

AutomationDesk

Introduction And Overview

For AutomationDesk 6.5

Release 2021-A – May 2021

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

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

© 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	5
Safety Precautions and Legal Information	7
General Warning.....	7
Legal Information.....	8
Introduction to AutomationDesk	9
AutomationDesk's Field of Application.....	10
Basic Concepts.....	12
Supported Test Strategies.....	17
Protecting Long-Term Tests Against Interrupting or Blocking.....	18
AutomationDesk and Automation Server.....	18
User Interface of AutomationDesk.....	19
Overview of Symbols.....	32
Overview of Libraries.....	35
Packaging of AutomationDesk.....	40
Demos for AutomationDesk	43
Basics on Demo Projects.....	43
Tutorials, Tutorial Videos, and PDF Documents	47
Tutorial for AutomationDesk.....	47
Tutorial Videos for AutomationDesk.....	48
PDF Documents for AutomationDesk.....	48
New Features of AutomationDesk	51
New Features of AutomationDesk 6.5.....	51
Compatibility, Migration, and Discontinuations	53
Compatibility.....	54
Compatibility of AutomationDesk 6.5.....	54
Compatibility of Firmware.....	56

Migration.....	58
Migrating AutomationDesk.....	58
General Migration Aspects.....	58
Library-Specific Migration Aspects.....	60
Migrating Python Scripts from Python 3.6 to Python 3.9.....	62
Main Changes in Python 3.9.....	62
Main Changes in Handling Python 3.9 with dSPACE Software.....	63
General Information on Using Python Installations.....	64
Technical Changes.....	65
Migrating Discontinued AutomationDesk Libraries.....	66
General Aspects on Migrating Discontinued Libraries in AutomationDesk.....	66
Migrating Blocks of the Platform Access Library.....	71
 Glossary.....	 81
 Index.....	 97

About This Document

Contents

This document introduces you to AutomationDesk.

Required knowledge





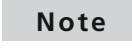


Working with AutomationDesk requires:


- 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>.

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.

Symbol	Description
	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.

Safety Precautions and Legal Information

Introduction

To avoid risk of injury and/or damage to the dSPACE hardware, read and ensure that you comply with the following safety precautions. These precautions must be observed during all phases of system operation.

Where to go from here

Information in this section

[General Warning..... 7](#)

Using the AutomationDesk software can have a direct effect on technical systems (electrical, hydraulic, mechanical) connected to it.

[Legal Information..... 8](#)

Legal information on ASAM binaries and ASAM documentation

General Warning

Danger potential

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

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

Using the AutomationDesk software can have a direct effect on technical systems (electrical, hydraulic, mechanical) connected to it.

The risk of property damage or personal injury also exists when AutomationDesk is controlled via an automation interface. AutomationDesk is then part of an overall system and may not be visible to the end user. It nevertheless produces a

direct effect on the technical system via the controlling application that uses the automation interface.

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

Liability

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

Data loss during operating system shutdown

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

Legal Information

Legal information on ASAM binaries and ASAM documentation

Note

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

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

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

Introduction to AutomationDesk

Introduction

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

Where to go from here

Information in this section

AutomationDesk's Field of Application.....	10
AutomationDesk can be used to make specific steps of an automation process more efficiently.	
Basic Concepts.....	12
Before you learn specific information about AutomationDesk you should know some of its basic concepts.	
Supported Test Strategies.....	17
AutomationDesk supports you in developing control-flow-based tests and signal-based tests.	
Protecting Long-Term Tests Against Interrupting or Blocking.....	18
Long-term tests must not be interrupted by your host PC activating a power-saving state triggered by a user action or by Microsoft Windows.	
AutomationDesk and Automation Server.....	18
AutomationDesk is also available as Automation Server.	
User Interface of AutomationDesk.....	19
To work with AutomationDesk via its user interface you must know the panes that it provides.	
Overview of Symbols.....	32
The elements and their states are represented by specific symbols.	
Overview of Libraries.....	35
AutomationDesk comes with several libraries that provide commonly usable elements for building automation programs.	

Packaging of AutomationDesk.....	40
AutomationDesk and AutomationDesk - Automation Server need additional packages for executing.	

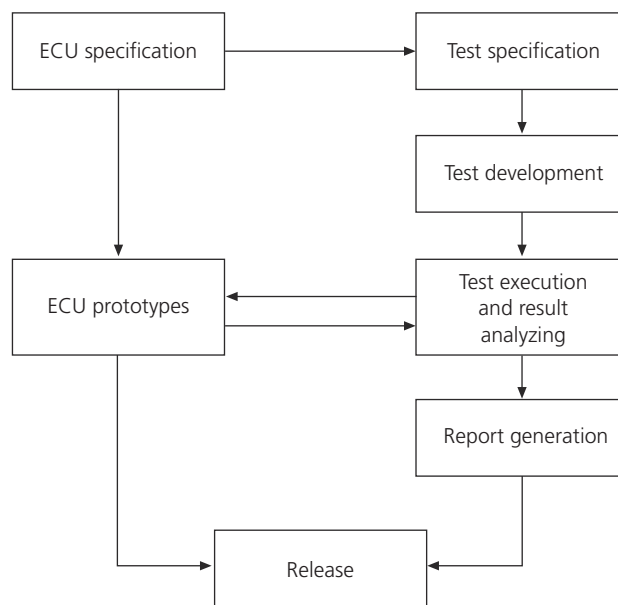
AutomationDesk's Field of Application

Introduction

AutomationDesk is a universal tool for creating and managing automation tasks. A typical automation task of automotive customers is the testing of a new electronic control unit (ECU). The test process can be made more efficient by using AutomationDesk.

Test process without AutomationDesk

The working field of AutomationDesk is explained in the following illustration.



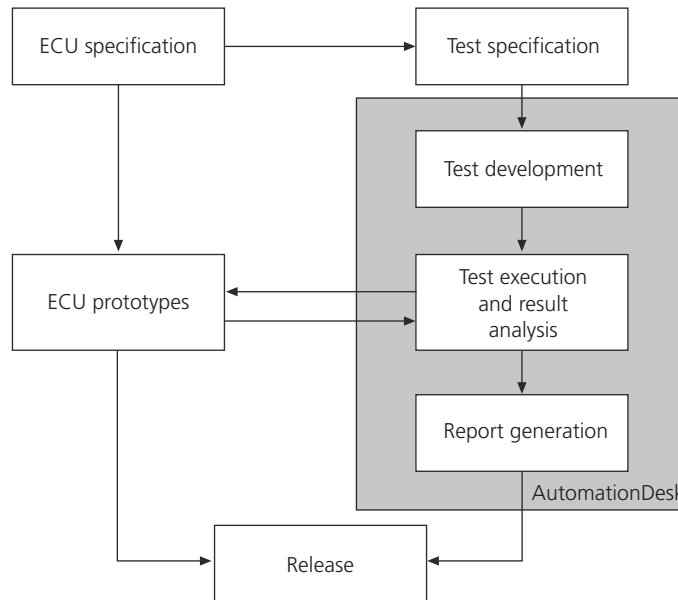
In parallel to the ECU development process, there is the ECU test process. The activities *Test development*, *Test execution*, *Result analyzing* and *Report generation* require and create a lot of data. Managing this amount of information normally requires more than one software tool.

Test process with AutomationDesk

The handling of these activities is now concentrated in one software tool – AutomationDesk.

ECU Development Process

Test Process

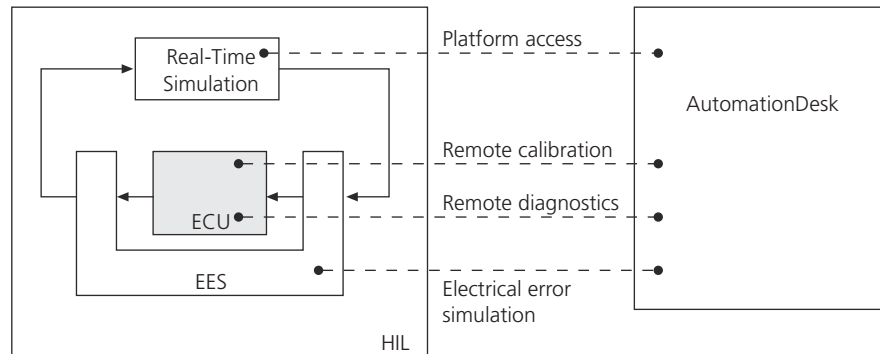


With AutomationDesk you can manage all the data you need for testing. You can create and specify control flows and test parameters, execute tests and log the results. All execution results are stored internally. They can be exported as XML data and stored in an HTML or PDF file on the fly. If you search for specific information, you will find it easily because of the structured view of your whole project. It is guaranteed that you can identify and reproduce your tests.

Main functional features of AutomationDesk

If you want to test an ECU in a Hardware-In-the-Loop (HIL) simulator, you can use AutomationDesk to realize all accesses to:

- Real-time simulation
- Diagnostic tools
- Calibration tools
- Electrical error simulation hardware



Basic Concepts

Introduction

Before you start working with AutomationDesk, you should know some of its general concepts.

AutomationDesk's data organization concept

In AutomationDesk, an automation task, such as a test, is represented by an AutomationDesk project.

AutomationDesk project A project contains the implementation of the automation routines, the parameterization data, the results of already executed tasks, and their generated reports. The Project Manager is used to handle and display these elements in a hierarchical structure. For more information, refer to [Managing Projects \(AutomationDesk Basic Practices !\[\]\(ec9132f1d27c8919987d92907322654d_img.jpg\)](#)).


Implementation concept for your automation routines

The automation routines can be implemented graphically in AutomationDesk sequences.


AutomationDesk sequence For programming a sequence, the Sequence Builder is used. It lets you arrange automation blocks to a control flow chart that represents the routine's implementation. The automation blocks and the data objects for parameterizing them are provided by the AutomationDesk library. For more information, refer to [Building Automation Sequences \(AutomationDesk Basic Practices !\[\]\(dd161862f9164df98f62b726e9846241_img.jpg\)](#)).

AutomationDesk library The AutomationDesk library provides templates for the data objects and automation blocks that you use to build your projects and sequences. For handling the library's elements, the Library Browser is used. The library is divided into feature-oriented sublibraries for implementing the control

flow, accessing other devices and applications, etc. For more information, refer to [Overview of Libraries](#) on page 35.

Integration of customized Python solutions If you are an experienced Python programmer, AutomationDesk provides automation blocks in which you can integrate your own Python solutions. For more information, refer to [Using Python in AutomationDesk](#) ([AutomationDesk Basic Practices](#) ).

Parameterization concept

You can parameterize projects, sequences, and automation blocks via data objects, such as integers, floats and strings. The scope of a data object depends on where it resides in the hierarchy that is built by its project, project folders, sequence and control flow automation blocks. For more information, refer to [Parameterizing Sequence Elements](#) ([AutomationDesk Basic Practices](#) ).

A data object can be specified by its value or by a reference to another data object.

Parameterizing by value You can specify a data object's value via a data-type-specific editor. For example, to edit integers, floats and strings, AutomationDesk provides the Value Editor.

Parameterizing by reference You can specify a reference to another data object via the Data Object Selector. The referenced data object must reside on a higher level within the object's project hierarchy path.

A data object reference is identified only by its name. An automation block searches for the referenced data object on the higher project hierarchy levels from lower to higher levels until it finds a data object with the specified name. If the data object reference cannot be resolved, an error message appears at execution time. If you delete the reference name, a previously specified local value is used again.

Project-specific data objects You can create project-specific data objects via the Project Manager. Via project folders, you can structure a project as a tree of subprojects. A project-specific data object can be referenced by all underlying subprojects, sequences, and automation blocks.


Project-specific data objects are used for the following purposes:

- The project contains sequences that need the same parameters. For example, you can use project-specific variables to centrally configure a model port via an MAPort data object.
- An automation block of the project provides a data object as a result that is required for other automation blocks or sequences.


Auto referencing data objects The interfaces of some automation block templates contain an extended data object that automatically references a project-specific data object of the related type in the block's hierarchy path when the block is instantiated. This applies to MAPort, EESPort, RS232Configuration, MATLAB, and MATFile data objects.


Parameterizing signal-based tests In addition to control-flow-based tests, AutomationDesk also lets you create signal-based tests. These are predefined tests that are parameterized via the shape of the involved signals. These signals specify how simulator variables are stimulated and which variables are captured.

The expected results are specified in the form of the reference signals that are used for evaluation.

All signals that are relevant for a specific signal-based test are bundled in a signal description set. You can specify signal description sets via the Signal Editor. For more information, refer to [AutomationDesk Implementing Signal-Based Tests](#) .


Automation task execution

If you execute a whole project, a subproject, or a sequence, the result is stored in the related node of the project tree. You can also execute a single selected automation block of a sequence. For more information, refer to [Executing Automation Sequences](#) ([AutomationDesk Basic Practices](#) ).

Debugging automation tasks You can debug a project or a sequence at automation block level. For more information, refer to [Executing Sequences in Debug Mode](#) ([AutomationDesk Basic Practices](#) .

Creating reusable elements


AutomationDesk lets you store automation tasks as templates for other projects and sequences.


Custom libraries Automation sequences and parameterized automation blocks can be stored in a custom library so they can be reused. For more information, refer to [Working with Custom Libraries](#) ([AutomationDesk Basic Practices](#) .

Data storing concept

AutomationDesk projects, custom libraries, and signal description sets are saved separately. When one of these is closed, it is checked whether its changes have been saved.

Projects Projects are stored not only to a file, but to a file and a folder.

A project definition is saved to an [ADPX file](#)  (`<ProjectName>.adpx`) in a human-readable XML format. It contains information such as project-specific data objects, the folder structure, and references to the project elements.

The related [@ADPX folder](#)  (`<ProjectName>@adpx`) is used as the starting point for serializing the project data in the file system. The @ADPX folder contains XML files that contain the referenced project elements, such as sequences (`<ElementName>.<GUID>.blkx`).

Additionally, the @ADPX folder can contain the following subfolders:

- The **ado** folder containing information that describes user-specific customization, which has no effect on the test execution. This includes, for example, the collapse state, bookmarks, breakpoints, and other display options, such as block note positions and block colors.
- The **attachments** folder containing files that are configured as attachments, for example, via the **Manage attachments** dialog.
- The **data** folder containing files that are created during export for data used in the project or library, such as XML files with variable mapping, or MDF files with capture results.
- The **reports** folder containing subfolders with generated reports (`Report.<GenerationTimeStamp>`).

- The **results** folder containing subfolders with the results of executed elements (`<ResultName>.<ExecutionTimeStamp>`).

Execution results and generated reports are saved to the file system automatically when they are created. All other project elements, i.e., folders, sequences, and automation blocks, are saved when the project is saved.

If you remove project elements (i.e., folders, execution results, and generated reports), the changes take place in the file system when you save the project.

For opening a saved project in the Project Manager, the ADPX file and the related @ADPX folder are required.

Libraries Libraries are stored not only to a file, but to a file and a folder.

A library definition is saved to an [ADLX file](#) (`<LibraryName>.adlx`) in an XML format. It contains information such as library-specific data objects, the library folder structure, and references to the library elements.

The related [@ADLX folder](#) (`<CustomLibraryName@adlx`) is used as the starting point for serializing the library data in the file system. The @ADLX folder contains XML files that represent the referenced library elements, such as sequence templates (`<TemplateName>.<GUID>.blkx`). Python modules and packages that you added to the custom library are also saved to this folder.

