erasing the recent platform configuration. The Platform Manager and the device drivers are reset to their initial states.

Note

- This command deletes any registered platform from the recent platform configuration and not only the platform you are currently working with.
- Clearing the recent platform configuration is relevant to each dSPACE software installed on your PC that provides platform management.

Tip

Before clearing the system, you can use the Manage Recent Platform Configuration dialog, to export the currently active recent platform configuration. Importing this configuration allows you to recover the system after you have cleared it.

Refer to [Manage Platforms](#) on page 314.

Collapse

Access

You can access this command via:

Ribbon	None
Context menu of	Platform Manager
Shortcut key	None
Icon	None

Purpose To collapse the platforms and subnodes of the node selected in the Platform Manager.

Result AutomationDesk hides the subnodes and platforms of the node selected in the Platform Manager.

Related topics

References

Expand.....	312
-------------	-----

Create Support Info

Access

You can access the command via:

Ribbon	Platforms – Platform Management
Context menu of	Platform Manager
Shortcut key	None
Icon	

Purpose

To generate an XML file containing textual information on platforms/devices that are currently detected by the Platform Manager.

Result

To help dSPACE Support in analyzing an observed problem, the **SupportInfo.xml** file is generated. The file contains relevant information on all the platforms that are currently detected by the Platform Manager. The file is saved automatically, and the path is displayed in a message.

The **SupportInfo.xml** file is overwritten each time you call the Create Support Info command.

Related topics

HowTos

How to Collect Diagnostic Information via dSPACE Installation Manager (Providing Diagnostic Information )

Expand

Access

You can access this command via:

Ribbon	None
Context menu of	Platform Manager
Shortcut key	None
Icon	None

Purpose

To expand the collapsed platforms and subnodes of the node selected in the Platform Manager.

Result

AutomationDesk now displays the hidden subnodes and platforms of the node selected in the Platform Manager.

Related topics

References

[Collapse.....](#) 310

Explore Logged Data

Access

This command is available only for platforms that support data logging or flight recording.

The following platforms support data logging:

- MicroAutoBox III
- SCALEXIO system based on a DS6001 Processor Board
(except for SCALEXIO multiprocessor systems)

The following platforms support flight recording:

- DS1007 PPC Processor Board
- DS1202 MicroLabBox
- MicroAutoBox II

The hardware must be connected to the host PC. You can access the command via:

Ribbon	None
Context menu of	Platform Manager – platform

Shortcut key	None
Icon	None

Purpose To save the logged data currently available in a USB mass storage device connected to the platform hardware.

Result AutomationDesk opens the Logged Data on Mass Storage Device dialog, which displays the connected platform hardware and lists the files which were written to the USB mass storage device during flight recording. The logged data is written in BIN format or MF4 format, depending on the used platform. You can select BIN or MF4 files from the list and upload them to the host PC. Multiple selection is possible by pressing **Ctrl** or **Shift** when clicking a file. The target folder is selected in a standard Windows dialog.

You can also delete obsolete BIN or MF4 files from the USB mass storage device.

Description The dialog's status bar displays information on the selected file (file type, modification date, and size).

Logged Data on Mass Storage Device dialog To display the files stored in a USB mass storage device, and to upload BIN or MF4 files to the host PC or delete BIN or MF4 files from the USB mass storage device. In the dialog, you can access the following commands via the context menu:

Upload Lets you upload the selected BIN or MF4 files to the specified target folder on the host PC.

Delete Lets you delete the selected BIN or MF4 files from the USB mass storage device.

Related topics HowTos

[How to Upload Flight Recorder Data Written to a USB Mass Storage Device](#)

(ControlDesk Measurement and Recording 

[How to Upload Logged Data](#) (ControlDesk Measurement and Recording 

Manage Platforms

Access	You can access this command via:									
Ribbon	Platforms – Platform Management									
Context menu of	Platform Manager									
Shortcut key	None									
Icon										
Purpose	To display and manage the platforms that were registered in your system.									
Result	AutomationDesk opens the Manage Recent Platform Configuration dialog, which lets you manage your recent platform configuration. You can remove elements from the recent platform configuration and hide registered platforms in the drop-down lists in AutomationDesk. You can import configurations for registered platforms from an XML file or export the recent hardware configuration to an XML file.									
Description	<p>When you register a single dSPACE processor or controller board, a multiprocessor system, a SCALEXIO system, or an XIL API MAPort platform, AutomationDesk stores the registration data in the recent platform configuration.</p> <p>After you close the Manage Recent Platform Configuration dialog, AutomationDesk may open a dialog prompting you to refresh the interface connections. If so, call the Refresh Interface Connections command. Refer to Refresh Interface Connections on page 325.</p>									
Manage Recent Platform Configuration dialog	<p>To manage the registered platforms and import or export the configuration of registered platforms.</p> <p>Recent Platform Configuration Lists the platforms that were registered in your system and whose registration data is stored in the recent platform configuration, and displays some information on the registered platforms.</p> <p>Commands The following commands are available via buttons and from the menus or context menus:</p> <table border="1"> <thead> <tr> <th>Command</th> <th>Access</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Activate</td> <td> <ul style="list-style-type: none"> ▪ Context menu of an inactive platform ▪ Shortcut key: Alt+A </td> <td>Lets you activate the selected inactive platform(s). An active platform is displayed in the Platform Manager.</td> </tr> <tr> <td>Collapse</td> <td>Context menu of a platform</td> <td>Lets you collapse the member items of the platform selected in the platform list.</td> </tr> </tbody> </table>	Command	Access	Description	Activate	<ul style="list-style-type: none"> ▪ Context menu of an inactive platform ▪ Shortcut key: Alt+A 	Lets you activate the selected inactive platform(s). An active platform is displayed in the Platform Manager.	Collapse	Context menu of a platform	Lets you collapse the member items of the platform selected in the platform list.
Command	Access	Description								
Activate	<ul style="list-style-type: none"> ▪ Context menu of an inactive platform ▪ Shortcut key: Alt+A 	Lets you activate the selected inactive platform(s). An active platform is displayed in the Platform Manager.								
Collapse	Context menu of a platform	Lets you collapse the member items of the platform selected in the platform list.								

Command	Access	Description
Deactivate	<ul style="list-style-type: none"> ▪ Context menu of an active platform ▪ Shortcut key: Alt+D 	Lets you deactivate the selected platform(s). An inactive platform is hidden. It is not displayed in the Platform Manager.
Expand	Context menu of a platform	Lets you expand the collapsed elements of the platform selected in the platform list.
Export	<ul style="list-style-type: none"> ▪ Button ▪ File menu ▪ Shortcut key: Alt+E 	Lets you select the XML file you want to export the recent platform configuration to.
Group by Active State	View menu	Lets you group the platforms according to their Active state.
Group by Platform Type	View menu	Lets you group the platforms according to their platform type.
Import	<ul style="list-style-type: none"> ▪ Button ▪ File menu ▪ Shortcut key: Alt+I 	Lets you select the XML file containing the platform configuration you want to import. The currently active platform configuration is replaced by the content of the imported XML file. <div style="background-color: #f0f0f0; padding: 10px; margin-top: 10px;"> Note <p>You are recommended to import only recent platform configurations that you previously exported.</p> </div>
Refresh	<ul style="list-style-type: none"> ▪ View menu ▪ Context menu of a platform ▪ Shortcut key: F5 	Lets you refresh the visualization of the recent platform configuration in the dialog.
Remove	<ul style="list-style-type: none"> ▪ Button ▪ Edit menu ▪ Context menu of a platform ▪ Shortcut key: Del 	Lets you remove the currently selected platform from the recent hardware configuration. The platform is no longer available as an assignable registered platform and is no longer displayed in the Platform Manager. <div style="background-color: #f0f0f0; padding: 10px; margin-top: 10px;"> Note <p>You are recommended to perform the Refresh Interface Connections command after removing a platform that required registration at the device driver.</p> </div>
Remove All	<ul style="list-style-type: none"> ▪ Button ▪ Edit menu ▪ Shortcut key: Shift+Del 	Lets you remove all listed platforms from the recent hardware configuration. The platforms are no longer available as assignable registered platforms and are no longer displayed in the Platform Manager. <div style="background-color: #f0f0f0; padding: 10px; margin-top: 10px;"> Note <p>You are recommended to perform the Refresh Interface Connections command after removing a platform that required registration at the device driver.</p> </div>

Command	Access	Description
Remove Multiprocessor	Context menu of a Multiprocessor System platform	Lets you remove the selected Multiprocessor System platform from the recent hardware configuration. However, all the DS1006 processor boards of the multiprocessor system are converted to single platforms, which are then listed as separate platforms in the platform list.
Select All	<ul style="list-style-type: none"> ▪ Context menu of a platform ▪ Shortcut key: Ctrl+A 	Lets you select all the items in the platform list.
Sort Alphabetically	View menu	Lets you sort the platform list alphabetically in ascending order by platform names.

Network View

Access

You can access this command via:

Ribbon	Platforms - Views
Context menu of	Platform Manager
Shortcut key	None
Icon	

Purpose

To display a network-based view of the registered hardware in the Platform Manager.

Result

The Platform Manager displays the logical network of the registered hardware.

Related topics

References

[Assembly View.....](#) 305

Pause

Access

This command is available only for the VEOS platform, and if an application is currently running. You can access this command via:

Ribbon	None
Context menu of	Platform Manager – VEOS platform - application
Shortcut key	None
Icon	None

Purpose

To pause an offline simulation.

Result

The offline simulation pauses.

You can restart the offline simulation with the **Start** command or continue it stepwise with the **Single Step** command.

Related topics

References

Real-Time Application / Offline Simulation Application - Load.....	320
Real-Time Application / Offline Simulation Application - Load and Start.....	321
Single Step.....	335
Start.....	336
Stop.....	337

Platform Manager

Access

The Platform Manager is one of AutomationDesk's controlbars. You can display it via:

Ribbon	View - Controlbars
Context menu of	None
Shortcut key	None
Icon	

Purpose

To display the hardware components of all hardware systems connected to your host PC and accessible via AutomationDesk.

Description	<p>The Platform Manager displays all the hardware components which can be accessed via AutomationDesk. The hardware components are arranged in a hierarchical tree structure. Each node in the tree displays the name of the hardware component it represents. If you select a hardware component in the Platform Manager, you can open a context menu with component-specific commands.</p> <p>The Platform Manager allows you to:</p> <ul style="list-style-type: none"> ▪ Download, start, stop and unload real-time applications. ▪ Check the state of a real-time application. Its state is visualized in the Platform Manager by symbols below the Processing Unit node. 																																		
Symbols	<p>The Platform Manager gives access to all the hardware components and subcomponents of connected SCALEXIO systems which can be accessed via AutomationDesk. It displays each component together with a symbol giving information on the component type.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td></td><td>Indicates the host PC.</td></tr> <tr> <td></td><td>Indicates a SCALEXIO rack or system.</td></tr> <tr> <td></td><td>Indicates real-time PC.</td></tr> <tr> <td></td><td>Indicates an I/O unit.</td></tr> <tr> <td></td><td>Indicates a link board, for example, an IOCNET router.</td></tr> <tr> <td></td><td>Indicates an angle unit group.</td></tr> <tr> <td></td><td>Indicates an angle unit.</td></tr> <tr> <td></td><td>Indicates the real-time application.</td></tr> <tr> <td></td><td>Indicates a dSPACE real-time board, for example DS1006.</td></tr> <tr> <td></td><td>Indicates a VEOS.</td></tr> <tr> <td></td><td>Indicates a running real-time application.</td></tr> <tr> <td></td><td>Indicates a stopped real-time application.</td></tr> <tr> <td></td><td>Indicates a terminated real-time application.</td></tr> <tr> <td></td><td>Indicates a signal measurement board, a signal generation board, a bus board, an I/O module, or a bus module.</td></tr> <tr> <td></td><td>Indicates a channel group.</td></tr> <tr> <td></td><td>Indicates a channel.</td></tr> </tbody> </table>	Symbol	Meaning		Indicates the host PC.		Indicates a SCALEXIO rack or system.		Indicates real-time PC.		Indicates an I/O unit.		Indicates a link board, for example, an IOCNET router.		Indicates an angle unit group.		Indicates an angle unit.		Indicates the real-time application.		Indicates a dSPACE real-time board, for example DS1006.		Indicates a VEOS.		Indicates a running real-time application.		Indicates a stopped real-time application.		Indicates a terminated real-time application.		Indicates a signal measurement board, a signal generation board, a bus board, an I/O module, or a bus module.		Indicates a channel group.		Indicates a channel.
Symbol	Meaning																																		
	Indicates the host PC.																																		
	Indicates a SCALEXIO rack or system.																																		
	Indicates real-time PC.																																		
	Indicates an I/O unit.																																		
	Indicates a link board, for example, an IOCNET router.																																		
	Indicates an angle unit group.																																		
	Indicates an angle unit.																																		
	Indicates the real-time application.																																		
	Indicates a dSPACE real-time board, for example DS1006.																																		
	Indicates a VEOS.																																		
	Indicates a running real-time application.																																		
	Indicates a stopped real-time application.																																		
	Indicates a terminated real-time application.																																		
	Indicates a signal measurement board, a signal generation board, a bus board, an I/O module, or a bus module.																																		
	Indicates a channel group.																																		
	Indicates a channel.																																		

Related topics**References**

[Switch Controlbars \(AutomationDesk Basic Practices\)](#)

Properties (Platform/Device)

Access

This command is available only if a platform is selected. You can access it via:

Ribbon	None
Context menu of	Platform Manager – platform
Shortcut key	Enter
Icon	None

Purpose

To view the properties of the selected platform.

Result

The platform properties are displayed in the Properties pane. You can also change the properties.

Real-Time Application - Load

Access

This command is available only for the following platforms: DS1006, DS1104, MicroAutoBox, and Multiprocessor System. The platform hardware must be connected to the host PC. You can access the command via:

Ribbon	None
Context menu of	Platform Manager – platform
Shortcut key	None
Icon	
Others	Drag an application file from File Explorer to the platform in the Platform Manager

Purpose

To load a real-time application to the RAM of the selected hardware, and start it automatically.