Additionally, the @ADLX folder can contain the following subfolders:

- The **ado** folder can contain information that is not specific to the implemented routines, but to how they are treated in the AutomationDesk user interface. This includes, for example, the collapse state, bookmarks, breakpoints, and other display options, such as whether block notes are displayed.
- The **attachments** folder stores files that you attached to File data objects.
- The **data** folder can contain files that store larger amounts of data, such as captured data or mapping tables.

If you remove library elements, the changes take place in the file system when you save the library.

For opening a saved library in the Library Browser, the ADLX file and the related @ADLX folder are required.

Signal description sets Signal description sets are stored in STZ files (`<SignalDescriptionSetName>.stz`).

An STZ file is a ZIP file that contains the signal descriptions in the STI format. The STI format is defined by the ASAM AE XIL API standard.

A related folder with the same name as the signals description set contains the data of signals that are contained in the signal set except for the captured signals.

If you change signals, the changes take place in the file system when you save the signal description set.

For opening a saved signal description set in the Signal Editor, only the STZ file is required.

For more information, refer to [Basics on the Serialization of AutomationDesk Projects \(AutomationDesk Basic Practices !\[\]\(683dba75afe26e28cd4de5730b776760_img.jpg\)](#)) and [Basics on the Serialization of Custom Libraries \(AutomationDesk Basic Practices !\[\]\(04de6559f804bf16e9c902ae57940aeb_img.jpg\)](#)).

Note

- Do not store files in the folder structure created by AutomationDesk.
- Do not delete or rename files or folders in the folder structure created by AutomationDesk.
- For limitations, refer to [Limitations When Using the Signal-Based Testing Library \(AutomationDesk Implementing Signal-Based Tests !\[\]\(9063468a59e93f469b71000ac5796bc3_img.jpg\)](#)).

Tip

You can add files, such as an STZ file, to a project-specific or library-specific **attachments** folder. Relevant data is then directly stored in the project or custom library and not saved to external files that are not included in a project's export, for example.

AutomationDesk's internal implementation concepts

AutomationDesk projects, sequences and blocks are internally executed by a Python interpreter.

AutomationDesk's programming language Each automation block provided by the AutomationDesk library is a graphical representation of an encapsulated Python class. When you drag an automation block from the Library Browser to the Sequence Builder, you create an instance of the underlying automation object. It is allocated in the namespace of the built-in Python interpreter.

Note

- The administrative elements displayed in the Project Manager are also graphical representations of Python objects.
- You can execute your created automation program without compiling any source code. You do not need to write any Python source code to create standard control and data flows.

Element-specific functions Each AutomationDesk element that was created as a Python object comes with its own functions. The functions that you need for handling the automation block are accessible via the element context menu. Most of them are standard functions that are the same in all elements. However, some elements also provide special features. For example, if you call the Edit command, the dialog that opens might differ from the others.

Further operational concepts

AutomationDesk is based on the following operational concepts:

Lock mechanism If you open an AutomationDesk project, a LOCK file is created in the file system to protect the project. Other users can open the project, but they cannot edit it. The project is indicated as read-only, and its

tooltip displays its lock attribute as "Protected". The same applies for custom libraries that you open in the Library Browser.





Default naming If you add more than one element of the same type in AutomationDesk, the element's label is its name followed by a number, starting with 1. The number increases within the same hierarchy. The second new element in another hierarchy starts with 1 again. For more information, refer to [General Limitations \(AutomationDesk Basic Practices !\[\]\(919a2cb85b99741a73c0c31a427236a8_img.jpg\)](#)).

Undo/Redo AutomationDesk provides an Undo command and a Redo command, if the context allows it.

The commands are stored in separate stacks for each project and custom library. Undo and Redo actions can therefore be performed only for the project or custom library that the previous action was executed in.

For example, if you move a template from CustomLibrary1 to CustomLibrary2, and you want to undo the action, it is not sufficient to call the **Undo** command once. Clicking **Undo** (or entering **Ctrl+Z**) with CustomLibrary2 selected removes the template but does not add it to CustomLibrary1. You then have to select CustomLibrary1 and execute the **Undo** command again to add the template.

Related topics

Basics	
Managing Projects (AutomationDesk Basic Practices 	
Overview of Libraries.....	35
Parameterizing Sequence Elements (AutomationDesk Basic Practices 	
Using Python in AutomationDesk (AutomationDesk Basic Practices 	
Working with Custom Libraries (AutomationDesk Basic Practices 	

Supported Test Strategies

Introduction

AutomationDesk supports you in implementing control-flow-based tests and signal-based tests.



By choosing the appropriate test implementation strategy, you can significantly improve the efficiency of transforming your test specifications to executable tests.

Control-flow-based testing

By using this strategy, you consider your test as an algorithm that is first implemented according to its control flow and then parameterized via data objects before it is executed.

To implement a test's algorithm in an AutomationDesk sequence, you arrange automation blocks with the required functionality to represent the test case's control flow. Structuring elements of the control flow, such as conditional branches and loops, are also provided as automation blocks.


The control-flow-based testing strategy provides a very flexible way to implement your test algorithms according to your specific requirements and purposes.

For more information, refer to [Building Automation Sequences](#) (AutomationDesk Basic Practices ) and to [Executing Automation Sequences](#) (AutomationDesk Basic Practices )

Signal-based testing

By using this strategy, you consider a test scenario as something that is stimulated by input signals. Its reactions are captured in the form of output signals. For evaluation, the output signals are compared to reference signals.

For implementing tests that are based on such a scenario, AutomationDesk provides predefined sequence templates and block templates in the **Signal-Based Testing** library. These templates let you implement tests that are mainly configured via the description of the involved signals. These tests include result evaluation and reporting.

For more information, refer to [AutomationDesk Implementing Signal-Based Tests](#) .

Protecting Long-Term Tests Against Interrupting or Blocking

Avoiding power-saving states

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

AutomationDesk and Automation Server

Introduction

Depending on your AutomationDesk license, you can use AutomationDesk with its user interface and/or the UI-free Automation Server.

AutomationDesk

AutomationDesk comes with a user interface (UI) providing several components for interactive access to projects, sequences, and data objects. For further information, refer to [User Interface of AutomationDesk](#) on page 19.

If you want to remote-control project handling in AutomationDesk, you can use the AutomationDesk API to program a client application for AutomationDesk. You can then create and execute projects with this application. You can also use the API to import data, such as a test parameterization, from the client application and export data, such as the execution results, to the client application. The available features of the API are restricted compared with the

interactive usage of AutomationDesk. For example, you cannot modify the contents of a sequence. For further information, refer to [Using the AutomationDesk API \(AutomationDesk Automation !\[\]\(1d3a1175dd4902218e694b9c098adb83_img.jpg\)](#)).


Automation Server

With the Automation Server license, you can access AutomationDesk only via API. The AutomationDesk user interface is not available. For example, you can use it for executing batch processes, or you can implement a custom user interface for interactive access to it. Its functionality is the same as for the remote-accessed AutomationDesk. For further information, refer to [Using the AutomationDesk API \(AutomationDesk Automation !\[\]\(4729e517bc6a7cd81c8025b9646574fb_img.jpg\)](#)).

The AutomationDesk API also lets you access AutomationDesk via a test management tool, such as SYNECT. For more information, refer to [Using AutomationDesk with SYNECT \(AutomationDesk Basic Practices !\[\]\(cbe80b694ebd74fcfe136a095b608235_img.jpg\)](#)).

Related topics

Basics

User Interface of AutomationDesk.....	19
Using the AutomationDesk API (AutomationDesk Automation )	

User Interface of AutomationDesk

Introduction

To work with AutomationDesk via its user interface, you must know its graphical components and the functionality they provide.

Basics

AutomationDesk provides its controls in several ribbons and [panes !\[\]\(0d5ec72f61334709c3fc9450209b754f_img.jpg\)](#). Some panes are arranged into [view sets !\[\]\(944d59db1282ea95b82255c3404a2195_img.jpg\)](#) that match a task you want to perform.

Independently of the panes, some AutomationDesk components open object-specific pages in the [working area !\[\]\(f81abf985c764528084c28d544d04dc4_img.jpg\)](#) at the center of the screen.

For some object types, AutomationDesk provides specific edit dialogs.

For general information on handling the user interface and detailed instructions on customizing the screen layout, refer to [Customizing AutomationDesk's User Interface \(AutomationDesk Basic Practices !\[\]\(6158d677a88ef2c71bd9d0110ea6e33c_img.jpg\)](#)).

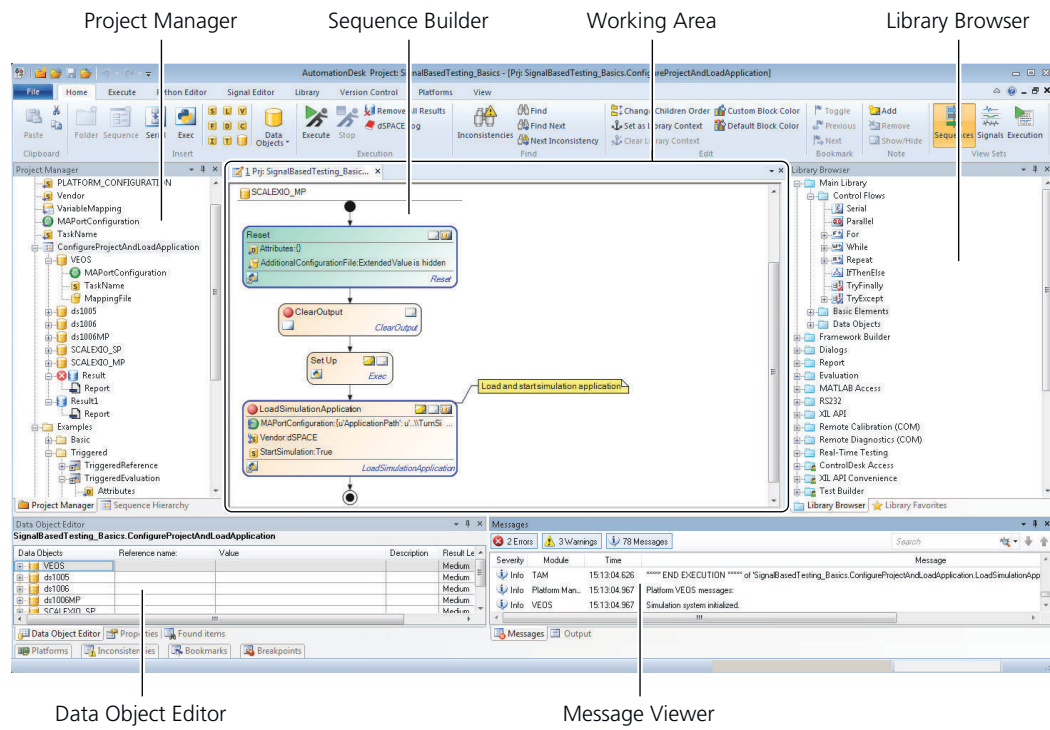
Overview of the default view sets

By default, AutomationDesk provides the following task-specific view sets:

Sequences view set The Sequences view set is suitable for implementing and configuring automation tasks.

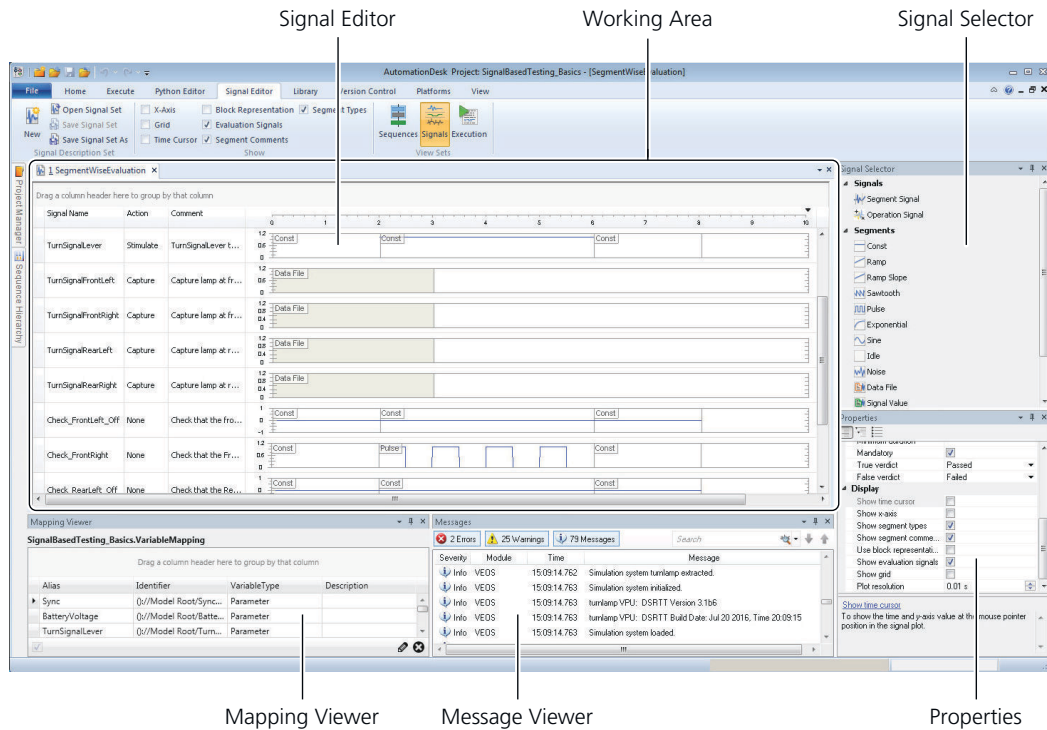
The following illustration shows the appearance of the Sequences view set when the Home ribbon is selected. The Project Manager displays the elements

of an opened [project](#) and, in the working area, one of the project's [sequences](#) is open so it can be edited on a Sequence Builder page.



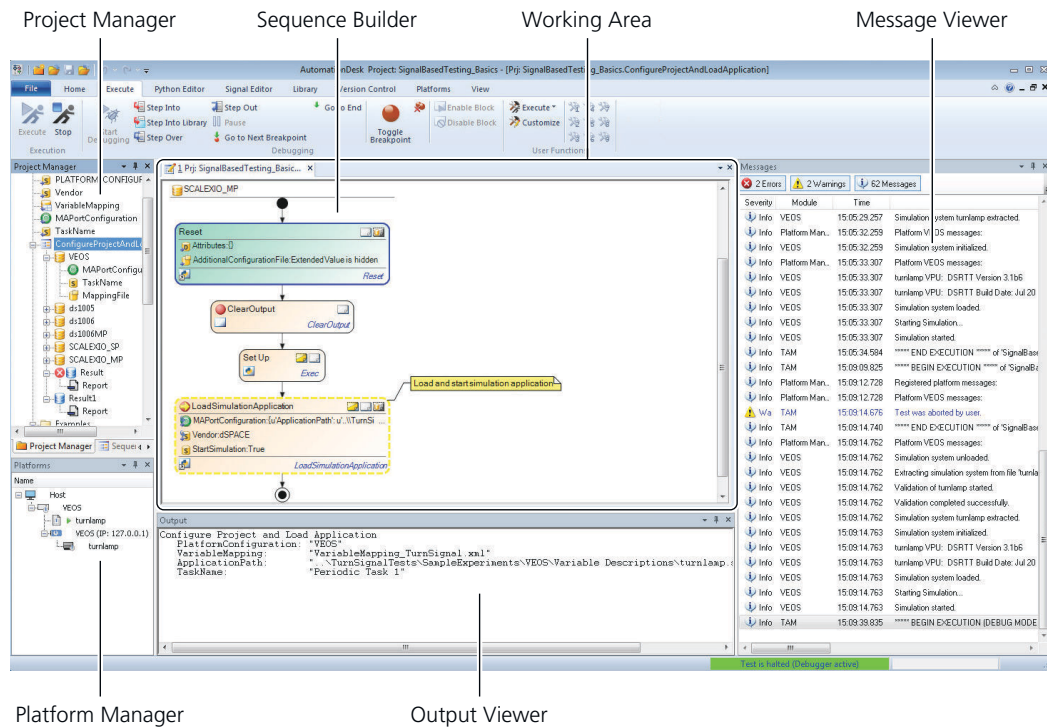
Signals view set The Signals view set is suitable for configuring [signal description sets](#).

The following illustration shows the appearance of the Signals view set when the Signal Editor ribbon is selected. In the working area, a signal description set is open so it can be edited on a Signal Editor page.



Execution view set The Execution view set is suitable for running and debugging the procedures of your tests.

The following illustration shows the appearance of the Execution view set when the Execute ribbon is selected. In the working area, a sequence is open so it can be edited on a Sequence Builder page for debugging.



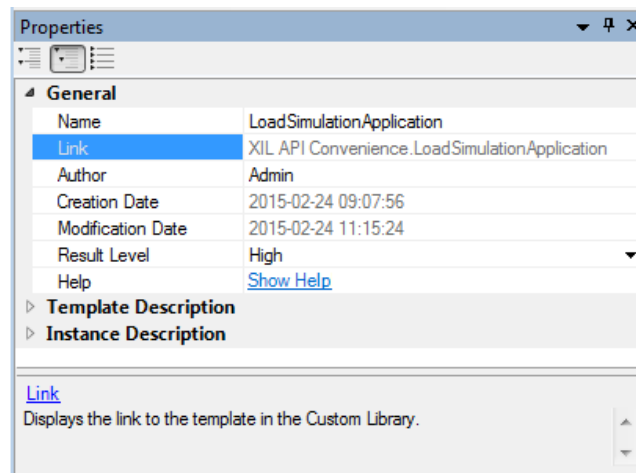
Panes for handling projects and implementing sequences

The following panes are provided to handle projects and implement sequences:

Project Manager The Project Manager gives you a hierarchical view of the project elements. You can use its functions to manage your automation project. For more information, refer to [Using the Project Manager \(AutomationDesk Basic Practices\)](#).

Properties In the Properties pane, you can view and edit the properties of an [element](#) that is selected in one of the other views, such as the Project Manager, the Sequence Builder, and the Signal Editor.

The pane is horizontally divided into two parts. The upper part shows the list of the properties with their names and values. Read-only values are shown in gray. Related properties are organized in categories that you can collapse and expand. If you select a property in the list, a short explanation of the property's meaning is displayed in the lower part of the pane, together with the related link to the online documentation.



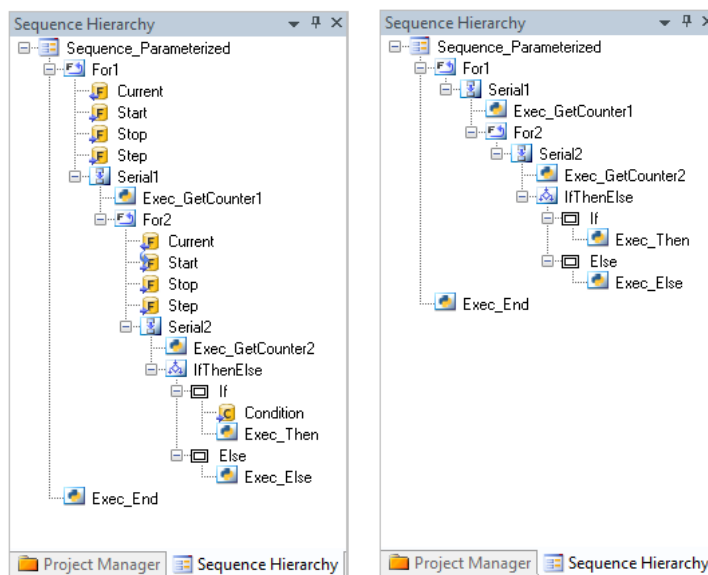
Library Browser The Library Browser provides a wide range of elements for automation tasks. There are [automation blocks](#) for creating control flows with basic elements, and [data objects](#) for defining project-specific parameters and variables. Several standard libraries are included for accessing a [platform](#), accessing the serial interface or working with diagnostic tools. You can create your own custom libraries to integrate user-defined library elements.

Sequence Builder With the Sequence Builder, you can build automation sequences by using automation blocks from the Library Browser and parameterize them. For more information, refer to [Working With Automation Blocks \(AutomationDesk Basic Practices\)](#).

Bookmarks Viewer In the Bookmarks Viewer, you can see the [bookmarks](#) that you specified in the sequences of the open project. By clicking the entries, you can navigate to the selected automation blocks.

Found Items Viewer In the Found Items Viewer, you can see the search results found by the **Find** command.

Sequence Hierarchy Browser Clicking the Sequence Hierarchy page displays the structure of the sequence that is open in the Sequence Builder. You can expand and collapse the sequence tree by clicking the tree nodes. The automation blocks in the Sequence Builder are collapsed or expanded simultaneously. This pane can also be used to edit the properties of automation blocks and create custom library elements by dragging a sequence element to the custom library folder of the Library Browser. When you use the Data Object Editor in single mode, its content is automatically synchronized to the selection in the Sequence Hierarchy Browser. Depending on the settings on the General page of the AutomationDesk Options dialog, the Sequence Hierarchy Browser contains the structure of the selected sequence with the blocks' data objects displayed or hidden.

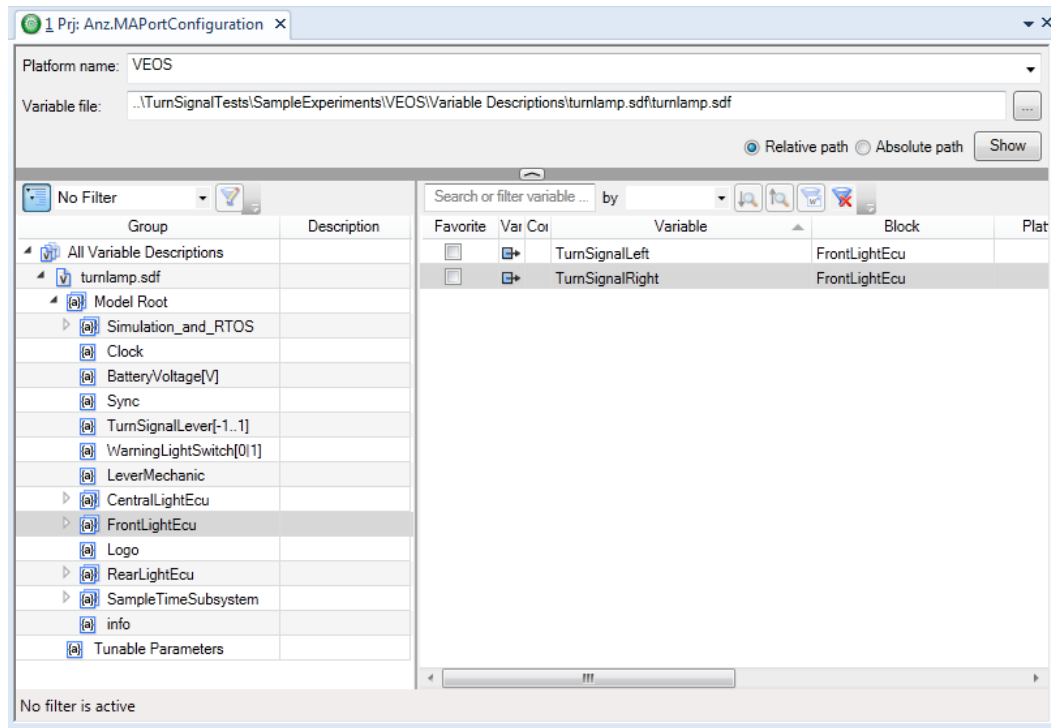


Sequence Overview The Sequence Overview shows a reduced-scale view of the currently active sequence in the Sequence Builder's working area. It supports zooming into the sequence and moving the view.

Library Favorites In the Library Favorites, you can see your individual collection of library elements that you often use. You can create this collection to avoid repeated searching in the Library Browser.

Variables In the Variables pane, you can configure the access to a platform via an MAPortConfiguration data object. You can specify a platform type and a [variable description file](#) to browse through the model tree.

You can add a variable to a variable mapping in the Project Manager or add a signal to a [signal description set](#) in the Signal Editor via drag & drop.



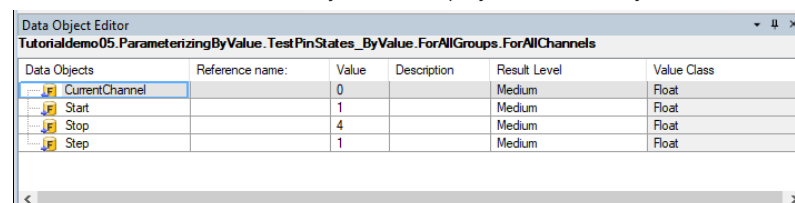
Panes for parameterizing projects and sequences

The following panes are used to parameterize or reference data objects in projects and sequences:

Data Object Editor With the Data Object Editor, you can view and edit the data objects of a selected element. The editor can be used in two different modes, which can be set on the General page of the AutomationDesk Options dialog:

- Single view

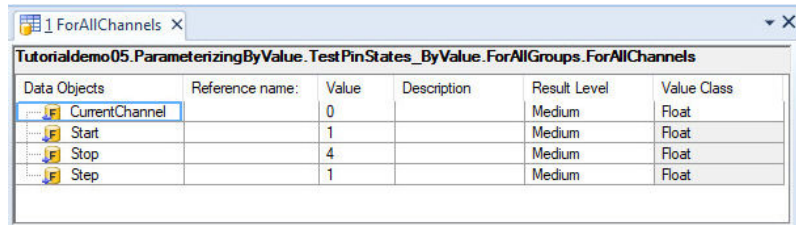
In single view mode, the Data Object Editor is displayed on the Data Object Editor page of the Tool Window. It is activated by the **View Data Objects** command or by clicking the page. The content of the Data Object Editor is automatically synchronized with the currently selected project element or automation block. The data objects are displayed immediately.



- Multiple view

When you use the **View Data Objects** command, the Data Object Editor opens as a separate page in the working pane. Its content does not change

when you select another automation block. It always shows the data objects of the element for which you opened it. You can open further Data Object Editor windows at the same time to view or edit the data objects of other automation blocks. This is useful when you want to display a referenced data object and its source data object at the same time, for example.



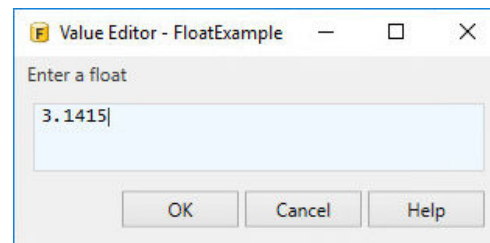
Data Objects	Reference name	Value	Description	Result Level	Value Class
CurrentChannel		0		Medium	Float
Start		1		Medium	Float
Stop		4		Medium	Float
Step		1		Medium	Float

You can switch the display mode of the Data Object Editor on the General page of the AutomationDesk Options dialog. For more information, refer to [Using the Data Object Editor \(AutomationDesk Basic Practices\)](#).

Value Editor If you want to specify a data object by a value, you can double-click it to start the Value Editor. The appearance of the Value Editor depends on the type of data object that you edit:

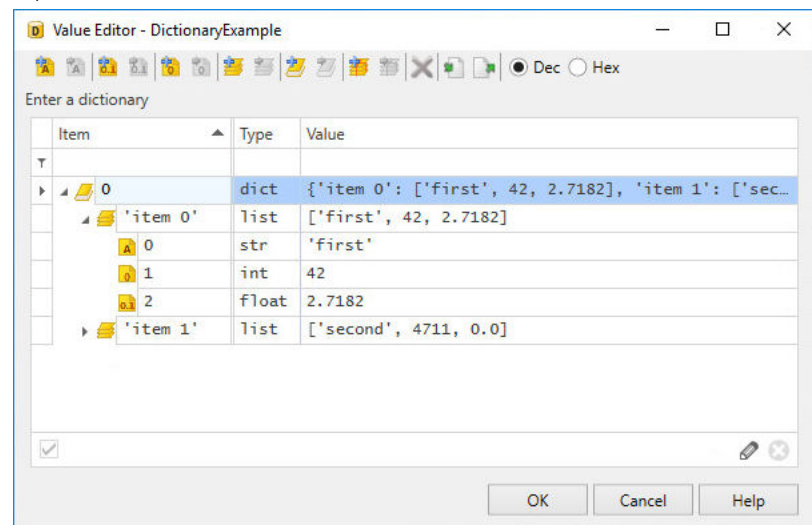
- Integer, float, string

The Value Editor provides one edit field:



- Tuple, list, dictionary

This dialog lets you edit each data element of a tuple, list, or dictionary in a separate row.

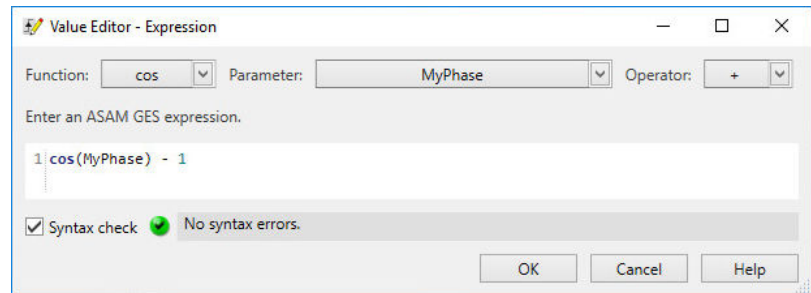


Item	Type	Value
0	dict	{'item 0': ['first', 42, 2.7182], 'item 1': ['sec...
'item 0'	list	['first', 42, 2.7182]
0	str	'first'
1	int	42
2	float	2.7182
'item 1'	list	['second', 4711, 0.0]

- ASAM GES value expression

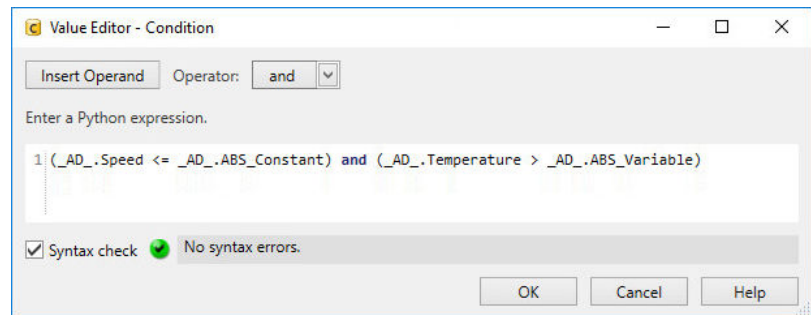
You can use the value editor as an Expression Editor to specify a value that complies with the ASAM General Expression Syntax (GES).