Result	AutomationDesk opens a standard Open dialog that lets you select an application or variable description file. Depending on the hardware type, the file must be in the PPC/RTA/x86 or SDF file format.
Description	<p>The selected application is loaded to the RAM of the selected hardware. After downloading, the application is started automatically if the <code>simState</code> parameter is set to RUN (2). If the parameter is not set to RUN, you can visualize it in an instrument and change its value to start the application.</p> <p>Before the application is loaded, AutomationDesk checks whether an application is already running. If one is, you are prompted to stop it and load the new application.</p>

Real-Time Application / Offline Simulation Application - Load

Access	This command is available only for the following platforms: DS1007, DS1202 MicroLabBox, MicroAutoBox III, SCALEXIO, and VEOS. For DS1007, MicroLabBox, MicroAutoBox III, and SCALEXIO platforms, the platform hardware must be connected to the host PC. You can access the command via:
Ribbon	None
Context menu of	Platform Manager – platform
Shortcut key	None
Icon	
Others	Use the <i>right</i> mouse button to drag an application file from File Explorer to the platform in the Platform Manager.

Purpose	To load an application to the RAM of the selected hardware or to VEOS. After downloading, the application is not started.
Result	<p>A standard Open dialog is opened for you to select an application file or the corresponding variable description file:</p> <ul style="list-style-type: none"> ▪ DS1007, MicroLabBox, MicroAutoBox III, SCALEXIO: You can select an RTA or SDF file. ▪ VEOS: You can select an OSA or SDF file. <p>The selected application is loaded to the RAM of the selected hardware or to VEOS. After downloading, the application state is STOPPED.</p>

VEOS: The Platform Manager displays the loaded offline simulation application and the virtual ECU(s) and the (optional) environment VPU contained in the OSA file.

Description	Before the application is loaded, AutomationDesk checks whether an application is already running. If one is, you are prompted to stop it and load the new application.
--------------------	---

Real-Time Application / Offline Simulation Application - Load and Start

Access	This command is available only for the following platforms: DS1007, DS1202 MicroLabBox, MicroAutoBox III, SCALEXIO, and VEOS. For DS1007, MicroLabBox, MicroAutoBox III, and SCALEXIO platforms, the platform hardware must be connected to the host PC. You can access the command via:
Ribbon	None
Context menu of	Platform Manager – platform
Shortcut key	None
Icon	
Others	Drag an application file from File Explorer to the platform in the Platform Manager ¹⁾

¹⁾ If you use the *right* mouse button to drag the application file, a context menu is opened for you to select an action.

Purpose	To load an application to the RAM of the selected hardware or to VEOS, and start it automatically.
----------------	--

Result	<p>A standard Open dialog opens for you to select an application file or variable description file:</p> <ul style="list-style-type: none"> ▪ DS1007, MicroLabBox, MicroAutoBox III, SCALEXIO: The selected file must be in the RTA or SDF file format. ▪ VEOS: The selected file must be in the OSA or SDF file format. <p>The selected application is loaded to the RAM of the selected hardware or to VEOS. After downloading, the application is started automatically.</p> <p>VEOS: The Platform Manager displays the loaded offline simulation application and the virtual ECU(s) and the (optional) environment VPU contained in the OSA file.</p>
---------------	--

Description	Before the application is loaded, AutomationDesk checks whether an application is already loaded. If one is, you are prompted to unload it and load the new application.
--------------------	--

Real-Time Application - Load to Flash (DS1006/DS1104/MicroAutoBox II)

Access	This command is available only for the following platforms: DS1006, DS1104, MicroAutoBox, and Multiprocessor System. The platform hardware must be connected to the host PC. You can access the command via:
---------------	--

Ribbon	None
Context menu of	Platform Manager – platform
Shortcut key	None
Icon	 F
Others	Use the <i>right</i> mouse button to drag an application file from File Explorer to the platform in the Platform Manager.

Purpose	To load an application to the flash memory of the selected hardware, and start it automatically.
----------------	--

Result	AutomationDesk opens a standard Open dialog that lets you select an application. The application file does not need to be within an experiment. The application is loaded to the flash memory, copied to the RAM and then started.
---------------	---

Description	Before the application is loaded, AutomationDesk checks whether an application is already running. If one is, you are prompted to stop it and load the new application.
--------------------	---

Note

An application that was already in the RAM will be overwritten when you load an application to the flash memory.

The  platform state icon next to the platform icon in the Platform Manager indicates that the application that is running was started from the flash memory. If the platform is rebooted, the application in the flash memory is automatically started.

Real-Time Application - Load to Flash (DS1007/MicroLabBox/MicroAutoBox III/SCALEXIO)

Access

This command is available only for the following platforms: DS1007, DS1202 MicroLabBox, MicroAutoBox III, and SCALEXIO. The platform hardware must be connected to the host PC. You can access the command via:

Ribbon	None
Context menu of	Platform Manager – platform
Shortcut key	None
Icon	
Others	Use the <i>right</i> mouse button to drag an application file from File Explorer to the platform in the Platform Manager.

Purpose

To load an application to the flash memory of the selected hardware. After loading, the application is not started.

Note

- Relevant for DS1007 platforms: The command is available only for DS1007 systems registered as a single processor system. Loading an application to the flash memory of a DS1007 system consisting of two or more DS1007 PPC Processor Boards is not possible.
- Relevant for SCALEXIO platforms: The command is available only for SCALEXIO systems registered as a single processor system. For other SCALEXIO systems, loading an application to the flash memory is not possible.

Result

A standard Open dialog is opened for you to select an application file or the corresponding variable description file. The file must be in the RTA or SDF file format. After loading, the application state is STOPPED.

Description

Before the application is loaded, AutomationDesk checks whether an application is already running. If one is, you are prompted to stop it and load the new application.

Note

An application that was already in the RAM will be overwritten when you load an application to the flash memory.

The  platform state icon next to the application icon in the Platform Manager indicates that the application was loaded from the flash memory to the RAM. If the platform is rebooted, the application in the flash memory is automatically started.

Real-Time Application - Load to Flash and Start (DS1007/MicroLabBox/MicroAutoBox III/SCALEXIO)

Access

This command is available only for the following platforms: DS1007, DS1202 MicroLabBox, MicroAutoBox III, and SCALEXIO. The platform hardware must be connected to the host PC. You can access the command via:

Ribbon	None
Context menu of	Platform Manager – platform
Shortcut key	None
Icon	
Others	Use the <i>right</i> mouse button to drag an application file from File Explorer to the platform in the Platform Manager.

Purpose

To load an application to the flash memory of the selected hardware, and start it automatically.

Note

- Relevant for DS1007 platforms: The command is available only for DS1007 systems registered as a single processor system. Loading an application to the flash memory of a DS1007 system consisting of two or more DS1007 PPC Processor Boards is not possible.
- Relevant for SCALEXIO platforms: The command is available only for SCALEXIO systems registered as a single processor system. For other SCALEXIO systems, loading an application to the flash memory is not possible.

Result

AutomationDesk opens a standard Open dialog that lets you select an application. The application file does not need to be within an experiment.

The application is loaded to the flash memory, copied to the RAM and then started.

Description

Before the application is loaded, AutomationDesk checks whether an application is already running. If one is, you are prompted to stop it and load the new application.

Note

An application that was already in the RAM will be overwritten when you load an application to the flash memory.

The  platform state icon next to the application icon in the Platform Manager indicates that the application that is running was started from the flash memory. If the platform is rebooted, the application in the flash memory is automatically started.

Real-Time Application - Reload

Access

This command is available only for the following platforms: DS1006, DS1104, MicroAutoBox, and Multiprocessor System. An application must be currently loaded. You can access the command via:

Ribbon	None
Context menu of	Platform Manager – platform
Shortcut key	None
Icon	

Purpose

To reload the currently loaded application to the RAM.

Result

AutomationDesk reloads the application that belongs to the active SDF file of the selected platform to the RAM. The application always starts automatically after it is reloaded.

Refresh Interface Connections

Access

You can access the command via:

Ribbon	Platforms – Platform Management
Context menu of	None

Shortcut key	None
Icon	

Purpose To refresh the interface connections between AutomationDesk and the hardware.

Result

- DS1007, MicroAutoBox III, MicroLabBox, SCALEXIO, and VEOS: Using the command interrupts the connection to the platform temporarily.
- All other platforms: AutomationDesk refreshes the interface connections by resetting the device drivers of the platform connections (via bus interface or Ethernet) and reinitializing the **Platform Manager** with the information from the recent hardware configuration. The device drivers for bus connections are always reset, but the device drivers for network connections are reset only if at least one platform using the network connection is registered.

Refresh Platform Configuration

Access You can access this command via:

Ribbon	Platforms - Platform Management
Context menu of	Platform Manager
Shortcut key	None
Icon	

Purpose To refresh the hardware configuration.

Result The platform configurations are refreshed. The view of the structure shown in the **Platform Manager** is updated.

Register Platforms

Access You can access this command via:

Ribbon	Platforms - Platform Management
Context menu of	Platform Manager

Shortcut key	None
Icon	

Purpose To register dSPACE real-time hardware and VEOS.

Result AutomationDesk now recognizes the registered platform.

Description The registered platform is displayed in the Platform Manager. The registration data is stored in the recent hardware configuration. The Platform Manager can remember the configuration when AutomationDesk is restarted.

Whether AutomationDesk searches for connected and registered platforms during startup depends on the settings made on the Platform Management page of the AutomationDesk Properties dialog. Refer to [Platform Management Page \(AutomationDesk Basic Practices\)](#).

Tip

You do not need to register the DS1104 since it supports the plug & play feature.

Register Platforms dialog To specify the register settings for a single processor or controller board, a DS1006 based multiprocessor system, a MicroAutoBox or a SCALEXIO platform, and to get information on the platforms registered so far.

Platforms Lets you select the platform type being registered.

Platform properties Lets you view and specify the register settings for the platform. The available properties depend on the selected platform type.

Property	Description
Common Properties	
Multiprocessor type	(Available only for the Multiprocessor System platform) Displays the processor board type the selected Multiprocessor System platform is based on. The is "DS1006".
Platform name	(Available for the DS1007, the MicroAutoBox III, the SCALEXIO platform, and the Multiprocessor System platform) Lets you specify an unique name for the selected platform. After registration, the name is displayed in the Platform Manager. The valid characters are "a ... z", "A ... Z", "0 ... 9", "_", "-" and " ". The name must not start or end with an underline, hyphen or blank. If you do not specify a platform name, AutomationDesk displays a default name in the Platform Manager.
Platform type	Displays the type of the selected platform, for example, DS1006 Processor Board.
Topology check	(Available only for the Multiprocessor System platform) Lets you specify if AutomationDesk checks the topology of the selected DS1006-based multiprocessor system. If enabled, AutomationDesk checks if all

Property	Description
	<p>the processor boards of the system are interconnected via Gigalinks. AutomationDesk does <i>not</i> check whether the topology of the connected boards is compatible with the topology required by the real-time application to be loaded to the system, i.e., it does not check whether the correct Gigalink ports of the processor boards are used for interconnection.</p> <p>The topology check is performed:</p> <ul style="list-style-type: none"> ▪ When the multiprocessor system is connected ▪ When you load an application to the multiprocessor system
Connection Settings Properties	
Alias name	(Available for the DS1202 MicroLabBox, the MicroAutoBox III, the DS1007, the SCALEXIO, and the VEOS platform) Lets you specify the alias of the connection that is used for assignment.
Board name	(Available for the DS1202 MicroLabBox, the MicroAutoBox III, the DS1007, and the SCALEXIO platform) Lets you specify the board name used to identify the board or processing unit. You can also scan the local network for connected platforms. Depending on the platform type, click on the following edit field: <ul style="list-style-type: none"> ▪ For the DS1007 platform, Scan for available processor boards ▪ For the DS1202 MicroLabBox, MicroAutoBox III, and VEOS, Scan for available platforms ▪ For the SCALEXIO platform, Scan for available processing units For details on the opened Scan Local Network dialog, refer to Scan Local Network for Processor Boards/ Processing Units /Platforms dialog on page 330.
Connection parameter	(Available for the DS1202 MicroLabBox, the MicroAutoBox III, the DS1007, and the SCALEXIO platform) Lets you select the connection parameter to specify member processing units (MicroAutoBox IIISCALEXIO) or processor boards (DS1007). You can select one of the following connection parameters: <ul style="list-style-type: none"> ▪ Alias name ▪ Board name (Not available for VEOS) ▪ IP address ▪ MAC address (Not available for VEOS)
Connection type	(Not available for the DS1202 MicroLabBox, the DS1007, and the SCALEXIO platform) Lets you specify the connection type of the platform hardware. <ul style="list-style-type: none"> ▪ Select BUS if the platform hardware is installed in the host PC or in an expansion box connected to the host PC via a bus interface. ▪ Select NET if the platform hardware is connected to the host PC via Ethernet. For MicroAutoBox and VEOS, only the NET connection type is available.
IP address	(Available for the DS1202 MicroLabBox, the MicroAutoBox III, the DS1007, the SCALEXIO, and the VEOS platform) Lets you specify the network client for assignment. You can enter it or select it from the list of formerly used entries. You can also scan the local network for connected platforms. Depending on the platform type, click on the following edit field: <ul style="list-style-type: none"> ▪ For the DS1007 platform, Scan for available processor boards ▪ For the DS1202 MicroLabBox, MicroAutoBox III, and VEOS, Scan for available platforms ▪ For the SCALEXIO platform, Scan for available processing units For details on the opened Scan Local Network dialog, refer to Scan Local Network for Processor Boards/ Processing Units /Platforms dialog on page 330.
MAC address	(Available for the DS1202 MicroLabBox, the MicroAutoBox III, the DS1007, and the SCALEXIO platform) Lets you specify the MAC address of the SCALEXIO Processing Unit. It is used to identify the identical hardware. You can also scan the local network for connected platforms. Depending on the platform type, click on the following edit field: <ul style="list-style-type: none"> ▪ For the DS1007 platform, Scan for available processor boards ▪ For the DS1202 MicroLabBox, MicroAutoBox III, and VEOS, Scan for available platforms ▪ For the SCALEXIO platform, Scan for available processing units

Property	Description
Network client	For details on the opened Scan Local Network dialog, refer to Scan Local Network for Processor Boards/ Processing Units /Platforms dialog on page 330. (Available for the NET connection type of the DS1006; also available for the MicroAutoBox and VEOS)
Port address	Lets you specify the network client as an alias or IP address. (Not available for the DS1202 MicroLabBox, the DS1007, the SCALEXIO platform, and the MicroAutoBox) Lets you specify the base address of the board as specified with the DIP switches or the rotary switches on the board.
Multiprocessor Configuration Properties	
Processors	Lets you specify the number of processors belonging to the multiprocessor system. Click  to add a processor, or click  to delete the selected processor. The type of the board to be added (DS1006) depends on the Multiprocessor type property.
<p>Tip</p> <p>You should specify the maximum number of processors, since you cannot add members to a multiprocessor system that is already registered.</p>	
Processor name	Displays or lets you specify the name of the selected processor board. When you register a multiprocessor system, AutomationDesk specifies default processor names and board port addresses like this: MASTER, 0x300 (first board), SLAVE, 0x310 (second board), SLAVE_B, 0x320 (third board), SLAVE_C, 0x330 (fourth board), ... You should change the processor names according to the variable description to be used with the Multiprocessor System platform.
Port address	Lets you specify the base address of the board as specified with the DIP switches or the rotary switches on the board.

Note

If you register a DS1006-based multiprocessor system, the connection type and network client are specified for the multiprocessor system, so these settings are valid for all the processor boards belonging to the multiprocessor system. The port addresses are specified individually for the processor boards in the Multiprocessor configuration.

Register Lets you complete the registration. The registered platform is displayed together with the platform properties in the Registered platforms list. The registered platform is also displayed in the Platform Manager.

Registered platforms list Displays all the registered platforms with the following information: platform name, platform type, serial number/identifier, MAC address, network client, port address, and processor name.

You can customize the display in the Registered platforms list using the following commands available from the context menu of column headers:

- **Best Fit:** Lets you optimize the width of the selected column.
- **Best Fit (all columns):** Lets you optimize the widths of all columns according to the width of the editor or browser.
- **Column Chooser:** Lets you open a dialog for customizing the columns of the platforms list. To add a column to the list, drag it from the opened dialog to the list header. To remove a column from the list, drag its header to the dialog.

- **Sort Ascending:** Lets you sort the list alphabetically in ascending order according to the selected column.
- **Sort Descending:** Lets you sort the list alphabetically in descending order according to the selected column.

Scan Local Network for Processor Boards/ Processing Units /Platforms dialog

To scan the local network for connected platform hardware or simulators, and select one or more platforms or a simulator to register.

Type Lets you select the filter item type you want to use to filter the results list. If you select 'None', no filtering is applied.

Value Lets you enter a filter string.

Match whole word Lets you specify to search only for a matching pattern substring.

(Re)scan Lets you start a new scan process. AutomationDesk scans the subnetwork your host PC is connected to for connected processor boards/processing units/platforms matching the specified filter settings, and refreshes the results list.

List of available processor boards/processing units/platforms Displays all the processor boards, processing units and platforms that the specified filter found in the network during the scan process. The results list contains the IP address, MAC address, board name, system name and serial number for each processor board, processing unit or platform that was found. If the scan process is performed for VEOS, the results list contains the IP address and host name for each found simulator, together with the respective product version and installation path of the VEOS installation on the simulator.

To select a processor board, processing unit or platform for registration, click its entry and then press the  button. The selected element is moved to the list of selected processor boards/processing units/platforms, where you can transfer its connection parameter value to the Register Platforms dialog.

Tip

You can multiselect processing units and processor boards.

List of selected processor boards/processing units/platforms Displays all the processor boards, processing units or platforms selected for registration so far. When you click **Apply**, the listed platform hardware is assigned to the platform you want to register, and the connection parameter value of each list item is transferred to the Register Platforms dialog.

The following buttons are available to move elements from one list to the other:



Moves the selected element(s) from the Available processor boards/processing units/platforms list to the Selected processor boards/processing units/platforms list.

	Moves the selected element(s) from the Selected processor boards/processing units/platforms list to the Available processor boards/processing units/platforms list.
--	---

Apply Lets you confirm the selection of processor board(s), processing unit(s), or platform for registration. When you click this button, the connection parameter value of each element in the Selected processor boards, Selected processing units or Selected platforms list is stored in the Register Platforms dialog.

Related topics

References

Clear System.....	310
Manage Platforms.....	314

Reload

Access

This command is available only for the following platforms: DS1007, DS1202 MicroLabBox, MicroAutoBox III, SCALEXIO, and VEOS. An application must be loaded. You can access the command via:

Ribbon	None
Context menu of	Platform Manager – platform – application
Shortcut key	None
Icon	

Purpose

To reload the currently loaded application to the RAM. After reloading, the application is not started.

Result

AutomationDesk reloads the selected real-time application/offline simulation application on the selected platform/to VEOS. The reload process is executed to the RAM.

After reloading, the application state is STOPPED.

Reload and Start

Access

This command is available only for the following platforms: DS1007, DS1202 MicroLabBox, MicroAutoBox III, SCALEXIO, and VEOS. An application must be currently loaded. You can access the command via:

Ribbon	None
Context menu of	Platform Manager – platform – application
Shortcut key	None
Icon	

Purpose

To reload the currently loaded application to the RAM. After reloading, the application is started.

Result

The selected application is reloaded. The reload process is executed to the RAM. After reloading, the application state is **RUNNING**.

Real-Time Application - Reload to Flash

Access

This command is available only for the following platforms: DS1006, DS1104, MicroAutoBox. An application must be loaded. You can access the command via:

Ribbon	None
Context menu of	Platform Manager – platform
Shortcut key	None
Icon	

Purpose

To reload the currently loaded application to the flash memory of the selected hardware.

Result

AutomationDesk reloads the application that belongs to the active SDF file of the selected platform to the flash memory. The application always starts automatically after it is reloaded.

Note

An application that was already in the flash memory will be overwritten when you reload the application to the flash memory.

The  platform state icon next to the platform icon in the Platform Manager indicates that the running application was started from the flash memory

Related topics**Basics**

[Managing Platforms and Simulation Applications.....](#) 20

Reload to Flash

Access

This command is available only for the following platforms: DS1007, DS1202 MicroLabBox, MicroAutoBox III, and SCALEXIO. An application must be loaded. You can access the command via:

Ribbon	None
Context menu of	Platform Manager – platform – application
Shortcut key	None
Icon	

Purpose

To reload the currently loaded application to the flash memory. After reloading, the application is not started.

Result

AutomationDesk reloads the selected real-time application on the selected platform. The reload process is executed to the flash memory.

After reloading, the application state is **StoppedFromFlash**.

Note

An application that was already in the flash memory will be overwritten if you reload the application to the flash memory.

The  platform state icon next to the application icon in the Platform Manager indicates that the running application was loaded from the flash memory.

Related topics**Basics**

[Managing Platforms and Simulation Applications.....](#) 20

Reload to Flash and Start

Access

This command is available only for the following platforms: DS1007, DS1202 MicroLabBox, MicroAutoBox III, and SCALEXIO. An application must be currently loaded. You can access the command via:

Ribbon	None
Context menu of	Platform Manager – platform – application
Shortcut key	None
Icon	

Purpose

To reload the currently loaded application to the flash memory. After reloading, the application is started.

Result

The selected application is reloaded on the selected platform. The reload process is executed to the flash memory.

After reloading, the application state is **RunningFromFlash**.

Note

An application that was already in the flash memory will be overwritten when you reload the application to the flash memory.

The  platform state icon next to the application icon in the Platform Manager indicates that the application that is running was started from the flash memory.

Related topics**Basics**

[Managing Platforms and Simulation Applications.....](#) 20

Show Connected Clients

Access

You can access this command via:

Ribbon	None
Context menu of	<ul style="list-style-type: none"> ▪ Platform Manager – SCALEXIO platform ▪ Platform Manager – Processing units of a SCALEXIO platform
Shortcut key	None
Icon	None

Purpose

To display the clients to which the selected SCALEXIO platform or SCALEXIO Processing Unit is connected.

Connected Client Overview dialog

To get details on the clients that are currently connected to the selected SCALEXIO platform or SCALEXIO processing unit.

For each processing unit, the Connected Client Overview dialog displays all the client processes that access the unit.

Host Name Displays the host name of the client that is connected to the selected processing unit.

IP Address Displays the IP address of the client that is connected to the selected processing unit.

User Name Displays the user name of the client that is connected to the selected processing unit.

Connection Time Displays the time when the client connection to the selected processing unit was established.

Process Name Displays the name of the client process that accesses the selected processing unit.

Refresh Lets you update the display of the client processes.

Single Step

Access

This command is available only for the VEOS platform, and if an application is currently pausing or stopped. You can access this command via:

Ribbon	None
Context menu of	Platform Manager – VEOS platform – application

Shortcut key	None
Icon	None

Purpose

To run a pausing or stopped offline simulation stepwise.

Result

The next step of an offline simulation is executed. You can run all the steps separately in consecutive order.

Description

You can use this command to inspect an offline simulation in detail. Each time you use the command, the next simulation step is executed.

Related topics**References**

Pause.....	317
Real-Time Application / Offline Simulation Application - Load.....	320
Real-Time Application / Offline Simulation Application - Load and Start.....	321
Start.....	336
Stop.....	337

Start

Access

This command is available only for the following platforms: DS1007, DS1202 MicroLabBox, MicroAutoBox III, SCALEXIO, and VEOS, and if an application is loaded. You can access this command via:

Ribbon	None
Context menu of	Platform Manager – platform – application
Shortcut key	None
Icon	None

Purpose

To start the selected application.

Description

AutomationDesk starts the selected application loaded on the platform.

Relevant for VEOS: An offline simulation starts from the beginning, or a pausing offline simulation continues.

Stop

Access

This command is available only for the following platforms: DS1007, DS1202 MicroLabBox, MicroAutoBox III, SCALEXIO, and VEOS, and if an application is currently running. You can access this command via:

Ribbon	None
Context menu of	Platform Manager – platform – application
Shortcut key	None
Icon	None

Purpose

To stop the selected application.

Description

AutomationDesk stops the selected application running on the platform.

The application is not unloaded automatically when it is stopped. If you want to unload it, choose Unload from the application's context menu.

Stop RTP

Access

This command is available only for the following platforms: DS1006, DS1104, and MicroAutoBox. You can access this command via:

Ribbon	None
Context menu of	Platform Manager – platform
Shortcut key	None
Icon	None

Purpose

To stop the application running on the selected platform.

Description

AutomationDesk stops the real-time application on the currently selected platform.

Stop RTPs

Access This command is available only for the Multiprocessor System platform. You can access this command via:

Ribbon	None
Context menu of	Platform Manager – platform
Shortcut key	None
Icon	None

Purpose To stop the applications running on the selected Multiprocessor System platform.

Description AutomationDesk stops the real-time applications on the currently selected Multiprocessor System platform.

Unload

Access This command is available only for the following platforms: DS1007, DS1202 MicroLabBox, MicroAutoBox III, SCALEXIO, and VEOS. An application must be loaded. You can access this command via:

Ribbon	None
Context menu of	Platform Manager – platform – application
Shortcut key	None
Icon	

Purpose To unload the selected application.

Description AutomationDesk unloads the application from the memory of the selected platform.

Update Firmware

Access

This command is available only for dSPACE real-time platforms. You can access this command via:

Ribbon	None
Context menu of	Platform Manager – platform
Shortcut key	None
Icon	None

Purpose

To update the firmware of the selected platform.

Result

The firmware is updated to the latest firmware version.

Note

- It is recommended to perform the Refresh Interface Connections command after updating the firmware.
- After updating the MicroAutoBox II firmware, you have to turn off the MicroAutoBox II. The firmware changes take effect after restart.

Description

AutomationDesk checks whether the AutomationDesk installation contains appropriate firmware which is later than the firmware that is currently installed on the hardware. If a later version is available, the firmware is updated.

The Update Firmware Wizard opens for updating or repairing firmware components for the following real-time platforms:

- DS1006
- DS1007
- DS1104
- MicroAutoBox II
- MicroAutoBox III
- MicroLabBox
- SCALEXIO

For the other platforms the dialog settings vary or you have to use another tool.

- RapidPro system:
[How to Update RapidPro Firmware \(RapidPro System Hardware Installation Guide\)](#)
- DS1552 Multi-I/O Module of a MicroAutoBox II:
[DS1401UpdateExtIO](#)

The Update Firmware Wizard provides the following dialogs to configure and start a firmware update.

Specifics for SCALEXIO platforms To ensure real-time applications are downloaded without restrictions or side effects, and with proper functionality, the following conditions must be fulfilled:

- All the processing units/processor boards of a SCALEXIO platform contain the firmware version that is available in the current RCP and HIL software installation.
- All the components of a SCALEXIO system (for example, real-time PC and I/O boards) contain the same firmware version.

Firmware version deviations for a SCALEXIO platform are indicated in the Platform Manager. The affected hardware components are marked by the  symbol. You can find associated warning messages in the Message Viewer.

Note

Note the following restrictions when you use a SCALEXIO system:

- The Update Firmware Wizard supports SCALEXIO systems as of dSPACE Release 2015-B. If you want to update the firmware version on a SCALEXIO system to an earlier version, you have to use ControlDesk or ConfigurationDesk from an earlier dSPACE Release.
- If your SCALEXIO system contains a DS2655M2 Digital I/O Module, a firmware update from firmware version 3.2 or earlier to a firmware version 3.3 or later, might lead to an error and the update process is then stopped.

To finish the firmware update you have to do the following steps:

- Restart the SCALEXIO system.
- Call the Refresh Interface Connections command in the Platform Manager.
- Repeat the firmware update process.
- If your SCALEXIO system contains one or more new hardware components that are not already supported by the currently active firmware on the SCALEXIO Processing Unit or DS6001 Processor Board, a firmware update might lead to an error and the update process is then aborted.

To finish the firmware update, perform the following steps:

- Restart the SCALEXIO system.
- Call the Refresh Interface Connections command in the Platform Manager.
- Repeat the firmware update process.

Select Mode dialog

Lets you select the firmware update mode.

- **Firmware update mode**

The firmware handling process is configured for updating all firmware components of the real-time hardware to a later version.

- **Firmware repair mode**

The firmware handling process is configured for repairing the selected firmware components of the real-time hardware by reloading the same firmware versions.

Next > Opens the next dialog.

OK Closes the Update Firmware Wizard without starting a firmware update.

Select Firmware Archive dialog

To select the firmware archive to be loaded.