This kind of value expression is used to specify properties of signal description elements, such as the duration of a signal segment.



- Condition, Python expression

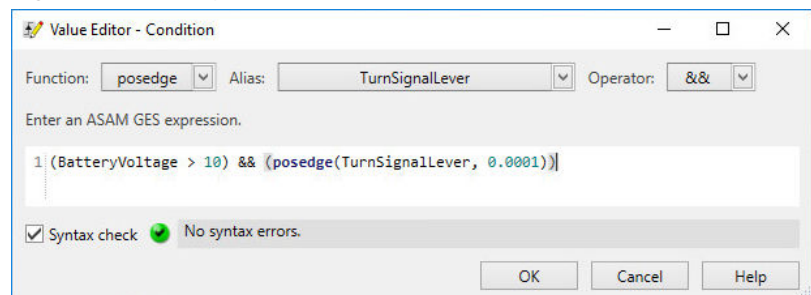
You can use the Value Editor as a Condition Editor to specify a Python expression for a condition as used by an IfThenElse automation block, for example. You can open the Condition Editor by double-clicking the Condition data object.



- Time tag condition, Stop trigger condition, ASAM GES condition expression

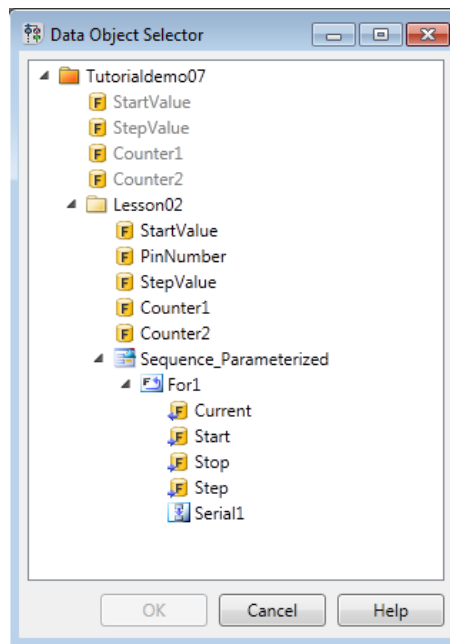
You can use the value editor as an Expression Editor to specify a condition that complies with the ASAM General Expression Syntax (GES).

This kind of condition is used to define stop triggers or time tags for signal segments, for example.



Data Object Selector If you want to specify a data object by reference, you can open the Data Object Editor and click the button in the Reference Name field to open the Data Object Selector. This displays a filtered view of the project with the data objects that you can use for the selected automation block. Select

a data object and click OK to reference it in the Data Object Editor. The available data objects are displayed in a tree structure.



Furthermore, you can use the Data Object Selector to specify references to data objects in custom libraries and in projects by selection. This avoids typing errors. To provide the additional tree structure in the Data Object Selector, you can set the library or project as the current library context.

Note

If you specified a data object with the same name on different hierarchy levels, you can select only the data object from the nearest hierarchy level.

Mapping Viewer With the Mapping Viewer, you can see which alias a model path of a [simulation application](#) is mapped to.

Mapping Viewer			
Anz.Mapping			
Alias	Identifier	VariableType	Description
BatteryVoltage	Q://Model Root/TurnSignalLever[-1..1]/Value	Parameter	Source
TurnSignalLever	Q://Model Root/TurnSignalLever[-1..1]/Value	Parameter	Switch
FrontLightEcu_TurnSignalRight	Q://Model Root/FrontLightEcu/TurnSignalRight	Measurement	Lamp

Via drag & drop, you can add variable items to a variable mapping or create the related signals in a [signal description set](#).

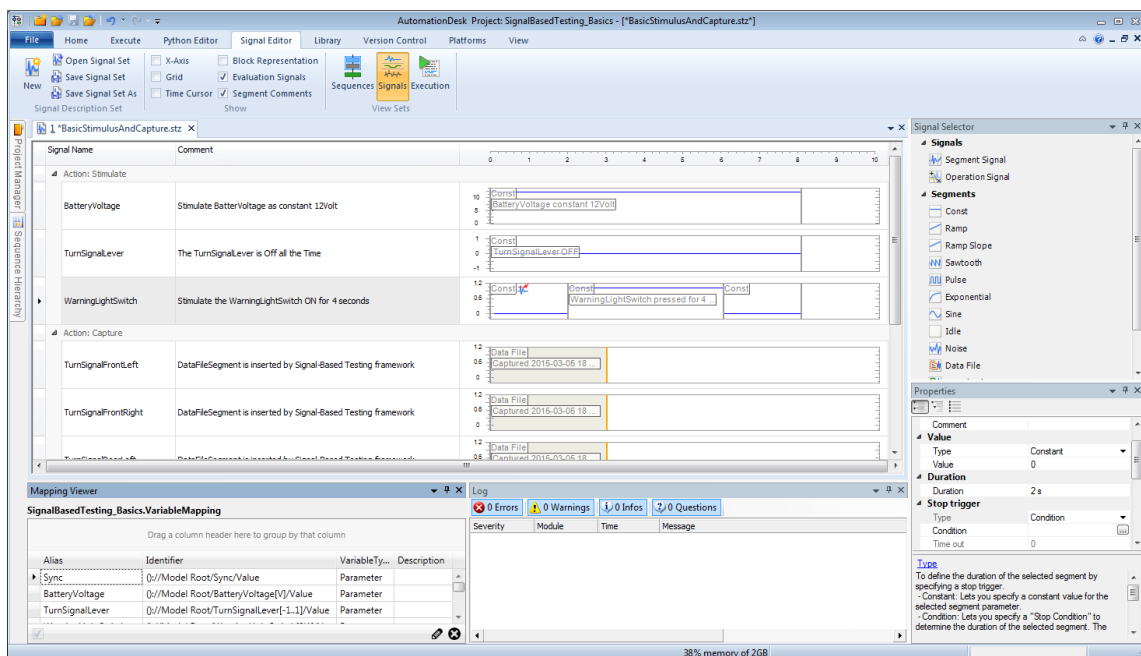
If an [XIL API Framework](#) is initialized, the Mapping Viewer displays the mapping that is configured centrally in the framework configuration. You can edit the mapping by using the Mapping Editor.

If no framework is initialized, you can use the Mapping Viewer to edit the variable mapping, i.e., the contents of the project's Mapping data object.

Panes for parameterizing signal-based tests

For some object types, AutomationDesk provides the following dialogs so you can specify or display their values or references:

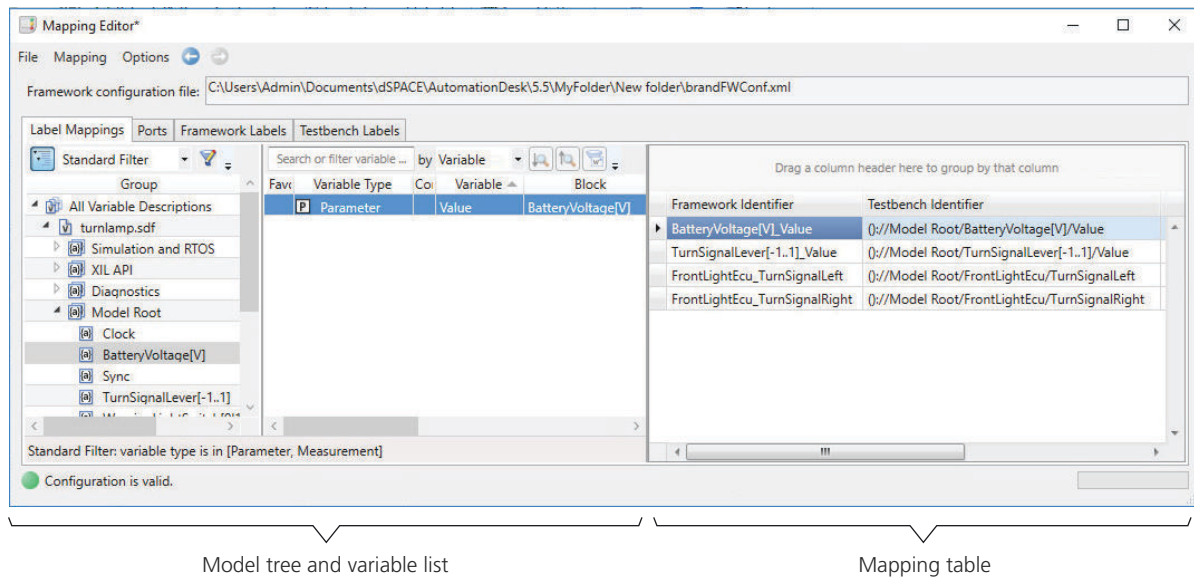
Signal Editor If you want to create or open a signal description set, you can switch to the Signals view set and then use the **New Signal Description Set** or **Open Signal Set** commands to start the Signal Editor. The view set also provides the Signal Selector pane for creating signals and the Properties pane for configuring them. In the editor's working area, the signals of the signal description set are displayed and can be modified graphically.



Signal Selector The Signal Selector provides the [signal](#) and [segment](#) elements, which you can add via drag & drop to a [signal description set](#) in the Signal Editor.

Mapping Editor With the Mapping Editor, you can edit the configuration of an [XIL API Framework](#). This includes the port configuration and the variable mapping to access a simulation application.

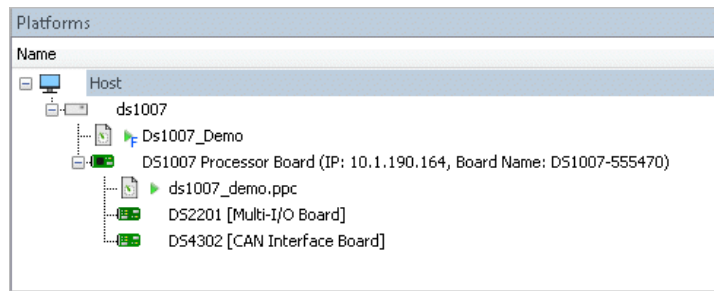
The configuration information is organized in pages. For example, the **Label Mappings** page contains the variable mapping. The available variable paths are provided by the model tree and the variable list. The mapping of paths to variable aliases is contained in the mapping table.



Panes for executing and debugging projects and sequences

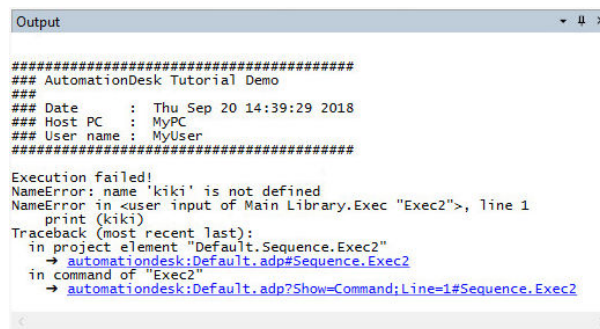
The following panes are used to execute or debug projects, sequences, and blocks:

Platform Manager The Platform Manager displays the registered [platforms](#) with their components and running [applications](#). It provides functions for registering a platform and managing the recent platform configuration. These functions are also available on the Platform ribbon. Additionally, the items of a displayed platform in the Platform Manager provide specific commands and Properties dialogs.



Message Viewer In the Message Viewer, you can see AutomationDesk's log messages, such as, information, warnings, and error messages.

Output Viewer In the Output Viewer, you can see the specified output from print commands and error messages of the Python interpreter. It can contain colored text and hyperlinks. This can be used for debugging purposes, for example.



Tip

If an AutomationDesk error message is displayed, you can click the related hyperlink to open the sequence with the focus on the erroneous block. If this is an Exec block, its script opens in the Python Editor with the cursor positioned on the exception causing statement.

Result Browser When you select the **View Results** command or execute a sequence with result logging, the Result Browser opens. This pane provides an overview of the selected [result](#). You can browse through the result tree to look at certain result values.

Results	Value
TestPinStates_ByReference	
PrintStartOfTest	
ForAllGroups	
ForAllChannels	
CurrentChannel	0
Start	1
Stop	4
Step	1
IsTheChannelHigh	
If	
ReportTestOK	
IsTheChannelHigh	
If	
ReportTestOK	
IsTheChannelHigh	
Else	
ReportTestFailed	
IsTheChannelHigh	
Else	
ReportTestFailed	
ForAllChannels	
PrintEndOfTest	

User Function Output Viewer In the User Function Output Viewer, you can see the output of external functions that you integrated into AutomationDesk before.

Breakpoints Viewer In the Breakpoints Viewer, you can see the [breakpoints](#) that you specified in the sequences of the opened project. The breakpoints are used when a sequence is executed in debug mode.

Inconsistencies Viewer In the Inconsistencies Viewer, you can see the inconsistencies found by the **Find Inconsistencies** command.

Related topics

Basics

[Using the Project Manager \(AutomationDesk Basic Practices\)](#)
[Working With Automation Blocks \(AutomationDesk Basic Practices\)](#)












Overview of Symbols

Introduction

AutomationDesk uses different symbols to display the most important information on an element in compressed form.







Main elements

The following table gives you brief descriptions of the symbols used for the main AutomationDesk elements:

Symbol	Element	Description
	Project	Represents the project's root node.
	Folder	Represents a folder element.
	Library folder	Represents a folder in the Library Browser.
	Custom library folder	Represents a folder in the custom library.
	Sequence	Represents a sequence.
	Automation block	Represents an automation block.
	Data container	Represents a structure element to group several data objects.
	Result	Represents a result in the Project Manager.
	Report (HTML)	Represents an HTML report in the Project Manager.
	Report (PDF)	Represents a PDF report in the Project Manager.
	Error message	Represents an error message in the Result Browser or a corrupted AutomationDesk block in the Sequence Hierarchy Browser.




















Data objects

The following table gives you brief descriptions of the symbols used for data objects:

Symbol	Data Object	Description
	Input / Output	Represents a project-specific data object in the Project Manager and Data Object Selector. The data type is represented by a character, for example, "F" for a float data object.
	Input	Represents a directly set input variable.
	Output	Represents a directly set output variable.
	Input	Represents a referenced input variable.
	Output	Represents a referenced output variable.
	Instantiated template	Represents a data object that is based on a template in a custom library.



Element states








The following table gives you brief descriptions of the symbols used for the different element states:

Symbol	State	Description
	Modified element	<p>The blue triangle indicates that the element is modified. This applies to elements that are saved to a separate XML file (ADPX, ADLX, BLKX, PY), i.e.:</p> <ul style="list-style-type: none"> ▪ In the Project Manager: Projects and sequences. ▪ In the Library Manger: Custom libraries, sequence templates, block templates and Python modules.
		