Currently selected archive Displays the selected firmware archive. If the firmware archives are installed on the default installation path, either the latest firmware archive for the registered and selected real-time hardware is displayed, or the firmware archive that you specified manually via the **Browse for archive** button.

Archive selection Lets you select the firmware archive to be used for firmware update.

- **Select an archive from installation**

By default, the latest firmware archive is selected that corresponds to the active dSPACE Release installation.

- **Select an archive from file system**

Lets you select a firmware archive in another version or from another path.

< Back Opens the previous dialog to change the update mode.

Next > Opens the next dialog.

OK Closes the Update Firmware Wizard without starting a firmware update.

Select Firmware Components dialog

To start the prepared firmware update.

Name Displays the names of the selected platform, its hardware components, and the related firmware components.

Current FW Displays the firmware versions currently loaded to the listed hardware components.

Available FW Displays the firmware versions available in the specified firmware archive.

Update

- In firmware update mode, the firmware components to be updated are marked. You cannot modify the selection.
- In firmware repair mode, you have to select at least one firmware component to enable the repair process.

Status Displays the status of the firmware update process. If the firmware component does not provide progress information, only the states 50% and 100% are displayed. If the update process finished successfully, the status is set to OK.

Update/Repair

- In firmware update mode, the Update button is enabled to start a firmware update process if later firmware versions are available in the firmware archive than those currently loaded to the hardware.
- In firmware repair mode, the Repair button is enabled to start a firmware repair process, if the firmware versions of the loaded firmware components and the versions of the specified firmware archive are identical, and if you have selected at least one firmware component in the Update column to be repaired.

< Back Opens the previous dialog to change the selected firmware archive.

OK Closes the Update Firmware Wizard without starting a firmware update.

Related topics

Basics

[Basics on the Firmware Manager \(Firmware Manager Manual\)](#)

XIL API Framework

Introduction

AutomationDesk provides commands for working with an XIL API Framework.

Where to go from here

Information in this section

[Configure \(XIL API Framework\)](#)..... 343

To specify which XIL API Framework implementation and configuration to work with.

[Disable Framework Support](#)..... 344

To let the automation block use the project-specific Mapping data object to access variables.

[Edit \(XIL API Framework\)](#)..... 345

To edit the configuration of the XIL API Framework.

[Enable Framework Support](#)..... 346

To let the automation block use the XIL API Framework to access variables.

Initialize (XIL API Framework)	347
To initialize the configured XIL API Framework.	
Insert (CheckValues)	348
To insert a CheckValues automation block of the XIL API library via the Home ribbon.	
Insert (GetValues)	348
To insert a GetValues automation block of the XIL API library via the Home ribbon.	
Insert (SetValues)	349
To insert a SetValues automation block of the XIL API library via the Home ribbon.	
Shutdown (XIL API Framework)	350
To shut down the active sXIL API Framework.	

Configure (XIL API Framework)

Access

You can access this command via:

Ribbon	Platforms – Framework
Context menu of	None
Shortcut key	None
Icon	

Purpose

To specify which XIL API Framework implementation and configuration to work with.

Result

The XIL API Framework to be initialized or to be shut down is specified.

Description

This command lets you specify the XIL API Framework that you can initialize later by using the **Initialize** command or shut down by using the **Shutdown** command.

Dialog settings

The dialog provides the following settings:

Framework implementation Lets you select the implementation of the XIL API Framework to work with.

Framework configuration Lets you specify the XML file that contains the configuration of the framework you want to work with in the current AutomationDesk session.

Initialize framework on startup Lets you select whether to initialize the XIL API Framework when the AutomationDesk session starts.

Related topics

HowTos

How to Create a Framework Configuration.....	59
How to Initialize an XIL API Framework.....	68

References

Initialize (XIL API Framework).....	347
Shutdown (XIL API Framework).....	350

Disable Framework Support

Access

You can access this command via:

Ribbon	None
Context menu of	The following automation blocks: <ul style="list-style-type: none">▪ CheckValues▪ GetValues▪ SetValues
Shortcut key	None
Icon	None

Purpose

To let the automation block use the project-specific Mapping data object to access variables.

Result

The automation block accesses variables by using the mapping that is configured in the Mapping data object at the top level of the AutomationDesk project.

Description

This command adds an MAPort data object and a Variant data object that references the project-specific Mapping data object to the automation block. Both are configured to auto-reference data objects at a higher hierarchy level.

You can use the Enable Framework Support command to return to the default behavior of the block, i.e., to let it access the XIL API Framework.

Related topics**HowTos**

How to Write and Read Variables.....	129
--	-----

References

CheckValues.....	156
Enable Framework Support.....	346
GetValues.....	157
SetValues.....	158

Edit (XIL API Framework)

Access

You can access this command via:

Ribbon	Platforms – Framework
Context menu of	None
Shortcut key	None
Icon	

Purpose

To edit the configuration of the XIL API Framework.

Result

The current framework configuration files are opened in the **Mapping Editor**.

Description

The **Mapping Editor** opens the framework configuration file (XML) that you specified via **Configure**. Refer to [Configure \(XIL API Framework\)](#) on page 343.

Additionally, the port configuration files and the mapping files that are configured in the framework configuration file are opened.

Related topics	Basics
	Basics on the Mapping Editor (AutomationDesk Basic Practices)
HowTos	How to Create a Framework Configuration..... 59
References	Mapping Editor (AutomationDesk Basic Practices)

Enable Framework Support

Access	You can access this command via:
Ribbon	None
Context menu of	The following automation blocks: <ul style="list-style-type: none">▪ CheckValues▪ GetValues▪ SetValues
Shortcut key	None
Icon	None
Purpose	To let the automation block use the XIL API Framework to access variables.
Result	The automation block accesses variables by using the mapping that is configured in the framework configuration file.
Description	<p>This command lets you remove the MAPort data object and the Variant data object that references the project-specific Mapping data object from the automation block. These data objects were added before by the Disable Framework Support command.</p> <p>The automation block returns to the default behavior, i.e., it uses the XIL API Framework for variable access.</p>

Related topics**HowTos**

How to Write and Read Variables.....	129
--	-----

References

CheckValues.....	156
Disable Framework Support.....	344
GetValues.....	157
SetValues.....	158

Initialize (XIL API Framework)

Access

You can access this command via:

Ribbon	Platforms – Framework
Context menu of	None
Shortcut key	None
Icon	

Purpose

To initialize the configured XIL API Framework.

Result

The XIL API Framework is initialized.

Description

This command initializes the XIL API Framework with the configuration that you specified via the Configure command.

The defined ports are started and the variable mapping is displayed in the Mapping Viewer. The first line in the Mapping Viewer shows the name of the current framework configuration file.

If a project contains a Mapping data object, the data object's icon turns green () to display that it is not relevant for variable mapping.

When the XIL API Framework is initialized, you can parameterize some data objects that are provided by the XIL API library by selecting valid values in a customized edit dialog.

You can terminate the XIL API Framework by using the Shutdown command.

Related topics**HowTos**

How to Create a Framework Configuration.....	59
How to Initialize an XIL API Framework.....	68

References

Configure (XIL API Framework).....	343
Shutdown (XIL API Framework).....	350

Insert (CheckValues)

Access

You can access this command via:

Ribbon	Home - Insert
Context menu of	None
Shortcut key	None
Icon	

Purpose

To insert a CheckValues automation block of the XIL API library.

Description

This command lets you add a CheckValues automation block to a sequence that is open in the Sequence Builder. If no block in the sequence is selected, the new block is added at the end of the sequence. Otherwise, it is added to the selected block.

Related topics**References**

CheckValues.....	156
----------------------------------	-----

Insert (GetValues)

Access

You can access this command via:

Ribbon	Home - Insert
Context menu of	None

Shortcut key	None
Icon	

Purpose

To insert a **GetValues** automation block of the XIL API library.

Description

This command lets you add a **GetValues** automation block to a sequence that is open in the Sequence Builder. If no block in the sequence is selected, the new block is added at the end of the sequence. Otherwise, it is added to the selected block.

Related topics**References**

GetValues	157
---------------------------------	-----

Insert (SetValues)

Access

You can access this command via:

Ribbon	Home - Insert
Context menu of	None
Shortcut key	None
Icon	

Purpose

To insert a **SetValues** automation block of the XIL API library via the Home - Insert ribbon.

Description

This command lets you add a **SetValues** automation block to a sequence that is open in the Sequence Builder. If no block in the sequence is selected, the new block is added at the end of the sequence. Otherwise, it is added to the selected block.

Related topics**References**

SetValues	158
---------------------------------	-----

Shutdown (XIL API Framework)

Access	You can access this command via:		
Ribbon	Platforms – Framework		
Context menu of	None		
Shortcut key	None		
Icon			
Purpose	To shut down the active XIL API Framework.		
Result	The configured framework is terminated.		
Description	<p>This command terminates the XIL API Framework so that you cannot use the framework's configuration of variable access for parameterizing automation blocks.</p> <p>The variable mapping displayed in the Mapping Viewer is cleared.</p> <p>If a project contains a Mapping data object, the data object's icon turns yellow to display that it is relevant for XIL API automation blocks (again).</p>		
Related topics	References		
	<table border="1"><tr><td>Initialize (XIL API Framework).....</td><td>347</td></tr></table>	Initialize (XIL API Framework)	347
Initialize (XIL API Framework)	347		

XIL API Convenience

Introduction	AutomationDesk provides commands for working with the XIL API Convenience library.				
Where to go from here	Information in this section				
	<table border="1"><tr><td>Edit (TaskName).....</td><td>351</td></tr><tr><td colspan="2">To specify the task name as a string.</td></tr></table>	Edit (TaskName)	351	To specify the task name as a string.	
Edit (TaskName)	351				
To specify the task name as a string.					

[Edit \(Variables\).....](#) 352

To specify a list of variable names.

[Edit \(Vendor\).....](#) 354

To specify the vendor of the XIL API implementation as a string.

Edit (TaskName)

Access

You can access this command via:

Ribbon	None
Context menu of	<ul style="list-style-type: none"> ▪ TaskName data objects. Their template is provided by the XIL API Convenience library. ▪ The TaskName data object of an InitializeCapture block.
Shortcut key	None
Icon	None

Purpose

To specify the task name as a string.

Result

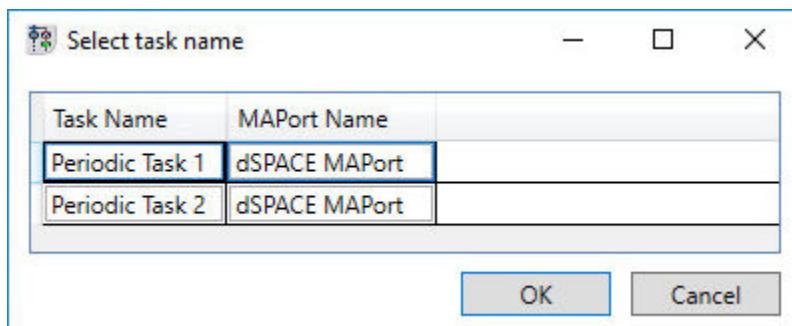
If an XIL API Framework is initialized, the Select Task Name dialog opens and lets you select the name from the list of tasks that are configured in the XIL API Framework.

If you work with the XIL API Testbench, the Value Editor opens to let you specify the name manually.

Dialog settings

The Select Task Name dialog provides the following settings:

Table of tasks You can specify a task by clicking it in the table of the tasks that are configured in the XIL API Framework. The selected task is framed in blue, as shown in the following illustration.



OK The selected task name is assigned to the TaskName data object.

Cancel The TaskName data object is left unchanged.

Related topics

References

TaskName (Data Object).....	159
-----------------------------	-----

Edit (Variables)

Access

You can access this command via:

Ribbon	None
Context menu of	<ul style="list-style-type: none"> ▪ Variables data objects. Their template is provided by the XIL API Convenience library. ▪ The Variables data object of an InitializeCapture block.
Shortcut key	None
Icon	None

Purpose

To specify a list of variable names.

Result

If an XIL API Framework is initialized, the Select Variables dialog opens and lets you specify a list of variable names by selecting them from the list of variables that are configured in the XIL API Framework.

If you work with the XIL API Testbench the Value Editor opens to let you specify the list manually.

Description

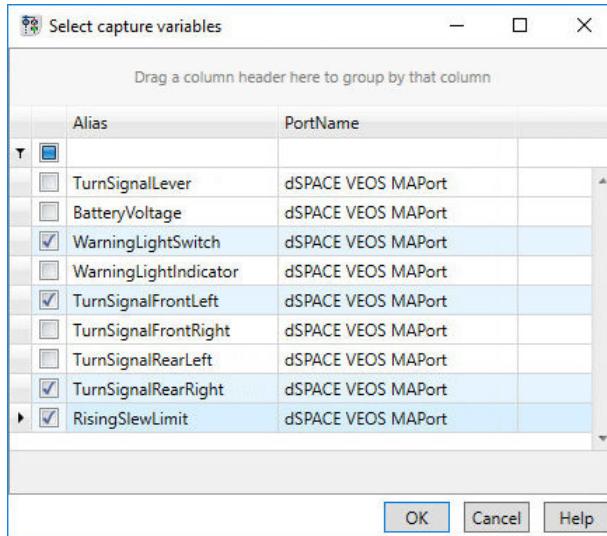
You can select a variable by selecting the checkbox next to its alias.

Dialog settings

The Select Variables dialog provides the following settings:

Filter definition row An additional row of edit fields is displayed below the table headers where you can specify column-specific filter expressions. You can click the checkbox in this row to switch between the following check-state-related filters:

- Show only checked rows.
- Show only unchecked rows.
- Show rows of all check states.



OK A list of the selected variable aliases is assigned to the Variables data object.

Cancel The Variables data object is left unchanged.

Column header commands

The following commands concerning the presentation of the list are available via the context menu of the column headers:

Sort Ascending Lets you sort the list alphabetically in ascending order according to the selected column.

Sort Descending Lets you sort the list alphabetically in descending order according to the selected column.

Clear Sorting Lets you display the list in unsorted order.

Group By This Column Lets you group the list according to the values in the selected column.

Show/Hide Group Panel Lets you display or hide a panel above the list headers. You can group the list by more than one column by dragging the column headers to the Group panel.

Full Expand Lets you completely expand all groups of a grouped list.

Full Collapse Lets you collapse all entries in a grouped list to the top-level groups.

Clear Grouping Lets you display the list without groups.

Show/Hide Column Chooser Lets you display or hide the dialog that lists the hidden columns. You can add the columns to the display by dragging them to the header of the list. You can hide columns from the list by dragging their header to the Column Chooser.

Best Fit (all columns) Lets you optimize the widths of all columns to fit the width of the list.

Filter Editor Lets you specify a filter condition for the list. For more information, refer to [Edit Filter \(AutomationDesk Basic Practices\)](#).

Show Search Panel Lets you display/hide an edit field above the column headers where you can enter a search string for the list. Matching entries are displayed and highlighted in yellow. Click Close to hide the panel again.

Related topics

References

Variables (Data Object).....	159
------------------------------	-----

Edit (Vendor)

Access

You can access this command via:

Ribbon	None
Context menu of	<ul style="list-style-type: none"> ▪ Vendor data objects. Their template is provided by the XIL API Convenience library. ▪ The Vendor data object of an InitMAPort block.
Shortcut key	None
Icon	None

Purpose

To specify the vendor of the XIL API implementation as a string.

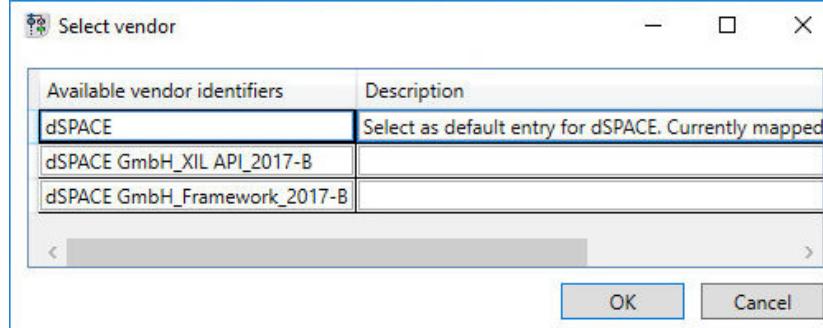
Result

The Select Vendor dialog opens and lets you select the name from the list of vendors of the XIL API implementations that are installed on the host PC.

Dialog settings

The Select Vendor dialog provides the following settings:

Table of vendors You can specify a vendor by selecting it in the table of the vendors of the installed XIL API implementations. The selected vendor is framed in blue, as shown in the following illustration.



OK The selected vendor identifier is assigned to the Vendor data object.

Cancel The Vendor data object is left unchanged.

Related topics**References**

Vendor (Data Object).....	160
---------------------------	-----

XIL API

Introduction

AutomationDesk's XIL API library provides specific commands and dialogs.

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

Edit (CaptureState).....	356
--------------------------	-----

To configure the enumeration type (Enum) of the CaptureState data object.

Edit (DataType).....	357
----------------------	-----

To configure the enumeration type (Enum) of the DataType data object.

Edit (MAPortConfiguration)	358
To configure the MAPort data object for platform access.	
Edit as Dictionary	360
To configure platform access via any implementation of the ASAM XIL API standard within your project.	
Export XIL API Mapping	361
To export the variable mapping to an XML file.	
Import XIL API Mapping	362
To import the variable mapping from an XML file.	
Insert (XIL API Elements)	363
To insert data objects of the XIL API library using the Home ribbon.	
Release Capture	364
To release a Capture instance.	
Release MAPort	365
To release an MAPort instance.	

Edit (CaptureState)

Access

You can access this command via:

Ribbon	None
Context menu of	CaptureState data object
Shortcut key	None
Icon	None
Others	Double-click a CaptureState data object

Purpose

To configure the enumeration type (Enum) of the CaptureState data object.

Result

The properties are assigned to the CaptureState data object.

Description

You can set the capture state to different enumeration types (Enums) via a drop down list. For details, refer to CaptureState (Data Object).

Dialog settings

Capture State The CaptureState object can take any of the following enumeration types (Enums).

CaptureState	Description
eCONFIGURED	Initial state before capturing is started and after it is stopped.
eACTIVATED	Capturing is started and waits for a start trigger.
eRUNNING	Measurement is running.
eFINISHED	Measurement is stopped via a stop trigger.

Tip

You can find a capturing state diagram ("state diagram of capturing") in the ASAM documentation [ASAM_AE_XIL_Generic-Simulator-Interface_BS-1-4-Programmers-Guide_V2-1-0.pdf](#).

You can get information on the state also via:

```
print(_AD_.Capture.GetState().__doc__)
```

Related topics**References**

CaptureResult (Data Object)	223
CaptureState (Data Object)	234
GetState	241
XIL API (Model Access)	207

Edit (DataType)

Access

You can access this command via:

Ribbon	None
Context menu of	DataType data object
Shortcut key	None
Icon	None
Others	Double-click on a DataType data object

Purpose

To configure the enumeration type (Enum) of the DataType data object.

Result

The properties are assigned to the DataType data object.

Description

You can set the data type to different enumeration types (Enums) via a drop down list. For details, refer to DataType (Data Object).

Related topics**References**

CaptureResult (Data Object).....	223
DataType (Data Object).....	275
XIL API (Model Access).....	207

Edit (MAPortConfiguration)

Access

You can access this command via:

Ribbon	None
Context menu of	MAPortConfiguration data object
Shortcut key	None
Icon	None
Others	Double-click an MAPortConfiguration data object

Purpose

To configure the MAPort data object for platform access.

Result

The selected MAPortConfiguration data object is parameterized.

Description

This command lets you conveniently configure access to a platform if you use the dSPACE implementation of the ASAM XIL API standard.

If an XIL API Framework is initialized, the Select MAPort dialog opens and you can select a model port from the list of ports that are known in the framework.

If you work with the XIL API Testbench, the Variables pane is opened and you can parameterize the selected MAPortConfiguration data object with a platform name and a variable description file.

Note

If you want to access a platform via the ASAM XIL API implementation of another vendor, you must parameterize the MAPortConfiguration data object using the **Edit as Dictionary** command.

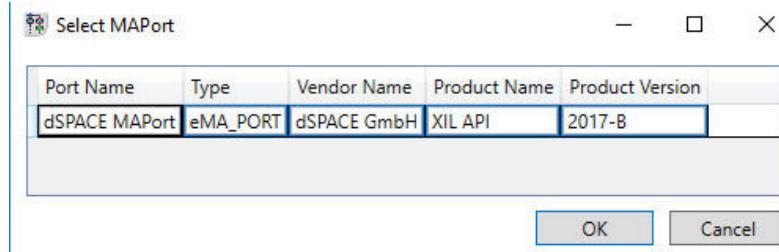
Tip

If you shut down an XIL API Framework, you must clear the MAPortConfiguration data object via **Clear Value (AutomationDesk Basic Practices)** before you can configure it again in the Variables Controlbar.

Select MAPort

The Select MAPort dialog provides the following settings:

Table of ports You can specify a port by selecting it in the table of the ports that are configured in the XIL API Framework. The selected port is framed in blue, as shown in the following illustration.



OK The selected port name is assigned to the MAPortConfiguration data object.

Cancel The MAPortConfiguration data object is left unchanged.

Variables pane

The Variables pane provides the following settings:

Platform name Lets you specify the platform name. An already registered platform can be selected from a list providing its name. If no platform is registered, the list only provides the default names of the platforms. Additionally, you can add any valid platform name to the list.

Variable file Lets you enter the path and file name of your application's variable file or click the button to open the standard Choose a file dialog. You can choose between the following file types:

- SDF file (*.sdf)
- TRC file (*.trc)
- Variable files (*.sdf, *.trc)
- All files (*.*)

Absolute path Lets you specify the path as an absolute path in the project.

Relative path Lets you specify the path as a relative path in the project. It is then a shortened path relating to the AutomationDesk project file. The path will not be changed to a relative path if the project and the specified file are saved to different drives.

Show Tree Displays the contents of the variable file in the Variables pane. Every entry takes effect immediately if you select a new item or close the dialog (by clicking the Close button in the title bar).

Related topics

References

Edit as Dictionary.....	360
MAPortConfiguration (Data Object).....	289
Variables Pane (AutomationDesk Basic Practices)	

Edit as Dictionary

Access

You can access this command via:

Ribbon	None
Context menu of	MAPortConfiguration data object
Shortcut key	None
Icon	None

Purpose

To configure the MAPort data object for platform access.

Result

The selected MAPortConfiguration data object is parameterized according to the respective implementation of the ASAM XIL API standard.

Description

This command lets you freely configure the MAPort data object, for example, when you use a vendor implementation different to the dSPACE implementation. For details, refer to MAPortConfiguration (Data Object).

The MAPortConfiguration data object is opened as a Dictionary data object in the Value Editor.

You can specify the configuration as pairs that each consists of a configuration parameter name and its value.

By default, an MAPortConfiguration data object is initialized with a configuration according to the dSPACE implementation of the ASAM XIL API standard.

- In the edit field of the **ApplicationPath** value, you can specify the variable description file with its absolute path or with its path relative to the folder where the AutomationDesk project is located.
- In the edit field of the **PlatformIdentifier** value, you can specify the platform name.

If you use the dSPACE implementation of the ASAM XIL API standard, you can use the **Edit (MAPortConfiguration)** command to configure the MAPort object in a more convenient way.

Related topics

References

Edit (Dictionary) (AutomationDesk Basic Practices)	358
Edit (MAPortConfiguration).....	358
MAPortConfiguration (Data Object).....	289

Export XIL API Mapping

Access

You can access this command via:

Ribbon	None
Context menu of	<ul style="list-style-type: none"> ▪ Mapping data object ▪ DataContainer data object ▪ Dictionary data object ▪ MAPortConfiguration data object
Shortcut key	None
Icon	None

Purpose

To export the variable mapping to an XML file.

Result

The contents of the variable mapping are written to the specified XML file.

Description

This command lets you export the mapping of aliases and model paths to an XML file. You can import the XML file to other projects later via [Import XIL API Mapping](#).

Export Mapping dialog

- Save in** Lets you select the path and folder to export the file to.
- File name** Lets you specify the name of the file. If you select a file from those listed under the selected path and folder, you are prompted before it is overwritten.
- Save as type** Lets you select the type to save the file as. Only the XML file type is available.

Related topics**References**

Import XIL API Mapping.....	362
-----------------------------	-----

Import XIL API Mapping

Access

You can access this command via:

Ribbon	None
Context menu of	▪ Mapping data object in the Project Manager
Shortcut key	None
Icon	None

Purpose

To import the variable mapping from an XML file.

Result

The contents of the specified XML file are imported.

Description

This command lets you import an XML file that you created before by using Export XIL API Mapping.

Import Mapping dialog

- Look in** Lets you select the path and folder to import the file from.
- File name** Lets you select the name of the file to import.
- Files of type** Lets you select the type of file to import. Only the XML file type is available.

Related topics**References**

Export XIL API Mapping	361
--	-----

Insert (XIL API Elements)

Access

You can access the data objects of the XIL API library via:

Ribbon	Home - Insert - Data Objects
Context menu of	None
Shortcut key	None
Icon	<ul style="list-style-type: none"> ■ MAPort ■ Capture ■ CaptureState ■ Watcher ■ CaptureResult ■ DataType ■ BaseValue ■ SignalValue ■ SignalGroupValue ■ SignalGenerator ■ SignalDescriptionSet ■ MAPortConfiguration ■ Mapping ■ EESPort

Purpose

To insert data objects of the XIL API library using the Home ribbon.

Description

The ribbon commands are enabled or disabled according to context.

You can add data objects to your project in the Project Manager and to specific automation blocks in the Sequence Builder, such as Exec automation blocks. The data object is added to the selected element.

Related topics**References**

XIL API (Model Access).....	207
-----------------------------	-----

Release Capture

Access

You can access this command via:

Ribbon	None
Context menu of	An initialized Capture data object
Shortcut key	None
Icon	None

Purpose

To release a Capture instance.

Result

The selected Capture data object is released and can be initialized again, for example, via an `InitializeCapture` block.

Description

This command lets you manually release a previously initialized Capture data object.

This command is available only for initialized Capture data objects.

Tip

You can automate the release of a Capture data object, for example, via a `ReleaseCapture` block.

Related topics**HowTos**

How to Release Resources After Data Capturing.....	105
--	-----

References

Capture (Data Object).....	231
InitializeCapture.....	186
ReleaseCapture.....	188

Release MAPort

Access

You can access this command via:

Ribbon	None
Context menu of	An initialized MAPort data object
Shortcut key	None
Icon	None

Purpose

To release an MAPort instance.

Result

The selected MAPort data object is released and can be initialized again, for example, via an InitMAPort block.

Description

This command lets you manually release a previously initialized MAPort data object. This is required, for example, if you changed the related MAPortConfiguration to work with another platform or to load another simulation application.

This command is available only for initialized MAPort data objects. The command is not available for MAPort data objects that are automatically created by the XIL API Framework.

Tip

You can automate the release of an MAPort data object, for example, via a ReleaseMAPort block.

Related topics

HowTos

[How to Build a Basic Sequence for Accessing Simulator Variables.....](#) 78

References

InitMAPort.....	295
InitMAPort.....	162
MAPort (Data Object).....	286
MAPortConfiguration (Data Object).....	289
ReleaseMAPort.....	300
ReleaseMAPort.....	167

Automation

Basics on Automating Simulation Platform Access

Introduction

AutomationDesk provides a COM-based API to automate the handling of AutomationDesk.

Related information

The AutomationDesk COM API provides specific objects for configuring platform access using the XIL API library.