		
		
		
	Modified subelement	<p>The triangle with blue outlines indicates that a subelement was modified. The modified state is propagated to the entire hierarchy path of the modified element.</p>
		
		
		
	Linked element	<p>Indicates that the sequence is an instance of a custom library template. The link mode is also displayed:</p> <ul style="list-style-type: none"> ▪ Dynamic link mode ▪ Static link mode
		
	Disabled element	<p>Indicates that the element (folder, sequence or automation block) is excluded from execution.</p>
		
		
	Library operation mode	<p>Indicates that the library is in online, offline, or online recording mode.</p>
		
		
	Discontinued element	<p>Indicates that the element belongs to a discontinued built-in library.</p>
		

Checkout states









If you use a version control system, the checkout state of a project element is displayed via check marks. The following table gives you a brief description of the symbols used for the different checkout states:

Symbol		Meaning
Project	Custom Library	
		<p>The standard symbol of the element is displayed, if:</p> <ul style="list-style-type: none"> ▪ The element is not under version control or ▪ The element is under version control but its check-out state is undefined. One possible reason is that the version control system is not connected or you have set a different VC project type.

Symbol		Meaning
Project	Custom Library	
		The black point means that the element is checked out and can be modified.
		The black point with checkmark means that the element is checked in.
		The black padlock means that the element cannot be checked out, for one of the following reasons: <ul style="list-style-type: none"> ▪ The element is checked out by another user. ▪ The element is already checked out by you in another working folder.
A gray triangle () next to one of the above symbols means that there is a newer version in the repository than in AutomationDesk. For this triangle to be displayed, the version control application must provide the status information.		


Result states

The following table gives you brief descriptions of the symbols used for the different result states:

Symbol	Result State (Verdict)	Description
 For example: 	Passed	Indicates an element in the Result Browser that has passed the qualification of a Decision block. The decision result of a sequence is written to the folder shown in the Result Browser, and to the result node shown in the Project Manager.
 For example: 	Failed	Indicates an element in the Result Browser that has failed the qualification of a Decision block. The decision result of a sequence is written to the folder shown in the Result Browser, and to the result node shown in the Project Manager.
 For example: 	Undefined	Indicates an element in the Result Browser that has neither passed nor failed the qualification of a Decision block. The decision result of a sequence is written to the folder shown in the Result Browser, and to the result node shown in the Project Manager.
	Error	Indicates that the execution of an automation block failed. The result is written to the folder shown in the Result Browser, and to the result node shown in the Project Manager.
	Exception	Indicates that an exception is occurred.

Overview of Libraries

Introduction

AutomationDesk provides a wide range of elements for building automation programs. The automation functions and [data objects](#)  are represented

graphically by [automation blocks](#) in the [Library Browser](#). They are held in different sublibraries according to their fields of application.

Main Library

The Main Library provides three types of library elements:

- **Control Flows** to create the control flow for an automation task with various loop variants, conditional constructs and flow controls for serial and parallel processing
- **Basic Elements** to specify standard program elements, for example, a sleep function
- **Data Objects** to specify project-specific variables, for example, integer, float or string values

For more information, refer to [Main Library \(AutomationDesk Basic Practices\)](#).

Report

The Report library provides library elements to add user-defined content to a report:

- Add a text to a report
- Add an image to a report
- Add a 2-D plot to a report
- Add a table to a report
- Add a URL to a report
- Add a data object to a report

For more information, refer to [Report \(AutomationDesk Basic Practices\)](#).

Dialogs

The Dialogs library provides library elements to provide user interaction in your sequence. You can create:

- Message dialogs to provide execution messages which you want to be confirmed
- Input dialogs to make the manual modifications of values possible during the sequence execution

For more information, refer to [Dialogs \(AutomationDesk Basic Practices\)](#).


Evaluation


The Evaluation library provides library elements to evaluate test [results](#) during execution of a [sequence](#). You can:

- Prepare evaluation signals by extracting them from measured data, like a CaptureResult, or creating them using lists or Python expressions.
- Resample the evaluation signal by using various interpolation methods.
- Apply various mathematical operations to create a new evaluation signal based on one or two input evaluation signals.
- Apply various evaluation methods to the evaluation signal to get an evaluation result.
- Add the evaluation signals to the report as plot.

For more information, refer to [Evaluation](#) ([AutomationDesk Basic Practices](#) ).

Framework Builder

The Framework Builder library provides automation blocks for building high-level [templates](#)  with preconfigured automation tasks, such as the test cases provided by the Test Builder library.

For more information, refer to [Framework Builder](#) ([AutomationDesk Basic Practices](#) ).



Test Builder

The Test Builder library provides high-level automation blocks with predefined functionalities for test purposes.

The features of this library, such as result evaluation and result logging, are based on the Framework Builder library.


For more information, refer to [Test Builder](#) ([AutomationDesk Basic Practices](#) ).




Platform Management

The Platform Management library provides library elements for managing dSPACE real-time [platforms](#)  and [VEOS](#)  as the offline simulation platform. It is based on the dSPACE Platform Management API and lets you automate a subset of functions, which are also available via the Platform Manager in the AutomationDesk user interface. For example, you can get a list of registered platforms and load a simulation application to a selected platform.


For more information, refer to [Platform Management](#) ([AutomationDesk Accessing Simulation Platforms](#) ).

XIL API

The XIL API library provides library elements that are based on the [XIL API](#)  specified by the ASAM association. The AutomationDesk XIL API library supports:

- **MAPort (Model Access Port)**
Accessing the simulation platform for reading, writing, capturing and generating stimulus signals.
For more information, refer to [XIL API Framework](#) ([AutomationDesk Accessing Simulation Platforms](#) ) and [XIL API \(Model Access\)](#) ([AutomationDesk Accessing Simulation Platforms](#) ).
- **EESPort (Electrical Error Simulation Port)**
Accessing the simulation hardware for simulating errors, such as short circuits.
For more information, refer to [XIL API \(Electrical Error Simulation\)](#) ([AutomationDesk Simulating Electrical Errors](#) ).

XIL API Convenience

The XIL API Convenience library provides library elements which provide more convenient methods to use the standard features of the [ASAM XIL API](#)  than the XIL API library.

Thus, you can save several work steps by using an automation block of the XIL API Convenience library compared with using the XIL API library for the same purpose.

For more information, refer to [XIL API Convenience \(Model Access\) \(AutomationDesk Accessing Simulation Platforms !\[\]\(5eb1325dfdc3f1cad8426726c0db51cd_img.jpg\)\)](#) and [XIL API Convenience \(Electrical Error Simulation\) \(AutomationDesk Simulating Electrical Errors !\[\]\(312638b5686dbc3f6ff8424fd17b3fb2_img.jpg\)\)](#).

RS232

Note

The RS232 library is included in AutomationDesk 6.5 for the last time. For later versions of AutomationDesk, a custom library that provides the same functionality will be available on demand. Refer to <https://www.dspace.com/go/AUD-RS232-CustLib>.

The RS232 library provides library elements for communication via the serial interface:

- Configure the serial interface
- Send data to the serial interface
- Receive data from the serial interface

For more information, refer to [Automation Blocks \(AutomationDesk Accessing RS232 !\[\]\(e1d6102fe77919492c04879c8450f1f5_img.jpg\)\)](#).

Remote Diagnostics (COM)

The Remote Diagnostics (COM) library provides library elements to perform diagnostics on an electronic control unit (ECU). It lets you access ControlDesk with the installed ControlDesk ECU Diagnostics Module as the diagnostic tool that supports the ASAM MCD-3 D 2.0.2 standard.

For more information on the Remote Diagnostics (COM) library, refer to [Automation Blocks \(AutomationDesk Accessing Remote Diagnostics COM !\[\]\(35dc653d59570f8f891c312eeece91a2_img.jpg\)\)](#).

The Remote Diagnostics (COM) library supported *DTS V7* as an interface to a diagnostic system for the last time with dSPACE Release 2016-A. You must migrate your projects that used this interface to use ControlDesk as the diagnostic tool. For more information, refer to [General Aspects on Migrating Discontinued Libraries in AutomationDesk](#) on page 66.

Remote Calibration (COM)

The Remote Calibration (COM) library provides library elements to control ASAM MCD-3 MC(version 1.0)-compatible measurement and calibration (MC) systems remotely via the COM/DCOM interface.

For more information, refer to [Automation Blocks \(AutomationDesk Accessing Remote Calibration COM !\[\]\(b538fe54c1f3a7343e37e85cc2d00497_img.jpg\)\)](#).

ControlDesk Access

The ControlDesk Access library provides library elements to access [ControlDesk !\[\]\(21226b58c700e5231ab98d27101bac58_img.jpg\)](#).

This allows you to automate the following features:

- Calibration and measurement
- Diagnostics

For more information, refer to [Automation Blocks \(AutomationDesk Accessing ControlDesk !\[\]\(feabb98897b440bc8695a03336a6e2df_img.jpg\)](#)).

The ControlDesk Access library provided failure simulating features for the last time in dSPACE Release 2016-A. If your project or custom library contains elements of the library's Convenience - FailureSimulation or the Basic Functions - Application - FailureSimulation folders, you must replace those elements. For more information, refer to [General Aspects on Migrating Discontinued Libraries in AutomationDesk](#) on page 66.

ModelDesk Access

The ModelDesk Access library provides library elements to access [ModelDesk !\[\]\(8d0f0e0fe25b320c33272c52aec1fbca_img.jpg\)](#). It can be used to activate and download an experiment with its road, traffic scenario and maneuver elements. For testing purposes, you can modify parameter sets and control maneuvers.

For more information, refer to [Automation Blocks \(AutomationDesk Accessing ModelDesk !\[\]\(642aa997563f9a325b310230bb5078b7_img.jpg\)](#)).

MotionDesk Access

The MotionDesk Access library provides library elements to access [MotionDesk !\[\]\(3cb60d42b10e53f9522bb0b392c1c4cd_img.jpg\)](#). For example, it can be used to open a project and to start the animation.

For more information, refer to [Automation Blocks \(AutomationDesk Accessing MotionDesk !\[\]\(d0262bbe9d2356661a2e89321dfcc781_img.jpg\)](#)).


MATLAB Access

The MATLAB Access library provides library elements to provide access to MATLAB. You can:

- Open a MATLAB instance via AutomationDesk
- Exchange data between AutomationDesk and MATLAB in both directions
- Execute commands in MATLAB
- Work with data stored in MAT file format

For more information, refer to [Automation Blocks \(AutomationDesk Accessing MATLAB !\[\]\(274fd520e03b61c1b9ffc861754cacdc_img.jpg\)](#)).

Real-Time Testing

AutomationDesk provides the Real-Time Testing library to manage Real-Time Testing (RTT ) projects on dSPACE real-time platforms.

For more information, refer to [Automation Blocks \(AutomationDesk Accessing Real-Time Testing !\[\]\(683dba75afe26e28cd4de5730b776760_img.jpg\)](#)).

Signal-Based Testing

The Signal-Based Testing custom library provides a different concept for creating tests. Instead of implementing a control flow with automation blocks, you configure the signals to be used for stimulating and capturing in a test case description. The configuration includes the evaluation and reporting of the test results.

For more information, refer to [Signal-Based Testing Library \(AutomationDesk Implementing Signal-Based Tests !\[\]\(3d8c13c92b853674f749aac6fa869926_img.jpg\)](#)).

Custom Library

The custom library with the name Custom Library makes it possible to integrate user-defined [automation blocks !\[\]\(96cc62f861fdd6e50510c0224a756dff_img.jpg\)](#) in AutomationDesk. It can be used to store:

- User-defined sequences
This includes blocks that implement subsequences.
- User-defined automation blocks
- User-defined data objects

For detailed information, refer to [Custom Library \(AutomationDesk Basic Practices !\[\]\(17acf1afa8cdf0b67c53d4865a5ed469_img.jpg\)](#)).

Discontinued libraries

Over the time, some libraries are discontinued. Mostly, a new library provides newer methods to fulfill the same functionality. You can load older projects and custom libraries that contain elements from a discontinued library, but its elements will not function. They are replaced by Discontinued block and Discontinued data object elements. For an overview of discontinued libraries and information on migrating projects, refer to [General Aspects on Migrating Discontinued Libraries in AutomationDesk](#) on page 66.

Related topics**Basics**

[Packaging of AutomationDesk.....](#) 40

Packaging of AutomationDesk

Introduction

AutomationDesk and AutomationDesk - Automation Server need additional packages for executing.

Contents of the product set

The AutomationDesk product set includes the following products:

- Firmware Archives
The firmware archives provide the latest firmware for your dSPACE hardware.

- dSPACE Python Extensions
dSPACE Python Extensions provides the following dSPACE Test Automation Python modules, which you can use in AutomationDesk Exec blocks:
 - `matlablib2` for accessing MATLAB.
 - `rs232lib2` for accessing a serial interface of your PC.
- Real-Time Testing
Real-Time Testing is required for executing blocks of the Real-Time Testing library and when you are using blocks for stimulating variables or signals from the XIL API, XIL API Convenience and Signal-Based Testing libraries.
- dSPACE XIL API .NET
dSPACE XIL API .NET is required for executing blocks of the XIL API and XIL API Convenience libraries. It supports model access (MAPort) and electrical error simulation (EESPort).

After installation, you have to activate your licenses and decrypt the encrypted archives in the Installation Manager. For working with AutomationDesk you need the standard AutomationDesk licenses and additional license packages.

No packages required for creating tests

There is no additional package required for creating projects, libraries, and sequences.

You can structure your project with folders and sequences, add data objects to the project, and execute any command available in the context of the Project Manager.

You can also build automation sequences by adding automation blocks from each available library. You can parameterize the blocks and data objects, and create new block templates in the custom library.

You can execute automation sequences in offline operation mode independently of the installed additional packages. If you use the online operation mode or the online recording operation mode, execution stops with an error message, if a required additional package is not available.

The following libraries are contained in the standard packaging of AutomationDesk and AutomationDesk - Automation Server. Executing sequences containing blocks from these libraries do not require additional packages.


- Main Library
- Framework Builder
- Test Builder
- Dialogs
- Report
- Evaluation
- MATLAB Access
- RS232
- Remote Calibration (COM)
- Remote Diagnostics (COM)

- ControlDesk Access
You can only execute those functions which are supported by your ControlDesk installation.
- ModelDesk Access
- MotionDesk Access
- Platform Management


Additional packages to access dSPACE simulation platforms and failure insertion systems


The following packages are available:

Platform API Package The Platform API Package is required for executing blocks of the following AutomationDesk libraries:

- XIL API
The Platform API Package provides the license for model access via XIL API's MAPort. The implementation is based on dSPACE XIL API .NET. For further information, refer to [dSPACE XIL API Implementation Guide](#) .
- XIL API Convenience
The library is based on the XIL API library and requires the same package for model access.
- Signal-Based Testing
The Signal-Based Testing library provides variable stimulation and data capturing that is implemented by using the XIL API library.

Failure Simulation Package The Failure Simulation Package is required for executing blocks of the following AutomationDesk libraries:

- XIL API
The Failure Simulation Package provides the license for electrical error simulation via XIL API's EESPort. The implementation is based on dSPACE XIL API .NET. For further information, refer to [dSPACE XIL API Implementation Guide](#) .
- XIL API Convenience
The library is based on the XIL API library and requires the same package for electrical error simulation.

For information on the system requirements, refer to [Appendix \(Installing dSPACE Software\)](#) .

Demos for AutomationDesk

Introduction

When you start AutomationDesk for the first time, the demo projects are copied from the installation folder to the user-specific `<DocumentsFolder>`, i.e., for example, `C:\Users\<UserName>\Documents\dSPACE\AutomationDesk\<VersionNumber>`.