Framework handling:

- [Framework \(Object\) \(AutomationDesk Automation\)](#)
- [FrameworkConfiguration \(AutomationDesk Automation\)](#)

Testbench handling:

- [Mapping \(Object\) \(AutomationDesk Automation\)](#)
- [PortConfig \(AutomationDesk Automation\)](#)
- [Testbench \(AutomationDesk Automation\)](#)
- [TestbenchFactory \(AutomationDesk Automation\)](#)

MAPort handling:

- [MAPort \(AutomationDesk Automation\)](#)
- [MAPortConfiguration \(AutomationDesk Automation\)](#)
- [MAPortFactory \(AutomationDesk Automation\)](#)

Variable handling for reading and writing:

- [BaseValue \(AutomationDesk Automation\)](#)
- [ValueFactory \(AutomationDesk Automation\)](#)

Signal handling for stimulating:

- [SignalDescription \(AutomationDesk Automation\)](#)
- [SignalDescriptionSet \(AutomationDesk Automation\)](#)
- [SignalDescriptionsReader \(AutomationDesk Automation\)](#)
- [SignalDescriptionsWriter \(AutomationDesk Automation\)](#)

- [SignalFactory \(AutomationDesk Automation\)](#)
- [SignalGenerator \(AutomationDesk Automation\)](#)
- [SignalGeneratorFactory \(AutomationDesk Automation\)](#)
- [SignalGeneratorReader \(AutomationDesk Automation\)](#)
- [SignalGeneratorWriter \(AutomationDesk Automation\)](#)
- [SignalGroupValue \(AutomationDesk Automation\)](#)
- [SignalSegment \(AutomationDesk Automation\)](#)
- [SignalValue \(AutomationDesk Automation\)](#)

Capture handling:

- [Capture \(AutomationDesk Automation\)](#)
- [CaptureResult \(XIL API\) \(AutomationDesk Automation\)](#)
- [CaptureResultReader \(AutomationDesk Automation\)](#)
- [CaptureResultWriter \(AutomationDesk Automation\)](#)
- [CaptureState \(AutomationDesk Automation\)](#)
- [CapturingFactory \(AutomationDesk Automation\)](#)

Common data objects handling:

- [Attributes \(AutomationDesk Automation\)](#)
- [DataType \(AutomationDesk Automation\)](#)
- [Symbol \(AutomationDesk Automation\)](#)
- [SymbolFactory \(AutomationDesk Automation\)](#)
- [TaskInfoFactory \(AutomationDesk Automation\)](#)
- [ValueInfo \(AutomationDesk Automation\)](#)
- [Watcher \(AutomationDesk Automation\)](#)
- [WatcherFactory \(AutomationDesk Automation\)](#)

When you work with the XIL API Convenience library, the same data objects are used.

The COM API for automating the platform management is contained in the installation of dSPACE XIL API .NET. For information on the Platform Management API, refer to [dSPACE Platform Management API Reference](#).

For basic information and instructions, refer to [Basics on Automating AutomationDesk Handling \(AutomationDesk Basic Practices\)](#).

Limitations

Where to go from here

Information in this section

Limitations for Platforms.....	369
Limitations When Using the XIL API Convenience Library.....	370
Limitations When Using the XIL API Library.....	371

Limitations for Platforms

Introduction

There are some limitations for working with platforms in AutomationDesk:

Limited number of host services

The number of host services in an application running on dSPACE real-time hardware is limited to 32.

DS1104: Downloading a slave DSP application

You cannot download a slave DSP application directly to a DS1104. Instead, you have to download the main application containing the slave DSP application to the DS1104. For details, refer to [Basics on Handling Simulation Applications \(ControlDesk Platform Management\)](#).

Downloading a DS230x application

You cannot download a DS230x application directly to a DS230x. Instead, you have to download the main application containing the DS230x application to the processor board that the DS230x is connected to.

Multiprocessor System platform: automation via MCD 3

When you perform automation via ControlDesk's MCD 3-compatible interface, you have access to the individual boards belonging to a Multiprocessor System platform.

However, you do not have access to the Multiprocessor System platform itself.

VEOS platform

- Only one VEOS platform in a project can be assigned to VEOS at a time.
- When an operating system process related to VEOS, e.g., `VEOSDaqManager.exe`, is stopped, the connection between the offline simulation application and the related VEOS platform in AutomationDesk is interrupted. Reconnecting to the VEOS Simulator, e.g., by refreshing the interface connection or re-registering the VEOS platform, will fail.
To reconnect to the VEOS core, you have to stop the `VEOSKernel.exe` manually.
- In test automation, blocks with time references (such as 'wait 3s') can produce a different behavior under VEOS than on SCALEXIO (especially with the VEOS-specific setting 'as fast as possible').
- You can only use VEOS with AutomationDesk, if the product versions from the same dSPACE release are activated in the dSPACE Installation Manager.

Limitations When Using the XIL API Convenience Library

Restriction on stimulating fixed-point variables

When you stimulate an application variable of a fixed-point data type via a SignalGenerator data object, the variable's scaling factor that is specified in the variable description file (TRC file) is ignored.

Reading, writing and capturing application variables of a fixed-point data type considers the scaling factor.

Reading IDF files

- The only IDF file format that can be read by the XIL API that is used by AutomationDesk is IDF version 5. Thus, you cannot read IDF version 6 files, which are written by ControlDesk 6.4 or earlier or by automation blocks that use ControlDesk 6.4 or earlier.
- For migration purposes, you can read IDF files that were generated with a dSPACE XIL API implementation up to and including version 2019-B.

Accessing ASAM MDF Files

If MDF files were generated by third-party products according to the ASAM standard, they may contain variables of types that are not supported by dSPACE products.

The following variable types are not supported:

- Cuboids
- Strings

- Struct
- Struct array

As a consequence, you cannot access string signals, for example.

This document contains only limitations relevant to AutomationDesk. 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 the XIL API Library

Limitations based on the dSPACE XIL API implementation

SetMinBufferSize, GetMinBufferSize automation blocks The SetMinBufferSize and GetMinBufferSize automation blocks are equivalents to the methods of the ASAM XIL API but are not supported by dSPACE platforms in this version. The execution of the blocks aborts with an exception.

Configuring SignalSegment data objects

- The eFORWARD interpolation type is not supported. All methods which access the InterpolationType property can be set to eBACKWARD or eLINEAR.
- The SignalValueSegment only supports values of F1024VectorValue data type.

Restriction on stimulating fixed-point variables

When you stimulate an application variable of a fixed-point data type via a SignalGenerator data object, the variable's scaling factor that is specified in the variable description file (TRC file) is ignored.

Reading, writing and capturing application variables of a fixed-point data type considers the scaling factor.

Serialization of data objects

If you copy or save a configured data object, the internally handled data object instance is reset. If you then access the data object, for example, via `_AD_.<DataObjectName>`, None is returned.

This applies to all data objects except for:

- CaptureResult
- DurationWatcher, ConditionWatcher
- BaseValue
- SignalGroupValue
- SignalValue
- CaptureState
- DataType

Reading IDF files

- The only IDF file format that can be read by the XIL API that is used by AutomationDesk is IDF version 5. Thus, you cannot read IDF version 6 files,

which are written by ControlDesk 6.4 or earlier or by automation blocks that use ControlDesk 6.4 or earlier.

- For migration purposes, you can read IDF files that were generated with a dSPACE XIL API implementation up to and including version 2019-B.
-

Accessing ASAM MDF Files

If MDF files were generated by third-party products according to the ASAM standard, they may contain variables of types that are not supported by dSPACE products.

The following variable types are not supported:

- Cuboids
- Strings
- Struct
- Struct array

As a consequence, you cannot access string signals, for example.

This document contains only limitations relevant to AutomationDesk. For further limitations when you use ASAM MDF files, refer to [Limitations for ASAM MDF Files \(Version 4.x\) \(ControlDesk Measurement and Recording\)](#).

Stimulating variables on offline simulation applications with multiple VPUs

When you work with offline simulation applications with multiple environment VPUs, you can only stimulate variables of *one* member application using the XIL API library.

Glossary

Introduction

The glossary briefly explains the most important expressions and naming conventions used in the AutomationDesk documentation.

Where to go from here

Information in this section

Symbols.....	374
A.....	374
B.....	376
C.....	376
D.....	377
E.....	378
F.....	379
H.....	379
I.....	379
L.....	380
M.....	381
O.....	382
P.....	382
R.....	383
S.....	384
T.....	386
U.....	386
V.....	386

W.....	387
X.....	387

Symbols

@ADLX folder A file system folder with the name <CustomLibraryName>@adlx that stores information on the elements in the related [custom library](#), such as [template](#) files, attached Python modules, or packages.

In the [Library Browser](#), you always use this folder in combination with the related [library file \(ADLX\)](#).

@ADPX folder A file system folder with the name <ProjectName>@adpx that stores information on the elements in the related [project](#), such as [sequence](#) files, attached files, [results](#), and [reports](#).

In the [Project Manager](#), you always use this folder in combination with the related [project file \(ADPX\)](#).

@BLKX folder A file system folder with the name <ElementName>@blkx that stores information on the subelements in an exported [element](#), such as [sequence](#) files, attached files, Python modules, or packages.

You always use this folder in combination with the related [parent element file \(BLKX\)](#).

A

ADL file An AutomationDesk legacy library file that contains the specification of a [custom library](#) which was saved to the file system.

These files were created using AutomationDesk 6.1 or earlier. You can open and migrate them to [library XML files \(ADLX\)](#).

ADL.ZIP file An AutomationDesk legacy archive file that contains the specification of a [custom library](#) which was managed under version control.

These files were created using AutomationDesk 6.1 or earlier and must be migrated to let you continue to work under version control.

ADLX file An AutomationDesk library XML file that contains the specification of a saved or exported [custom library](#). The ADLX file is located in the same folder as the [@ADLX folder](#).

You can open, edit, save, import, and export ADLX files via the [Library Bowser](#).

ADO file An AutomationDesk display options file that contains information on how a [project](#) or [library](#) is displayed when it is opened in the AutomationDesk user interface. This includes [bookmarks](#), [breakpoints](#), and the collapse state of folders and blocks.

These files are created when a project or library is saved or closed.

ADP file An AutomationDesk legacy project file that contains a [project's](#) specification, its [results](#), and its [reports](#).

These files were created using AutomationDesk 6.1 or earlier. You can open and migrate them to [project XML files \(ADPX\)](#).

ADP.ZIP file An AutomationDesk legacy archive file that contains the specification of a [project](#) which was managed under version control.

These files were created using AutomationDesk 6.1 or earlier and must be migrated to let you continue to work under version control.

ADPX file An AutomationDesk project XML file that contains the specification of a saved or exported AutomationDesk [project](#). The ADPX file is located in the same folder as the [@ADPX folder](#).

You can open, edit, save, import, and export ADPX files via the [Project Manager](#).

ALX file An AutomationDesk library legacy XML file that contains the specification of an exported [custom library](#).

These files were created using AutomationDesk 6.0 or earlier. You can import ALX files in the [Library Bowser](#).

APX file An AutomationDesk project legacy XML file that contains the specification of an exported AutomationDesk [project](#).

These files were created using AutomationDesk 6.0 or earlier. You can import APX files in the [Project Manager](#).

ASAM AE XIL API An API standard for the communication between test automation tools, such as AutomationDesk, and test benches, such as dSPACE real-time hardware. The notation XIL indicates that the standard can be used for various *in-the-loop* systems, e.g., SIL, MIL, PIL, and HIL. The XIL API standard is defined by the Association for Standardisation of Automation and Measuring Systems (ASAM).

ASAM General Expression Syntax (ASAM GES) The syntax definition that is used in AutomationDesk to specify trigger conditions. It is part of the XIL API standard that is defined by the Association for Standardisation of Automation and Measuring Systems (ASAM).

Automation block A part of a [sequence](#) that implements an automation task, similar to a subroutine.

Templates for automation blocks are provided by AutomationDesk [libraries](#). Via the [Sequence Builder](#), you can arrange automation blocks to implement the control flow of your automation task.

AutomationDesk Options A dialog that lets you modify the appearance and behavior of some AutomationDesk [panes](#) and the layout of the generated [reports](#).

Automotive Simulation Model (ASM) The dSPACE product that provides open MATLAB®/Simulink® models that are relevant for the simulation of automotive engines (gasoline and diesel) and vehicle dynamics.

B

BLKX file An AutomationDesk element XML file that contains the specification of a saved or exported AutomationDesk [element](#). You can import and export BLKX files in the [Project Manager](#), the [Sequence Hierarchy Browser](#), the [Sequence Builder](#), and the [Library Bowser](#).

Block-specific data object A [data object](#) that resides in the interface of an [automation block](#). It can be used to parameterize the block or to return a resulting data object after block execution.

Most blocks provided by AutomationDesk provide a static interface. However, some blocks let you add data objects to their interfaces dynamically, for example, [Exec blocks](#).

Bookmark A label that you can attach to an [automation block](#) to use it later for quick navigation within the user interface.

Breakpoint A flag that you can set for a [sequence](#) or an [automation block](#) that pauses the execution in debug mode when the element with a set breakpoint is reached. You can manually control whether to resume the execution or to terminate it.

Built-in library The type of [library](#) that is included in AutomationDesk as a software component.

In contrast to [custom libraries](#), you cannot create your own built-in libraries and you cannot view the library's source code.

C

Capture A data object type of the [ASAM AE XIL API](#) that is used to parameterize the capturing of measurement data.

In addition to the [model access port \(MAPort\)](#) to be used and the [variables](#) to be captured, you can specify, a condition to start or to stop data capturing, for example.

CaptureResult A data object type of the [ASAM AE XIL API](#) that is used to handle the captured data. It contains the time stamps and the related measured values of the captured [variables](#).

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>`

Component Object Model (COM) An interface in Microsoft Windows that allows software products of different providers to communicate and to control each other.

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

Control-flow-based testing A test strategy that is based on implementing an automation task by specifying its control flow in [sequences](#).

Custom library The type of [library](#) that you can create and include in AutomationDesk. The [elements](#) that you can add to a custom library are [templates](#) for [data objects](#), [automation blocks](#), and [sequences](#). You can use the library elements as templates by adding [library links](#) to projects or sequences.

Some predefined custom libraries are part of the AutomationDesk product. They are read-only by default.

Data object Objects that can store a value according to the data object's type. You can specify a data object *by value* via an editor that depends on the type or *by reference* via the [Data Object Editor](#).

Data objects can be instantiated specific to a [project](#), to a [sequence](#), or to an [automation block](#).

Templates for data objects of various types are provided via AutomationDesk [libraries](#) and can be created via the [Project Manager](#) or the [Sequence Builder](#), for example.

Data Object Editor A [pane](#) that lets you access the values and references of the data objects of the selected object.

Data Object Selector A dialog that lets you specify a [data object](#) by selecting one from the tree of available data objects.

DataContainer An element that lets you bundle [data objects](#) to structure them. DataContainers can be nested.

Debug mode A mode that lets you execute a [project](#) or a [sequence](#) successively and control the execution manually, for example, by using [breakpoints](#).