Basics on Demo Projects

Overview

You find demo projects and other examples for demonstrating how to use AutomationDesk in the following sub folders of your **Documents** folder.

Demo Projects Folder	Description
Advanced Demos	Contains projects and custom libraries for experienced users to demonstrate testing a turn signal on different platforms without changing any of the blocks in the sequences. You can adapt the device access by loading the different Device Access custom libraries.
API	Contains source code examples for automating tests via AutomationDesk's COM API. The scripts are available as C#, Visual Basic, and Python. For more information, refer to Application Examples (AutomationDesk Automation 📖) .
ControlDesk Access	Contains demo projects to demonstrate automating the access to ControlDesk. <ul style="list-style-type: none"> ControlDeskAccessExample.zip Provides a demo project with diagnostics features available when using the CalDemo ControlDesk project. ControlDesk_DiagnosticsExample.zip Provides a demo project with diagnostics features available when using the DiagDemo ControlDesk project. ControlDeskAccess_ExtensionExample.zip and Custom Ext ControlDesk Access.zip This demo project with its related custom library shows you how to extend a custom library with ControlDesk functionality.

Demo Projects Folder	Description
Custom Library	This folder does not provide demo projects. This is the default storage path for the Custom Library that is opened in AutomationDesk's Library Browser.
Custom Report Stylesheets	This folder does not provide demo projects. Here, you find example style sheet files (XSL) that you can use as templates for the customization of reports in AutomationDesk.
Dialog Library	Contains demo projects to demonstrate working with the input blocks and message blocks in the Dialogs library, and how to customize an Edit dialog for data objects.
Evaluation Library	Contains demo projects to demonstrate working with the blocks in the Evaluation library. As a special use case, the handling of timestamps is shown in a separate project.
Framework Builder Library	Contains demo projects to demonstrate working with the Framework Builder library. The demonstration frameworks, which only provide skeletons for automation tasks, are available in the related custom libraries.
FrameworkConfiguration	This folder does not provide demo projects. Here, you find XML files that show you how to work with an XIL API framework according to the ASAM AE XIL API standard. There are configuration files for the framework, MAPorts, and label mappings.
Main Library	Contains demo projects to demonstrate working with the blocks in the Main Library. These blocks let you implement the control flow of your test. For the most relevant blocks you find implementation examples.
MATLAB Access Library	Contains a demo project to demonstrate automating the access to MATLAB. You find examples of using MATLAB as remote-controlled application, and instructions how to read and write data in MATFile format.
ModelDesk Access	Contains a demo project to demonstrate automating the access to ModelDesk. The MOD_Vehicle ASM ModelDesk demo project is used, for example, to show how to run maneuvers in ModelDesk.
MotionDesk Access	Contains a demo project to demonstrate automating the access to MotionDesk. The MotionDesk_Vehicle MotionDesk demo project is used, for example, to show how to start an animation in MotionDesk.
Platform Management	Contains a demo project to demonstrate automating the management of a platform, e.g., loading and starting a simulation application on the registered platform.
RealTimeTesting	Contains a demo project to demonstrate automating the access to Real-Time Testing.
Remote Calibration (COM)	Contains demo projects to demonstrate remote-controlling measurement and calibration systems, such as ControlDesk, INCA, or CANape.
Remote Diagnostics (COM)	Contains a demo project to demonstrate remote-controlling ControlDesk as diagnostics system.
Report Library	Contains demo projects to demonstrate working with the blocks in the Report library and customizing report plots.
SignalBasedTesting	Contains demo projects to demonstrate using the signal-based test concept with the turn signal test as an example.
Test Builder	Contains demo projects to demonstrate using the Test Builder library for different experience levels. The features of this library, such as result evaluation and result logging, are based on the Framework Builder library.

Demo Projects Folder	Description
TurnSignalTests	This folder does not provide demo projects. Here you find required data to work with Real-Time Testing. It also contains the simulation applications for several platforms, which are used in the demo projects with platform access.
Tutorial Demos	Contains the demo projects that are described in the AutomationDesk Tutorial. In contrast to the other demo projects, the Tutorial demos can be directly opened in AutomationDesk. They must not be imported as ZIP file.
XIL API	Contains demo projects to demonstrate using the blocks of the XIL API library for model access with the turn signal tests as an example, and for stimulating signals.
XIL API Convenience	Contains demo projects to demonstrate using the blocks of the XIL API Convenience library for model access, electrical error simulation, and working with the XIL API framework.

Tip

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 !\[\]\(bd1a142de767a21e5362c595f844a4ff_img.jpg\)](#)).

Opening a demo project

The AutomationDesk demo projects are stored as ZIP files. You must use the **Import** command to open them.

For a fast access to the demo projects, click **Import Demo Project** on AutomationDesk's **Start Page**.

There are **Readme** data objects in the demo projects with a short description and some instructions if necessary.

Tutorials, Tutorial Videos, and PDF Documents

Introduction Here you get an overview on information for AutomationDesk beginners, videos for specific AutomationDesk features, and documents in PDF format for printing.

Where to go from here

Information in this section

[Tutorial for AutomationDesk.....](#)47

The AutomationDesk installation provides a tutorial for AutomationDesk beginners.

[Tutorial Videos for AutomationDesk.....](#)48

The dSPACE website provides tutorial videos for AutomationDesk.

[PDF Documents for AutomationDesk.....](#)48

Provides an overview of the PDF documents for AutomationDesk users.

Tutorial for AutomationDesk

Contents of the tutorial


The tutorial gives you basic information before you start working with AutomationDesk. It gives you step-by-step instructions for implementing automation tasks. It shows you how to implement control-flow-based tests and signal-based tests. It starts with example tasks that do not require a registered platform. Later on, the example tasks are to be executed on a dSPACE platform or VEOS.

Refer to [AutomationDesk Tutorial](#) .

Tutorial Videos for AutomationDesk

Tutorial video

The following tutorial video for AutomationDesk is available on the dSPACE website. Refer to [AutomationDesk Tutorial videos](#).

Tutorial Video	Description
Signal-Based Testing	<p>This AutomationDesk tutorial video provides basic information on signal-based testing and specific instructions on how to use the Signal-Based Testing library and the Signal Editor.</p> <p>For more information on signal-based testing, refer to AutomationDesk Implementing Signal-Based Tests .</p>

Note

- Accessing tutorial videos requires a mydSPACE login.
- The tutorial videos might show the handling in an earlier AutomationDesk version. If the handling in your AutomationDesk version is different, refer to the installed user documentation.

Public product videos

For public product videos, refer to [AutomationDesk product videos](#).

Related topics

Basics

[Tutorial for AutomationDesk](#)..... 47

PDF Documents for AutomationDesk

Overview

The following table shows the PDF documents available for AutomationDesk:

Subject	PDF Document
Introduction and overview	AutomationDeskIntroductionAndOverview.pdf¹⁾
Tutorial	AutomationDeskTutorial.pdf¹⁾
Basic practices	
AutomationDesk Basic Practices	AutomationDeskBasicPractices.pdf
AutomationDesk Implementing Signal-Based Tests	AutomationDeskImplementingSignalBasedTests.pdf
AutomationDesk Accessing Simulation Platforms	AutomationDeskAccessingSimulationPlatforms.pdf

Subject		PDF Document
	AutomationDesk Simulating Electrical Errors	AutomationDeskSimulatingElectricalErrors.pdf
Accessing external devices		
	AutomationDesk Accessing ControlDesk	AutomationDeskAccessingControlDesk.pdf
	AutomationDesk Accessing MATLAB	AutomationDeskAccessingMATLAB.pdf
	AutomationDesk Accessing ModelDesk	AutomationDeskAccessingModelDesk.pdf
	AutomationDesk Accessing MotionDesk	AutomationDeskAccessingMotionDesk.pdf
	AutomationDesk Accessing Real-Time Testing	AutomationDeskAccessingRealTimeTesting.pdf
	AutomationDesk Accessing Remote Calibration COM	AutomationDeskAccessingRCCOM.pdf
	AutomationDesk Accessing Remote Diagnostics COM	AutomationDeskAccessingRDCOM.pdf
	AutomationDesk Accessing RS232	AutomationDeskAccessingRS232.pdf
Automation		AutomationDeskAutomation.pdf
Application Notes		
	Using COM in AutomationDesk	UsingCOMInAutomationDeskApplicationNote.pdf
	Using VirtualCOM in AutomationDesk	UsingVirtualCOMInAutomationDeskApplicationNote.pdf

¹⁾ A printed copy of this document is available on demand. You can order it free of charge by using the following link:
<http://www.dspace.com/go/requestreleasematerial>

New Features of AutomationDesk

Introduction	Gives you an overview on the new features introduced with AutomationDesk 6.5.
--------------	---

Where to go from here

Information in this section

New Features of AutomationDesk 6.5	51
--	----

Information in other sections

New Features and Migration
Provides information on the new features of all the dSPACE software products in the current dSPACE Release. It also gives you an overview of software products with no or minor changes. There are instructions on migrating from older dSPACE Releases, especially from older product versions, if required.

New Features of AutomationDesk 6.5

General enhancements	Python 3.9 support Python scripts that are used in AutomationDesk projects are automatically migrated when you open the projects. For the required manual migration steps and general information on the migration, refer to Migrating Python Scripts from Python 3.6 to Python 3.9 on page 62.
----------------------	--

Compatibility, Migration, and Discontinuations

Introduction	Gives you information on compatibility issues and migration instructions for changed and discontinued features.
--------------	---

Where to go from here

Information in this section

Compatibility.....	54
Migration.....	58

Compatibility

Where to go from here

Information in this section

Compatibility of AutomationDesk 6.5.....	54
To give you an overview of dSPACE software products that are compatible with AutomationDesk 6.5.	
Compatibility of Firmware.....	56
To provide guidelines to determine the correct firmware version.	

Compatibility of AutomationDesk 6.5

General product compatibility

AutomationDesk 6.5 is compatible with other dSPACE software products. Compatibility means that different products can be used in parallel after software installation.

Note

dSPACE recommends using only software products from the same dSPACE Release. This ensures maximum run-time compatibility.

Working with real-time applications

AutomationDesk 6.5, which is part of dSPACE Release 2021-A, lets you work with real-time applications on dSPACE hardware, such as the DS1007 or SCALEXIO, if these applications were built with one of the three previous dSPACE Releases.

Working with a real-time application means:


- Downloading it to the hardware
- Reading, writing, or stimulating variables of the application

Note

It will not be possible to use AutomationDesk 6.5 from dSPACE Release 2021-A to access applications generated with products from future dSPACE releases.

For details, refer to [Run-Time Compatibility of dSPACE Software \(Installing dSPACE Software !\[\]\(5abce1a84a655b073239ab33e1199487_img.jpg\)](#)).

Working with Real-Time Testing	<p>As of dSPACE Release 2021-A, AutomationDesk and Real-Time Testing support Python 3.9. The activated Real-Time Testing version and the dSPACE software using Real-Time Testing, e.g., AutomationDesk, must support the same Python version. Therefore, you cannot use the following products in combination:</p> <ul style="list-style-type: none"> AutomationDesk 6.5 or later with Real-Time Testing 4.4 or earlier. Real-Time Testing 5.0 with AutomationDesk 6.4 or earlier.
	<p>Note</p> <p>If you use a SCALEXIO system, MicroLabBox, MicroAutoBox III, DS1007, or VEOS, the firmware determines the activated Real-Time Testing version.</p>
Working with a SCALEXIO system	<p>The products for working with a SCALEXIO system and with MicroAutoBox III must be compatible. This is guaranteed only for products delivered with the same dSPACE Release. Contact dSPACE for more information.</p>
Working with VEOS	<p>VEOS 5.2 supports only AutomationDesk 6.5 from dSPACE Release 2021-A.</p>
Working with ModelDesk	<p>Up to and including ModelDesk 5.2, you can use the maneuver compatibility mode in ModelDesk to work with maneuvers created with ModelDesk 4.6 and earlier.</p> <p>As of ModelDesk 5.3, the maneuver compatibility mode is discontinued. Maneuvers created with ModelDesk 4.6 and earlier are no longer supported. Executing ModelDesk Access blocks to activate and download these maneuvers leads to exceptions.</p>
Working with the Signal Editor	<p>To work with AutomationDesk's Signal Editor:</p> <ul style="list-style-type: none"> The RTT versions used for building the real-time application and used by the Signal Editor must be identical. The RTT version must be 1.7.1 or higher.
Working with measurement and calibration systems	<p>The Remote Calibration (COM) library supports servers that are compatible to the ASAM MCD-3 (version 1.0) standard. With AutomationDesk 6.5 the following measurement and calibration systems are tested:</p> <ul style="list-style-type: none"> ControlDesk 7.4 INCA 7.0.0, MC3Server 15.70.0 CANape 17, VMC3Server 1.0.75

Working with diagnostics systems	<p>The Remote Diagnostics (COM) library supports servers that are compatible to the ASAM MCD-3 D 2.0.2 standard. With AutomationDesk 6.5 the following diagnostics system is tested:</p> <ul style="list-style-type: none"> ▪ ControlDesk 7.4 with the ControlDesk ECU Diagnostics Module.
Working with version control	<p>With the AutomationDesk version control interface, you can integrate version control systems that offer a Microsoft Source Code Control (SCC) interface. Since AutomationDesk 5.3 a 64-bit version of the SCC interface is required.</p> <p>The integration of the following version control systems in AutomationDesk is tested:</p> <ul style="list-style-type: none"> ▪ PTC® Integrity Client 12, (12.1.0.2) Build 2, API Version 4.16.2 ▪ Apache® Subversion 1.7.9 (Open Source software that you can download from http://subversion.apache.org) (Plug-in required for the SCC interface) ▪ Microsoft Team Foundation Server, Version 2013
Supported MATLAB Releases	<p>AutomationDesk 6.5 supports real-time applications only if they were built with one of the supported MATLAB Releases.</p> <p>For details, refer to Required MATLAB Releases (Installing dSPACE Software ).</p>

Compatibility of Firmware

Introduction	To provide guidelines to determine the correct firmware version.
General guideline	dSPACE recommends using only software products from the same dSPACE Release. This ensures maximum run-time compatibility.
Firmware compatibility guidelines	<p>Firmware and real-time application Firmware is downward-compatible. 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.

Hardware dependency of the required firmware version

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

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

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

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

Migration

Where to go from here

Information in this section

Migrating AutomationDesk.....	58
Information on useful preprocessing and postprocessing in addition to the automatic project update.	
Migrating Python Scripts from Python 3.6 to Python 3.9.....	62
Information on migrating Python scripts when using dSPACE software with Python support, for example, AutomationDesk or ControlDesk.	
Migrating Discontinued AutomationDesk Libraries.....	66
Information on migrating your projects and custom libraries that contain discontinued elements.	

Migrating AutomationDesk

Introduction

After you install a newer version of AutomationDesk, projects and custom libraries are automatically updated after confirmation when they are loaded or imported. However, there are some points to note when preparing your projects and custom libraries.

Where to go from here

Information in this section

General Migration Aspects.....	58
Provides information on preconditions to be fulfilled before starting a migration.	
Library-Specific Migration Aspects.....	60
Provides information on migration aspects for certain AutomationDesk libraries.	

General Migration Aspects

Introduction

Provides information on preconditions to be fulfilled before starting migration.

Preconditions to be fulfilled

If you open an AutomationDesk project with a newer AutomationDesk version, the software automatically detects whether migration is necessary. When you click OK in the message dialog, migration is started. If you also want to continue working with the old project, you should not overwrite it with the migrated project, because the versions are not downward compatible. Save the migrated project to another path or name.

Note

- It is strongly recommended to make backup copies of your projects and custom libraries.
- The automatic migration does not support projects created with AutomationDesk 1.x.
- The AutomationDesk projects and custom libraries to be migrated must be writable. If you have manually set the files to read-only, clear the flag before you open the project or custom library in AutomationDesk. If you are using a version control system, refer to [How to Migrate Projects or Custom Libraries Under Version Control \(AutomationDesk Basic Practices !\[\]\(aca6fcc8bd95e8255b9ea1b1d08ef300_img.jpg\)](#)).

Before you open an older project with the new AutomationDesk version, ensure the following preconditions are fulfilled:

- You must create backups of the project and of the linked custom libraries.
- AutomationDesk must be running properly. There must not be any error messages displayed in the Message Viewer.
- The built-in libraries, required custom libraries and other packages must be correctly loaded.

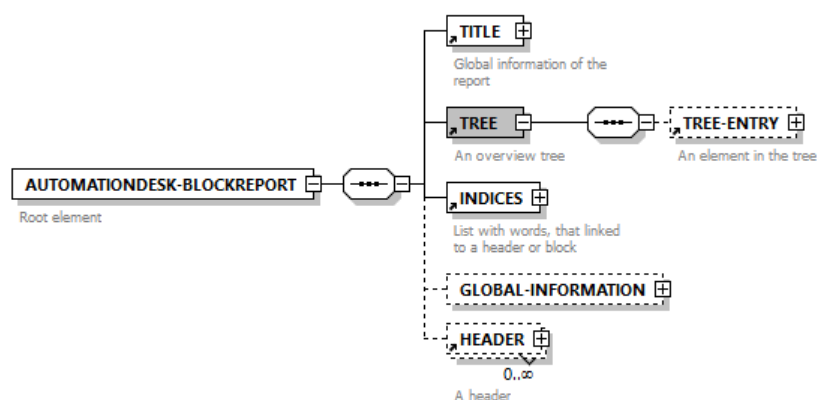
Projects and custom libraries under version control

If you use a version control system, you can get and open projects and custom libraries in the legacy formats. They are automatically migrated to the new serialization format and loose their connections to the version control system. You must add the migrated projects and custom libraries to new version control projects to avoid mixing the formats in the repository. For more information, refer to [How to Migrate Projects or Custom Libraries Under Version Control \(AutomationDesk Basic Practices !\[\]\(3e2231b1ad3ca8da8658228c00dd08e0_img.jpg\)](#)).

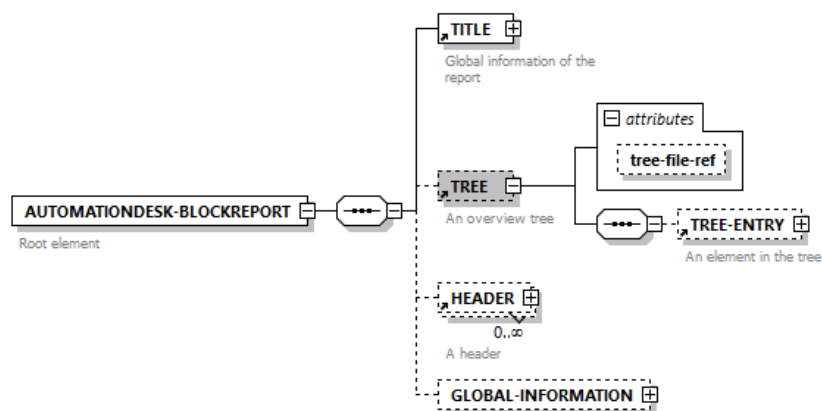
Migrating custom stylesheets used by AutomationDesk 6.3 or earlier versions

If you use custom stylesheets for your HTML or PDF reports, and you use the TREE element for the navigation elements, you must adapt them.

Up to and including AutomationDesk 6.3, the XML schema for the AUTOMATIONDESK-BLOCKREPORT element contained five elements. The TREE element was directly filled with content by TREE-ENTRY elements.



As of AutomationDesk 6.4, the XML schema for the **AUTOMATIONDESK-BLOCKREPORT** element contains only four elements. The **INDICES** element is removed. The **TREE** element now provides an attribute for specifying a separate file with the content of the required tree entries.



If you do not migrate the **TREE** elements, the navigation in the generated HTML report or the PDF bookmarks are not available or set to default values.

For an example, refer to the custom stylesheets available in the AutomationDesk demo folder.

Library-Specific Migration Aspects

Introduction

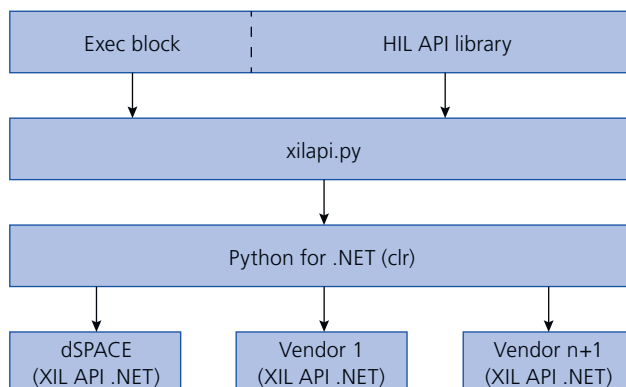
Provides information on migration aspects for certain AutomationDesk libraries.

XIL API

With AutomationDesk 5.0, the HIL API library and the HIL API Convenience library are replaced by the XIL API library and the XIL API Convenience library. The XIL API libraries are based on the dSPACE XIL API .NET implementation, refer

to [dSPACE XIL API Implementation Guide](#) and [dSPACE XIL API Reference](#). For an overview of the software architecture of AutomationDesk 5.0, refer to the following illustration:

AutomationDesk 5.0 and later



The blocks from the HIL API library or the HIL API Convenience library are automatically migrated to the XIL API blocks. Only scripts that you use in Exec blocks have to be modified. Search for `hilapi` and replace it by `xilapi`.

For example, replace `import hilapi` by `import xilapi`.

Remote Diagnostics (COM) library

When you use ControlDesk 5.1 or later, accessing an ECU is possible only via the `ControlDeskNG.D3system202` interface that uses the configuration of an ODX database contained in a ControlDesk experiment. As a consequence, configuring logical links via the `ConfigureLogicalLink` automation block is not required any more.

You have to migrate AutomationDesk projects which use the following diagnostic interfaces:

- `ControlDeskNG.DSystem`
 - `ControlDeskNG.D3System.201`
1. You have to migrate your experiments. For instructions, refer to [Migrating from Prior Versions of ControlDesk](#) ([ControlDesk Introduction and Overview](#)).
 2. Change the interface in your AutomationDesk project (see [How to Set Up Diagnostic Projects](#) ([AutomationDesk Accessing Remote Diagnostics COM](#))).
 3. Delete the `ConfigureLogicalLink` block (if present) in your AutomationDesk project.

Test Builder library

The Test Builder library supports the changed verdict handling with AutomationDesk 4.0.

The changed verdict handling requires enhancements for some block interfaces. Therefore, you have to synchronize your TestCase sequences and standard sequences containing Test Builder blocks with the Test Builder library before using them.

Migrating Python Scripts from Python 3.6 to Python 3.9

Where to go from here

Information in this section

[Main Changes in Python 3.9](#)..... 62

The end of life of Python 3.6 is scheduled for the end of December 2021. Therefore, dSPACE switches to Python 3.9 with dSPACE Release 2021-A.

[Main Changes in Handling Python 3.9 with dSPACE Software](#)..... 63

With Python 3.9, some aspects of handling Python with dSPACE software have changed.

[General Information on Using Python Installations](#)..... 64

If you work with dSPACE software from dSPACE Release 2020-B or earlier, which supports Python 3.6, and dSPACE software from dSPACE Release 2021-A or later, which supports Python 3.9, both Python versions are installed on the PC and can be used in parallel.

[Technical Changes](#)..... 65

Migrating from Python 3.6 to Python 3.9 has caused minor changes in the Python API.

Main Changes in Python 3.9

Main reason for using Python 3.9 instead of Python 3.6

The end of life of Python 3.6 is scheduled for the end of December 2021. Therefore, dSPACE switches to Python 3.9 with dSPACE Release 2021-A.

Related documents available on the Python website

For a short overview of the main changes in Python 3.9, refer to [Technical Changes](#) on page 65. For information on all the changes in the Python language and environment from Python 3.6 to Python 3.9, refer to Python Software Foundation at www.python.org.

The Python Software Foundation provides the following documents:

- What's New from Python 3.6 to 3.7:
<https://docs.python.org/3.7/whatsnew/3.7.html>

- What's New from Python 3.7 to 3.8:
<https://docs.python.org/3.8/whatsnew/3.8.html>
- What's New from Python 3.8 to 3.9:
<https://docs.python.org/3.9/whatsnew/3.9.html>

Related topics

Basics

Technical Changes..... 65

Main Changes in Handling Python 3.9 with dSPACE Software

Introduction

With Python 3.9, some aspects of handling Python with dSPACE software have changed.

Libraries

The libraries and components used with Python 3.9 and distributed on dSPACE DVDs have changed as shown in the following table.

Package	Python 3.6 (Release 2020-B)	Python 3.9 (Release 2021-A)
comtypes	1.1.7	1.1.7
Core	3.6.8	3.9.1
cycler	0.10.0	0.10.0
future	0.18.2	0.18.2
grpcio	1.29.0	1.34.0
grpcio_tools	1.29.0	1.34.0
kiwisolver	1.2.0	1.3.1
lxml	—	4.6.2
matplotlib	3.2.1	3.3.3
numpy	1.18.4	1.19.3
pillow	7.1.2	8.0.1
pip	20.1.1	20.3.1
protobuf	3.12.2	3.14.0
pycparser	—	2.20
pyglet	1.5.5	1.5.11
pyarsing	2.4.7	2.4.7
pypubsub	4.0.3	4.0.3
Python-dateutil	2.8.1	2.8.1
pythonnet	2.4.2	2.5.3

Package	Python 3.6 (Release 2020-B)	Python 3.9 (Release 2021-A)
pytz	2020.1	2020.4
pywin32	227.10	300.10
scipy	1.4.1	1.5.4
six	1.15.0	1.15.0
wxPython	4.1.0	4.1.1
yapsy	1.12.2	1.12.2

General Information on Using Python Installations

Introduction

If you work with dSPACE software from dSPACE Release 2020-B or earlier, which supports Python 3.6, and dSPACE software from dSPACE Release 2021-A or later, which supports Python 3.9, both Python versions are installed on the PC and can be used in parallel.

Limitations when using Python 3.6 and Python 3.9 in parallel

You can use both Python versions in parallel. However, you must observe the following limitations:

- You can set the file associations for PY and PYW files to only one Python version. This is usually the latest Python version you installed.

Note

If you install dSPACE Release 2021-A including SystemDesk, Python 3.6 will be installed at last. Therefore, PythonWin 3.6 is registered to open Python files. To set the file associations to PythonWin 3.9, execute the **PythonInstaller.exe** from `<DVDRoot>\Products\Common\Python3.9` again or use the **Edit with PythonWin 3.9** command in the context menu of a Python file.

- Environment variables are used by both Python versions. You must set their values, for example, for PYTHONHOME, to the Python installation you want to work with. For an overview of environment variables set by Python, refer to: <https://docs.python.org/3.9/using/windows.html>.

Note

To avoid unintentional effects, which can be difficult to identify, you are recommended to not use environment variables. For the same reason, you should not extend the system path by a Python installation path.

Using dSPACE test automation with both Python versions in parallel

If a test automation script uses dSPACE Python Modules and you do not want to migrate the script, you have to work with both Python versions in parallel. The dSPACE Python Modules for Python 3.6 are available up to and including Release 2020-B.

Note

It is recommended to use only dSPACE software from the same Release when using Python scripts. Using both Python versions in the same application context, such as automating the access to an older version of ControlDesk via AutomationDesk or using AutomationDesk with an activated Real-Time Testing version that does not support Python 3.9, might cause a conflict.

Technical Changes

Introduction

Migrating from Python 3.6 to Python 3.9 has caused minor changes in the Python API.

Note

Independently of the number of required manual modifications, you must test the migrated code.

Migration issues

The following changes in Python 3.9 require manual migration of your Python scripts.

Python.NET: Return value of `ICollection` type is converted to `PyList` Since Python.NET 2.4, return values of `System.Collections.Generic.ICollection` type have been converted to the `PyList` type. Because some methods and properties of an `ICollection` object are not available with a `PyList` object, dSPACE provided the modified Python.NET 2.4.1 and 2.4.2 packages, which suppressed the conversion.

As of Python.NET 2.5.3, the modifications by dSPACE are no longer provided. You must therefore migrate the methods and properties of an `ICollection` object to the related methods and properties of a `PyList` object. For example, you must use `len(MyObject)` instead of `MyObject.Count`.

Note

The created Python list is a copy of the .NET object. Modifications to the Python list therefore have no effect on the associated .NET object. To apply the modifications to the .NET object, assign the Python list afterward.

ctypes: Loading library after changing the PATH variable Loading a library via `ctypes` can fail if the path to the library was added to the PATH variable beforehand.

For loading a library in a safe manner, use `ctypes.WinDLL(NameOfDll, winmode=8)` instead of `ctypes.windll.LoadLibrary(NameOfDll)`.

Migrating Discontinued AutomationDesk Libraries

Introduction

If your projects and custom libraries contain elements from discontinued libraries, this section gives you useful information for handling them in newer versions of AutomationDesk.

Where to go from here

Information in this section

General Aspects on Migrating Discontinued Libraries in AutomationDesk.....	66
Information on the migration of discontinued AutomationDesk libraries.	
Migrating Blocks of the Platform Access Library.....	71
Information on the migration to use the XIL API.	