Documents folder A standard folder for user-specific documents.

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

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

dSPACE Log A [pane](#) that displays the errors, warnings, information, and advice issued by all installed dSPACE products.

E

Edit dialog The dialog that lets you specify the value of a [data object](#). The default edit dialog depends on the data type of the data object, but you can also use a customized edit dialog.

Electrical error simulation (EES) The simulation of errors in the wiring, such as loose contacts, broken cables, or short-circuits. Electrical error simulation is performed by the EES hardware of an HIL simulator.

Electrical error simulation port (EESPort) A data object type of the [ASAM AE XIL API](#) that is used to provide access to the electrical error simulation (EES) hardware of an HIL simulator.

Element The representation of a resource of a [project](#) in the [Project Manager](#) or a [library](#) in the [Library Bowser](#).

An element is displayed as an icon that reflects the element's type followed by the element's name.

Error configuration file A file in XML format that contains the specification of the simulated electrical errors as a series of states which are each specified via an [error set](#).

Error set A list of electrical errors that occur to the signals at the same time and that specifies the simulated state of the wiring. An empty error set specifies a state with no errors.

Exec block An [automation block](#) that is specified by the Python script to be executed.

You can edit the script via AutomationDesk's [Python Editor](#).

F

FDX file An AutomationDesk project folder legacy XML file that contains the specification of an exported AutomationDesk [project folder](#).

These files were created using AutomationDesk 6.0 or earlier. You can import FDX files in the [Project Manager](#).

H

Hyperlink A click-able reference. When you click the link, the target is opened in an appropriate component.

I

Input dialog A dialog window that demands a manual input.

Instance description The property of an instantiated [element](#) that contains a text which describes the element's purpose.

L

LabeledValue A type of data object for which you can define a dictionary of valid label-value pairs. LabeledValues can be set either by specifying a label or by specifying a value.

LFX file An AutomationDesk library folder legacy XML file that contains the specification of an exported [library folder](#).

These files were created using AutomationDesk 6.0 or earlier. You can import LFX files in the [Library Bowser](#).

Library A container for [templates](#) that you can use to instantiate [data objects](#), [sequences](#), or [automation blocks](#) in your [projects](#).

Libraries are handled via the [Library Bowser](#). Each library is organized as a tree and can be structured using [library folders](#).

There are [built-in libraries](#) and [custom libraries](#).

Library Bowser A pane in AutomationDesk that provides access to the elements of the open libraries.

Library folder An element that structures the contents of a library as a tree.

Library link A type of [element](#) that you can create in a [project](#) or [sequence](#). This type of element is linked to a [template](#) in a [library](#).

Depending on the [link mode](#), the library link represents an instance of the linked library element or a reference to this library element.

Library links let you reuse a library element at multiple positions in one or multiple projects.

Link mode The way in which an instantiated object in your [project](#) can be connected to its related [template](#) in the [library](#).

The link mode determines the synchronization behavior after you modified an object's template.

The following link modes are available:

- *Dynamically linked* - A modification of the template takes immediate effect.
- *Statically linked* - A modification of the template takes effect after you manually synchronized it.

If you break the link between an instantiated object and its template, the object becomes independent from the template and cannot be linked again.

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

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

M

Mapping (For [XIL API Testbench](#) only) An object type of the [ASAM AE XIL API](#) for a data object that contains a mapping of variable aliases to their model paths in the related [simulation application](#).

Only one Mapping data object is supported per [project](#) and it always resides at the top level of the [object hierarchy](#).

Mapping Editor (For [XIL API Framework](#) only) A component that lets you configure an XIL API Framework. This includes, for example, the mapping of aliases to model paths, which you can use in your test cases and which are required to access [variables](#) via a model access port.

The configuration is saved to a framework configuration file (XML) as well as related port configuration and mapping files.

Mapping Viewer A [pane](#) that displays the contents of the used variable mapping.

If you are working with an [XIL API Framework](#), the mapping relates to the framework configuration, which you can edit via the [Mapping Editor](#).

If you are working with the [XIL API Testbench](#), the mapping relates to the project's [Mapping](#) data object, which you can edit in the Mapping Viewer.

MAT file A file that contains measurement data in a format that allows data exchange with MATLAB.

MDF file A file that contains measurement data in a format that complies with the ASAM Common MDF standard. For version 4.1 of this standard, the file name extension is MF4.

Message dialog A dialog that demands manual confirmation for a message, error, warning, or information.

Message Viewer A [pane](#) that displays the history of all error and warning messages that occur while you are working with AutomationDesk.

MF4 file Refer to [MDF file](#).

Model access port (MAPort) A data object type of the [ASAM AE XIL API](#) that is used to provide access to the [variables](#) of a running [simulation application](#).

ModelDesk The dSPACE software product for parameterizing [ASM models](#) via graphical representations of the modeled components and controlling the related real-time simulation, offline simulation, or MATLAB®/Simulink® simulation.

MotionDesk The dSPACE software product that lets you visualize the movement of 3-D objects controlled by a running simulation application.

O

Object hierarchy The hierarchy tree that is built by all objects that are instantiated in a specific [project](#).

Offline simulation application (OSA) A simulation application that can be executed without real-time hardware on a host PC with [VEOS](#). The OSA file that implements the simulation application can be built from a Simulink model by the VEOS Player.

Operation mode A feature that is provided by some [libraries](#) and lets you decide whether to work online with the related device or to work with previously recorded data.

Operation signal The signal type of [signals](#) that are specified as an arithmetic operation (addition or multiplication) of two other signals.

Output Viewer A [pane](#) that displays all output messages generated by AutomationDesk.

P

PADL.ZIP file An AutomationDesk legacy element archive file that contains the specification of a [custom library](#), a [library folder](#), or a [template](#) which was managed under version control.

These files were created using AutomationDesk 6.1 or earlier and must be migrated to let you continue to work under version control.

PADP.ZIP file An AutomationDesk legacy element archive file that contains a [project](#), a [project folder](#), a [sequence](#), or a [result](#) which was managed under version control.

These files were created using AutomationDesk 6.1 or earlier and must be migrated to let you continue to work under version control.

Pane The section of a window that provides related controls. AutomationDesk panes are arranged in [view sets](#).

Parameter Any variable type that can be calibrated.

PCONFIG file An [ASAM AE XIL API](#) EES port configuration file that provides the hardware-dependent information for an [electrical error simulation \(EES\)](#) in XML format.

Platform A software component representing a simulator where a [simulation application](#) is computed in real-time (on dSPACE real-time hardware) or in non-real-time (on [VEOS](#)).

Platform Manager A software component that is commonly used by various dSPACE products to register and access [platforms](#) and to control the execution of [simulation applications](#) on the [platforms](#).

Project A container for all instantiated resources that implement a specific automation task.

Projects are handled via the [Project Manager](#). Each project is organized as a tree and can be structured using [project folders](#).

Project folder An element that structures the contents of a [project](#) as a tree.

Project Manager A [pane](#) in AutomationDesk that provides access to the [elements](#) of the open [projects](#).

Project-specific data object A [data object](#) that is created within a [Project](#) or a [Project folder](#) in the [Project Manager](#). It can be used to parameterize elements a lower level in the [object hierarchy](#).

Properties A [pane](#) that lets you access the properties of selected elements.

Python Editor A component that lets you edit the Python scripts for [Exec blocks](#), their [templates](#), and Python modules and packages that are integrated in AutomationDesk [libraries](#). Each of these elements can be opened in a separate Python Editor [pane](#).

R

Real-time application An application that can be executed in real time on dSPACE real-time hardware. A real-time application can be built from a Simulink model containing RTI blocks, for example.

Real-Time Testing (RTT) The dSPACE software product that provides components for creating and executing Python scripts which run on the real-time hardware in parallel to the [real-time application](#).

Record depth The attribute of an execution that specifies which project [elements](#) are to include in the execution's [result](#) depending on the element's [result levels](#).

The following record depths are provided:

- No result
- High elements only
- High and medium elements

Report A document in PDF or in HTML format that is generated from an execution's [result](#).

Result A set of data that results from the execution of a [project](#), a [project folder](#), or a [sequence](#).

From a result, you can generate a [report](#).

Result Browser A component that displays the [result](#) of the execution of a [project](#), a [project folder](#), or a [sequence](#) during the execution in form of a tree of the involved data objects and their values.

Each result that you open in the Result Browser is displayed in a separate [pane](#).

Result level The attribute of an element that specifies whether to include the element in an execution's [result](#), depending on the execution's [record depth](#).

AutomationDesk provides the None, Medium, and High result levels.

Result parameter The attributes that specify whether an [element](#) is included in an execution's [result](#). For this, AutomationDesk provides the [result level](#) and the [record depth](#) attributes.

Root element The top-level element of a tree data structure. A root element represents the entire element tree of a [project](#) in the [Project Manager](#) or a [library](#) in the [Library Bowser](#), for example.

S

Segment signal The signal type of the signals that are specified as a sequence of [signal segments](#).

Sequence The implementation of an automation task as a control flow specified with [automation blocks](#).

Sequences are edited via the [Sequence Builder](#).

Sequence Builder A component that lets you graphically edit the control flow of a [sequence](#), sequence [template](#) or subsequence template. Each of these elements can be opened in a separate Sequence Builder [pane](#).

Sequence Hierarchy Browser A [pane](#) in AutomationDesk that provides access to the [elements](#) of the [sequence](#) that is currently displayed in the [Sequence Builder](#).

SequenceFrame A [template](#) that is provided by the Framework Builder [built-in library](#) and that lets you specify a predefined frame for implementing similar [sequences](#).

SFX file A sequence frame legacy XML file that contains the specification of an exported [SequenceFrame](#) object.

These files were created using AutomationDesk 6.0 or earlier. You can use the [Project Manager](#) to import SFX files for handling instantiated sequence frames and the [Library Bowser](#) for handling sequence frame [templates](#).

Signal The specification or measurement of the change of a value over time.

Signals can be specified by their shape in a [signal description set](#) as a [segment signal](#) or as an [operation signal](#).

Signal description set A container for a set of [signal](#) specifications that implement a specific [signal-based test](#).

Signal description sets are handled via the [Signal Editor](#) as a table of the contained signals.

Signal Editor A component that lets you graphically edit a [signal description set](#) as a table of its contained [signals](#).

Multiple signal description sets can be opened in separate Signal Editor [panes](#).

Signal file A file in CSV format that defines via failure classes which electrical errors can be simulated by the specific EES hardware.

Signal generator A software component, that can be configured and controlled via a [data object](#) in AutomationDesk. A signal generator can be downloaded to a [platform](#) and stimulate [variables](#) in a running [simulation application](#) in real-time.

Signal segment One member in the sequence of segments that builds a [segment signal](#). A segment is specified by its type and by its other properties.

The segment type is specified at the segment's creation via the [Signal Selector](#). Its other properties can be specified via the [Signal Editor](#) or the [Properties](#) [panes](#).

Signal Selector The [pane](#) that provides elements to add [segment signals](#), [operation signals](#), and [segments](#) of various segment types to your [signal description set](#) by dragging them to the [Signal Editor](#).

Signal-based testing A test strategy that is based on implementing an automation task by using [templates](#) of the Signal-Based Testing [library](#) and specifying all involved [signals](#) in a [signal description set](#).

Simulation application The generic term for [offline simulation application \(OSA\)](#) and [real-time application](#).

SQX file An AutomationDesk sequence legacy XML file that contains the specification of an exported AutomationDesk [sequence](#).

These files were created using AutomationDesk 6.0 or earlier. You can use the [Project Manager](#) to import SQX files for handling instantiated sequences and the [Library Bowser](#) for handling sequence [templates](#).

Stylesheet An XSL file that specifies the layout for the generation of a [report](#) from an execution's [result](#).

STZ file A ZIP file that contains the description of a [signal description set](#) in STI format. The STI format is defined by the [ASAM AE XIL API](#) standard.

You can create and manage STZ files in AutomationDesk's [Signal Editor](#).

Subsequence An [automation block](#) that can contain other automation blocks to implement a part of a sequence's control flow, for example, a loop or a subroutine.

T

Task A thread that is executed on dSPACE real-time hardware.

The execution of tasks is triggered by timer events, I/O events, or software events.

TBX file A block template legacy XML file that contains the specification of an exported [library folder](#).

These files were created using AutomationDesk 6.0 or earlier. You can import TBX files in the [Library Bowser](#).

Template The reusable pattern of a [data object](#), an [automation block](#), or a [sequence](#).

To make a template executable, you must instantiate it as an object in your [project](#).

Template description The property of a [template](#) that provides a text which describes the template's purpose.

TSX file A sequence frame legacy XML file that contains the specification of an exported TestSequence (Test Framework) object.

These files were created using AutomationDesk 6.0 or earlier. You can use the [Project Manager](#) to import TSX files for handling instantiated sequence frames and the [Library Bowser](#) for handling TestSequence [templates](#).

U

User function The call of an external program that you can integrate in AutomationDesk's user interface.

V

Value Editor A component that opens a modal [Input dialog](#) to edit the selected [data object's](#) value.

The appearance of the dialog depends on the type of the selected data object.

Variable A parameter in the [simulation application](#) that can be read and written.

A parameter identified by its [variable path](#).

Variable description file The SDF file, the RTA file, or the [OSA](#) file that contains the specifications for an executable [simulation application](#).

Variable path The path to the [variable](#) in the hierarchy of the model from which the [simulation application](#) is built.

Variables pane A component that lets you edit the configuration of an [model access port \(MAPort\)](#). You can select the [platform](#) type to be accessed and specify the [variable description file](#) to be used. Then you can browse the tree of the provided model [variables](#). Each MAPort configuration can be opened in a separate Variables [pane](#).

Variant A type of [data object](#) that can reference other data objects of any type.

VEOS A dSPACE software product that can execute [offline simulation applications](#) on a HostPC independently of real time. No real-time hardware is required.

Verdict A type of [data object](#) that is used to qualify the current success status of a [sequence](#), [subsequence](#), or [automation block](#).

View set A configuration of the screen arrangement. You can create various view sets and switch between them. By default, AutomationDesk provides the preconfigured view sets Sequences, Signals, and Execution.

VirtualCOM An interface object for handling AutomationDesk's COM objects. VirtualCOM ensures a proper cleanup of deleted objects in AutomationDesk's namespace.

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

XIL API Framework An access layer that is defined in the [ASAM AE XIL API](#) standard.

It lets you centrally configure the access to the entire test infrastructure in XML files. This decouples test cases from the real and virtual test systems you use.

XIL API Testbench An access layer that is defined in the [ASAM AE XIL API](#) standard.

It lets you configure the access from a test to its environment, such as a simulator, by using ports. For example, the access to [variables](#) of a [simulation application](#) is configured by using a [model access port](#). This decouples test software from test hardware.

A

accessing hardware simultaneously
 access levels 32
 error messages 34
accessing simulation platforms
 XIL API Convenience library 76
AddCustomPlotsToReport
 XIL API Convenience library 172
AddPlotsToReport
 XIL API Convenience library 173
Assembly View command 305
Attributes (Data Object)
 XIL API library 273
AutomationDesk
 Platform Management Library 142

B

BaseValue (Data Object)
 XIL API library 274
basics
 Platform Management library 49
 Platform Manager 41
 platforms 21
 simulation application 22

C

Capture (Data Object)
 XIL API library 231
CaptureEvent
 XIL API library 222
CaptureResult (Data Object)
 XIL API library 223
CaptureResultReader (Data Object)
 XIL API library 232
CaptureResultWriter (Data Object)
 XIL API library 233
CaptureState (Data Object)
 XIL API library 234
CapturingFactory (Data Object)
 XIL API library 235
CheckValues
 XIL API library 156
Clear Flash dialog 306
ClearConfiguration
 XIL API library 236
Common Program Data folder 14, 377
Configure (XIL API Framework) command 343
ConfigureSignalSegment
 XIL API Convenience library 195
ConfigureStartCondition
 XIL API Convenience library 175
ConfigureStopCondition
 XIL API Convenience library 177
ConfigureStopDuration
 XIL API Convenience library 179
CreateCapture
 XIL API library 290
CreateSignalGenerator
 XIL API library 291

D

DataType (Data Object)
 XIL API library 275
DestroyOnTarget
 XIL API Convenience library 196
 XIL API library 258
Disable Framework Support command 344
Documents folder 14, 378
Duration
 XIL API library 214
DurationFactory (Data Object)
 XIL API library 216

E

edit
 XIL API 356, 357
Edit
 TaskName 351
 Variables 352
 Vendor 354
Edit (XIL API Framework) command 345
Edit as Dictionary command
 MAPortConfiguration (XIL API) 360
Edit command
 MAPortConfiguration (XIL API) 358
Edit command (CaptureState) 356
Edit command (DataType) 357
Enable Framework Support command 346
ErrorInfo (Data Object)
 XIL API library 217
Export Mapping dialog 362
Export XIL API Mapping command 361
ExtractSignalValue
 XIL API library 225

F

Fetch
 XIL API library 237

G

GetCaptureResult
 XIL API Convenience library 181
 XIL API library 239
GetCaptureState
 XIL API Convenience library 182
GetDataType
 XIL API library 292
GetDictionaryFromCaptureResult
 XIL API Convenience library 184
GetMetaData
 XIL API library 226
GetMinBufferSize
 XIL API library 240
GetPlatform 144
GetPlatformManagement 145
GetRecordedData
 XIL API Convenience library 191
GetRegisteredPlatformNames 146
GetSignalGroupNames

 XIL API library 227
GetSignalGroupValue
 XIL API library 228
GetSimulationApplication 147
GetSimulationState 148
GetState
 XIL API library 241
GetTaskNames
 XIL API library 293
GetValues
 XIL API library 157
GetVariableNames
 XIL API library 294
GetVariables
 XIL API library 242

H

HIL API library
 get application properties 133
 vendor configuration 125

I

Import Mapping dialog 362
Import XIL API Mapping command 362
InitBaseValue
 XIL API library 278
InitConditionWatcher
 XIL API library 281
InitDurationWatcher
 XIL API library 284
Initialize (XIL API Framework) command 347
InitializeCapture
 XIL API Convenience library 186
InitializeSignalGenerator
 XIL API Convenience library 197
InitMAPort
 XIL API Convenience library 162
 XIL API library 295
InitSignalGeneratorSTZReader
 XIL API library 259
InitSignalGeneratorSTZWriter
 XIL API library 260
inserting AutomationDesk library elements
 XIL API 363
IsReadable
 XIL API library 297
IsWritable
 XIL API library 298

L

limitations
 dSPACE XIL API implementation 371
 platforms 369
 XIL API Convenience library 370
 XIL API library 371
loading
 simulation application (automated) 51
 simulation application (manually) 43
LoadSignalGenerator
 XIL API library 261

LoadSimulationApplication 150
 XIL API Convenience library 204
R
 LoadToTarget
 XIL API Convenience library 200
 XIL API library 262
 Local Program Data folder 14, 380

M
 MAPort (Data Object)
 XIL API library 286
 MAPortConfiguration (Data Object)
 XIL API library 289
 MAPortConfiguration (XIL API)
 Edit as Dictionary command 360
 Edit command 358
 MAPortConfiguration dialog 359
 MAPortFactory (Data Object)
 XIL API library 288
 Mapping (Data Object)
 XIL API library 209
 MetalInfo
 XIL API library 216
 Model access
 XIL API 207
 XIL API Convenience 160

N
 Network View command 316

O
 optional processors
 working with multiprocessor systems 34

P
 platform
 basics 21
 platform management
 Clear Flash dialog 306
 registering 27
 Platform Management library
 basics 49
 Platform Management Library 142
 GetPlatform 144
 GetPlatformManagement 145
 GetRegisteredPlatformNames 146
 GetSimulationApplication 147
 GetSimulationState 148
 LoadSimulationApplication 150
 RefreshPlatformConfiguration 151
 StartSimulation 152
 StopSimulation 153
 UnloadSimulationApplication 154
 platform manager 317
 Platform Manager
 basics 41
 platforms
 limitations 369
 Platform Manager 41
 PortConfig (Data Object)

XIL API library 211
R
 Read
 XIL API Convenience library 164
 XIL API library 299
 ReadValues
 XIL API Convenience library 165
 RefreshPlatformConfiguration 151
 registering a platform
 dSPACE real-time hardware 27
 VEOS 27
 Release Capture 364
 Release MAPort command 365
 ReleaseCapture
 XIL API Convenience library 188
 XIL API library 243
 ReleaseMAPort
 XIL API Convenience library 167
 XIL API library 300
 ReleaseSignalGenerator
 XIL API Convenience library 203
 XIL API library 263
 restarting
 simulation application (automated) 53
 simulation application (manually) 45

S
 SaveSignalGenerator
 XIL API Convenience library 201
 XIL API library 264
 Script
 XIL API library 250
 ScriptParameterInfo (Data Object)
 XIL API library 267
 SetAssignments
 XIL API library 264
 SetMetaData
 XIL API library 229
 SetMinBufferSize
 XIL API library 244
 SetStartTriggerCondition
 XIL API library 245
 SetStopTriggerCondition
 XIL API library 246
 SetValues
 XIL API library 158
 SetVariables
 XIL API library 247
 Shutdown (XIL API Framework) command 350
 SignalDescription
 XIL API library 270
 SignalDescriptionSet
 XIL API library 267
 SignalDescriptionSetReader
 XIL API library 271
 SignalDescriptionSetWriter
 XIL API library 272
 SignalFactory (Data Object)
 XIL API library 268

SignalGenerator (Data Object)
 XIL API library 254
 SignalGeneratorFactory (Data Object)
 XIL API library 256
 SignalGeneratorReader (Data Object)
 XIL API library 257
 SignalGeneratorWriter (Data Object)
 XIL API library 257
 SignalGroupValue (Data Object)
 XIL API library 276
 SignalSegment
 XIL API library 269
 SignalValue (Data Object)
 XIL API library 277
 simulation application
 basics 22
 loading automated 51
 loading manually 43
 restarting automated 53
 restarting manually 45
 stopping automated 53
 terminating automated 55
 terminating manually 46
 unloading automated 55
 Start
 XIL API library 248
 StartCapture
 XIL API Convenience library 189
 StartSignalGenerator
 XIL API Convenience library 202
 XIL API library 265
 StartSimulation 152
 XIL API Convenience library 205
 StartStreamToDisk
 XIL API Convenience library 193
 Stop
 XIL API library 249
 StopCapture
 XIL API Convenience library 190
 stopping
 simulation application (automated) 53
 StopSimulation 153
 XIL API Convenience library 206
 Symbol
 XIL API library 220
 SymbolFactory (Data Object)
 XIL API library 221

T
 TargetScriptFactory (Data Object)
 XIL API library 252
 TargetScriptsFileReader (Data Object)
 XIL API library 252
 TaskInfo (Data Object)
 XIL API library 217
 TaskName 159
 terminating
 simulation application (manually) 46
 simulation applications (automated) 55
 Testbench (Data Object)
 XIL API library 211

TestbenchFactory (Data Object)
 XIL API library 212
 trigger conditions 282

U

unloading
 simulation applications (automated) 55
 UnloadSimulationApplication 154

V

ValueFactory (Data Object)
 XIL API library 277
 VariableInfo (Data Object)
 XIL API library 218
 Variables 159
 Variables pane 359
 Vendor 160
 vendor configuration 125
 VEOS support 21

W

Watcher (Data Object)
 XIL API library 279
 WatcherFactory (Data Object)
 XIL API library 280
 working with multiprocessor systems
 optional processors 34
 Write
 XIL API Convenience library 168
 XIL API library 301
 WriteValues
 XIL API Convenience library 169

X

XIL API
 model access 207
 XIL API Convenience
 model access 160
 XIL API Convenience library
 accessing platforms via third-party XIL API
 servers 71, 114
 accessing simulation platforms 76
 AddCustomPlotsToReport 172
 adding plots to the report 102
 AddPlotsToReport 173
 basic concept 76
 building a basic sequence for variable
 access 78
 capturing data 96
 ConfigureSignalSegment 195
 ConfigureStartCondition 175
 ConfigureStopCondition 177
 ConfigureStopDuration 179
 demo projects 78
 DestroyOnTarget 196
 downloading a simulation application 81
 GetCaptureResult 181
 GetCaptureState 182
 GetDictionaryFromCaptureResult 184

GetRecordedData 191
 InitializeCapture 186
 InitializeSignalGenerator 197
 InitMAPort 162
 LoadSimulationApplication 204
 LoadToTarget 200
 making simulator variables available 84
 overview of the Model Access Port folder 77
 Read 164
 reading simulator variables 91
 ReadValues 165
 ReleaseCapture 188
 ReleaseMAPort 167
 ReleaseSignalGenerator 203
 releasing resources after data capturing 105
 releasing resources after stimulating
 variables 112
 SaveSignalGenerator 201
 specifying a variable pool 84
 specifying signals for stimulation 106
 StartCapture 189
 starting a simulation application 81
 StartSignalGenerator 202
 StartSimulation 205
 StartStreamToDisk 193
 stimulating variables 108
 StopCapture 190
 stopping signal generation 112
 StopSimulation 206
 typical workflow 77
 Write 168
 WriteValues 169
 writing simulator variables 87
 XIL API Convenience library commands 350
 XIL API framework
 basics 57
 typical workflow 58
 working with a third-party XIL API
 framework 71
 XIL API Framework 155
 XIL API Framework commands 342
 XIL API library
 Attributes (Data Object) 273
 BaseValue (Data Object) 274
 basics 118
 Capture (Data Object) 231
 capture data 134
 CaptureEvent 222
 CaptureResult (Data Object) 223
 CaptureResultReader (Data Object) 232
 CaptureResultWriter (Data Object) 233
 CaptureState (Data Object) 234
 CapturingFactory (Data Object) 235
 CheckValues 156
 ClearConfiguration 236
 CreateCapture 290
 CreateSignalGenerator 291
 DataType (Data Object) 275
 DestroyOnTarget 258
 Duration 214
 DurationFactory (Data Object) 216

ErrorInfo (Data Object) 217
 example 121
 ExtractSignalValue 225
 Fetch 237
 GetCaptureResult 239
 GetDataType 292
 GetMetaDataTable 226
 GetMinBufferSize 240
 GetSignalGroupNames 227
 GetSignalGroupValue 228
 GetState 241
 GetTaskNames 293
 GetValues 157, 158
 GetVariableNames 294
 GetVariables 242
 InitBaseValue 278
 InitConditionWatcher 281
 InitDurationWatcher 284
 initialize MAPort 126
 InitMAPort 295
 InitSignalGeneratorSTZReader 259
 InitSignalGeneratorSTZWriter 260
 IsReadable 297
 IsWritable 298
 LoadSignalGenerator 261
 LoadToTarget 262
 main elements 208
 MAPort (Data Object) 286
 MAPortConfiguration (Data Object) 289
 MAPortFactory (Data Object) 288
 Mapping (Data Object) 209
 MetalInfo 216
 migrating a variable pool to Mapping data
 object 138
 overview 119
 PortConfig (Data Object) 211
 preparing a project 123
 providing simulator variables 127
 Read 299
 read and write variables 129
 ReleaseCapture 243
 ReleaseMAPort 300
 ReleaseSignalGenerator 263
 SaveSignalGenerator 264
 Script 250
 ScriptParameterInfo (Data Object) 267
 SetAssignments 264
 SetMetaDataTable 229
 SetMinBufferSize 244
 SetStartTriggerCondition 245
 SetStopTriggerCondition 246
 SetVariables 247
 SignalDescription 270
 SignalDescriptionSet 267
 SignalDescriptionSetReader 271
 SignalDescriptionSetWriter 272
 SignalFactory (Data Object) 268
 SignalGenerator (Data Object) 254
 SignalGeneratorFactory (Data Object) 256
 SignalGeneratorReader (Data Object) 257
 SignalGeneratorWriter (Data Object) 257

SignalGroupValue (Data Object) 276
SignalSegment 269
SignalValue (Data Object) 277
Start 248
StartSignalGenerator 265
Stop 249
Symbol 220
SymbolFactory (Data Object) 221
TargetScriptFactory (Data Object) 252
TargetScriptsFileReader (Data Object) 252
TaskInfo (Data Object) 217
Testbench (Data Object) 211
TestbenchFactory (Data Object) 212
ValueFactory (Data Object) 277
VariableInfo (Data Object) 218
Watcher (Data Object) 279
WatcherFactory (Data Object) 280
Write 301
XIL API library commands 355