General Aspects on Migrating Discontinued Libraries in AutomationDesk

Introduction

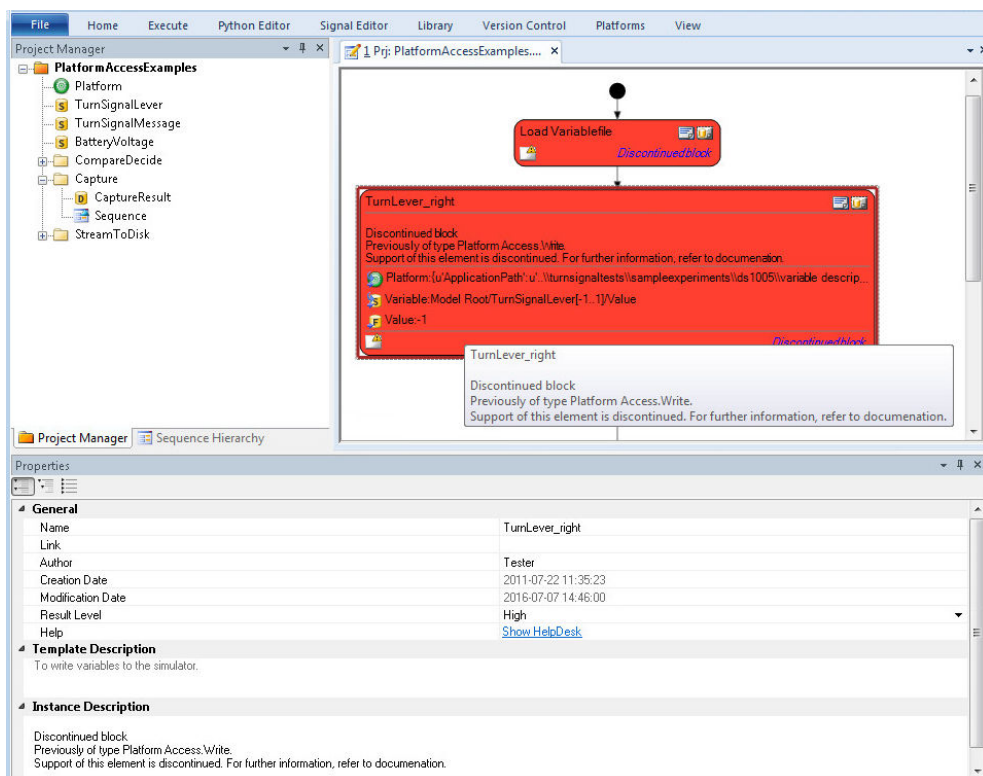
Over the time, some libraries are discontinued. Mostly, a new library provides newer methods to fulfill the same functionality.

You can load projects and custom Libraries in AutomationDesk that contain elements from these libraries, but the elements do not function. Execution stops with an error message.

To make your project executable again, you must delete or replace the discontinued elements.

Handling projects with discontinued libraries or blocks

Identifying discontinued features If you open a project that contains elements of a discontinued library, the block and data object icons are replaced by icons colored in red. The illustration below shows a Platform Access project imported in AutomationDesk.



The elements keep their names and provide information on their functionality within the project, for example:

- Type information, contained in the object's Instance Description property and displayed in its tooltip
- Further information such as reason for discontinuation, hints on alternatives
- The discontinued data objects do not provide information on their former parameters or configuration.
- The discontinued automation blocks provide the former parameter interface with the last used references and values.

Searching for discontinued elements You can find discontinued elements in your project or your custom library by running a search for 'discontinued' in the Find dialog (see [Find \(AutomationDesk Basic Practices\)](#))).

Executing and debugging You can disable single discontinued automation blocks to exclude them from execution and debugging (see [Disable \(AutomationDesk Basic Practices\)](#))).

AutomationDesk API and discontinued data objects

The AutomationDesk API provides remote access to AutomationDesk. If you load a project created with an earlier version of AutomationDesk via API (open or import), the automatic project update is executed.

If you use an API script that creates or provides access to project-specific data objects, it might access instances of objects that are discontinued. When you execute an API script that loads a project containing discontinued data objects, it might aborts with an exception. You can identify the discontinued data objects by using the **Type** property. Discontinued data objects return **adDeprecated** (=55) as the element type. You can find the discontinued data objects in your API script or AutomationDesk project in this way and delete them.

For general information on the AutomationDesk API, refer to [Using the AutomationDesk API](#) (AutomationDesk Automation .

Discontinued libraries and blocks


The following libraries and blocks have been discontinued:

Library/Block	Discontinued with AutomationDesk Version ...	Migrate To ...
Platform Access	5.3	XIL API Convenience - Model Access Port For more information, refer to Migrating Blocks of the Platform Access Library on page 71.
ControlDesk NG Access - ... - Failure Simulation	5.3	XIL API Convenience - Electrical Error Simulation Port For more information, refer to Migrating the discontinued failure simulation features of the ControlDesk Access library on page 69.
Test Framework	5.3	Test Builder For more information, refer to Migrating the discontinued Test Framework library on page 69.
Remote Diagnostics (COM) for DTS7	5.3	Remote Diagnostics (COM) via ControlDesk For more information, refer to Migrating the discontinued remote diagnostics via the DTS7 interface on page 69.
XIL API: ▪ InitCaptureResultIDFReader ▪ InitCaptureResultIDFWriter	5.3	XIL API: ▪ CaptureResultReader for accessing MDF files ▪ CaptureResultWriter for accessing MDF files For more information, refer to Migrating the discontinued XIL API library blocks that initialize access to IDF files on page 69.
CANscope	6.3	No migration provided.
CANstress	6.3	No migration provided.
MATLAB Access: ▪ MinimizeCommandWindow ▪ RestoreCommandWindow ▪ ShowApplication	6.5	No migration provided.

Library/Block	Discontinued with AutomationDesk Version ...	Migrate To ...
▪ HideApplication		

Migrating the discontinued failure simulation features of the ControlDesk Access library

The ControlDesk Access library provided failure simulating features for the last time in dSPACE Release 2016-A. If your project or custom library contains elements of the library's Convenience - FailureSimulation and Basic Functions - Application - FailureSimulation folders you must replace those elements.

With AutomationDesk 5.0, the electrical error simulation port (EESPort) of the ASAM XIL API standard is supported by the XIL API library and the XIL API Convenience library. The EESPort blocks of the XIL API Convenience library provide a future-proof access to failure simulation hardware. For more information on the XIL API electrical error simulation, refer to [AutomationDesk Simulating Electrical Errors](#) .

For help on migration, refer to <http://www.dspace.com/go/pscta>.





Migrating the discontinued Test Framework library

AutomationDesk provided the Test Framework library for the last time with dSPACE Release 2016-A. You must migrate your projects to use its successor, the Test Builder library. For this migration, you can use a tool that is provided by dSPACE Support. For more information, refer to <http://www.dspace.com/go/TestBuilderMigration>.

Migrating the discontinued remote diagnostics via the DTS7 interface

The Remote Diagnostics (COM) library supported *DTS V7* as an interface to a diagnostic system for the last time with dSPACE Release 2016-A. As an alternative, you can migrate your projects to use ControlDesk with the ECU Diagnostics Module.

As a prerequisite, you must perform the following steps in ControlDesk:

- You must create a ControlDesk project and an experiment. For more information, refer to [How to Define a Project \(ControlDesk Project and Experiment Management\)](#)  and [How to Define an Experiment \(ControlDesk Project and Experiment Management\)](#) .
- You must create and configure an ECU Diagnostics device. For more information, refer to [How to Add a Platform/Device to an Experiment \(ControlDesk Platform Management\)](#)  and [How to Configure an ECU Diagnostics Device \(ControlDesk ECU Diagnostics\)](#) .

If you need further help, contact dSPACE support.

Migrating the discontinued XIL API library blocks that initialize access to IDF files

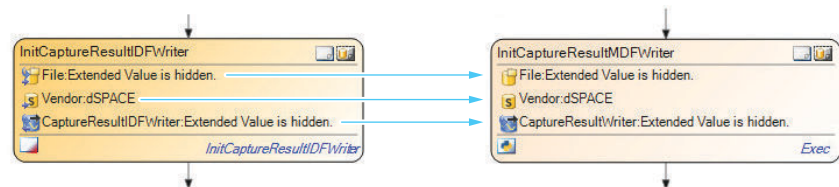
When working with the XIL API, the preferred format for files to store captured results is the ASAM Measured Data Format (MDF). AutomationDesk uses the 4.1

format that is represented by the file extension **MF4**. Thus, the blocks for initializing readers and writers to access IDF files have been discontinued.

Migrating InitCaptureResultIDFWriter blocks Replace each InitCaptureResultIDFWriter block with an Exec block that provides a File, a String and a CaptureResultReader data object as parameters. Rename the String data object **Vendor**, and set its value to **dSPACE** or reference it to a global String data object with the vendor information. As the Exec block's Python source, enter:

```
_AD_.CaptureResultWriter = xilapi.InitCaptureResultWriter(
    _AD_.Vendor,          # vendor = "dSPACE"
    "CaptureResultMDFWriter", # CaptureResultWriterType
    _AD_.File.GetAbsolutePath() ) # the instance-specific
                                # capture result MDF file
                                # with the file extension MF4
```

Parameterize the Exec block in the same way as the InitCaptureResultIDFWriter block:



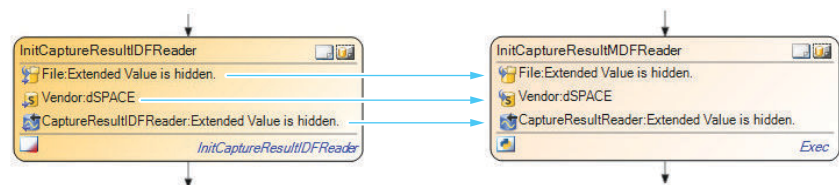
Migrating InitCaptureResultIDFReader blocks Replace each InitCaptureResultIDFReader block with an Exec block that provides a File data object, a String data object, and a CaptureResultReader data object as parameters.

Rename the String data object **Vendor**, and set its value to **dSPACE** or reference it to a global String data object with the vendor information.

As the Exec block's Python source, enter:

```
_AD_.CaptureResultReader = xilapi.InitCaptureResultReader(
    _AD_.Vendor,          # vendor = "dSPACE"
    "CaptureResultMDFReader", # CaptureResultReaderType
    _AD_.File.GetAbsolutePath() ) # the instance-specific
                                # capture result MDF file
                                # with the file extension MF4
```

Parameterize the Exec block in the same way as the InitCaptureResultIDFReader block:



Migrating Blocks of the Platform Access Library

Introduction

To make your project or custom library that contains elements of the Platform Access library executable again, you can replace the discontinued elements with elements of the XIL API Convenience library.

Basic migration steps

When you open an AutomationDesk project or a custom library that contains Platform Access library elements, the data objects of a discontinued type are migrated automatically. The discontinued automation blocks are replaced by DiscontinuedBlock elements that you must migrate manually. If there are Exec blocks in your project or library that use the **rtp1ib2** module to access platforms, you must also migrate these blocks manually.

Automatically migrated library elements Data objects of types that are specific for the Platform Access library are automatically converted to data objects of a further continued type, according to the following table. The data objects' names remain the same.

Platform Access Data Object Type	Migrated To	Description
Platform	MAPortConfiguration	The converted MAPortConfiguration data object contains the platform's name and the simulation application's SDF file name and path.
Variable	String	The value of the converted String data object contains the variable's model path.
CaptureResult	Dictionary	The converted Dictionary data object contains key-value pairs as <VariableName>:<Chronological list of captured values>. The former capture results are converted only to avoid information loss. The Dictionary data objects are not meant to be used for capture results in the migrated sequences.

Accordingly, also the RecordedData data object of the GetRecordedData block that is provided by the ControlDesk Access library is converted to a Dictionary data object.

Migrating DiscontinuedBlock elements The automation blocks of the Platform Access library cannot be replaced one-to-one with XIL API Convenience library blocks. For migration, you must consider the task that each block is involved in as shown in the following table:

Block of Platform Access Library	Refer To
InitPlatformAccess LoadVariablefile Disconnect	Migrating the platform access initialization on page 72
Read ReadMatrix Write WriteMatrix	Migrating the access on single simulator variables on page 73

Block of Platform Access Library	Refer To
Capture InitCapture ConfigureCapture StartCapture FetchCapture GetCaptureStatus	Migrating the capturing of data on page 74
StreamToDisk	Migrating the streaming of captured data to the disk on page 78
AddCapturetoReport	Migrating the reporting of captured data on page 78

Migrating Exec blocks that use the rtplib2 module Replace the Exec blocks that use the **rtplib2** module with the blocks of the XIL API Convenience library that provide the same functionality.

If you need more functional granularity, use the blocks of the XIL API library.

For information on accessing library elements via an Exec block, refer to [XIL API Convenience \(Model Access\)](#) ([AutomationDesk Accessing Simulation Platforms](#) ⓘ) and [XIL API \(Model Access\)](#) ([AutomationDesk Accessing Simulation Platforms](#) ⓘ).

Note

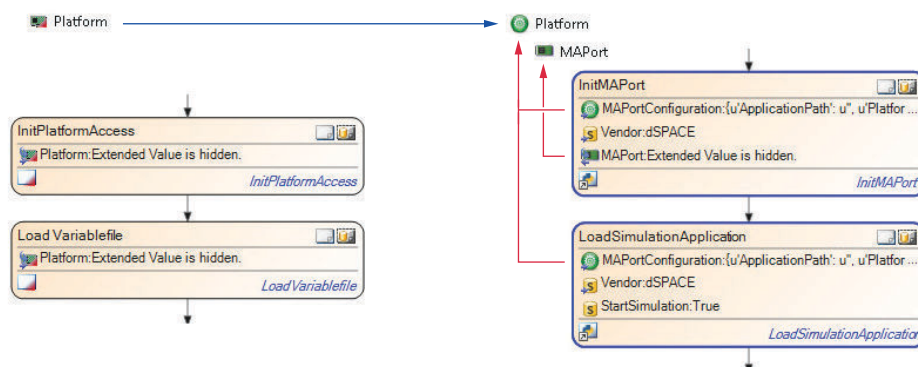
Do not use the *Common Language Runtime* (clr) module of the *Python for .NET* package in Exec blocks to access AutomationDesk's **xilapi** module, because the clr module is already internally used. For more information on migrating the **rtplib2** accesses, refer to the *rtplib2 to XIL API .NET MAPort Migration Guide* in <http://www.dspace.com/go/pscta>.

Migrating the platform access initialization

Migrating InitPlatformAccess and LoadVariablefile blocks For connecting to the platform and starting the simulation application, the XIL API Convenience library provides the **InitMAPort** and **LoadSimulationApplication** blocks.

You can use the automatically migrated **MAPortConfiguration** data object to specify the platform and the simulation application.

You must add the XIL API library-specific MAPort data object manually. It is used by other blocks to reference the configured platform access.



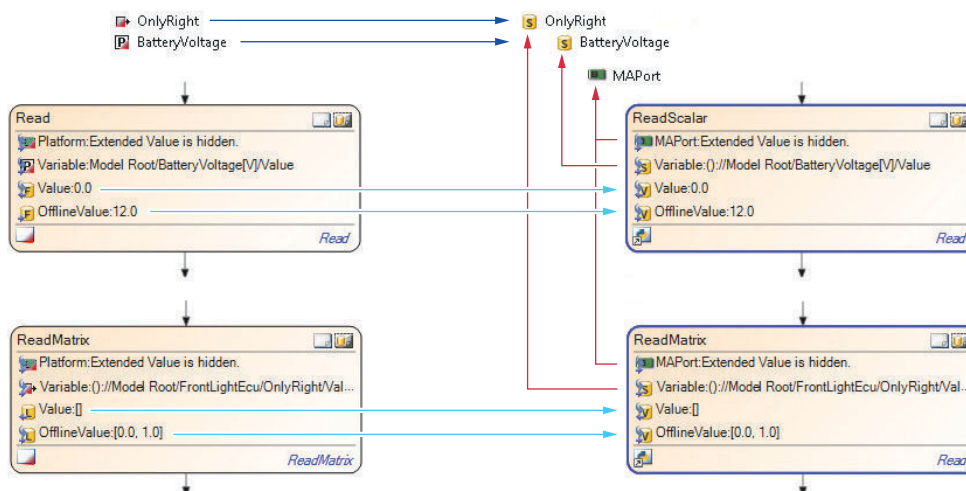
Migrating Disconnect blocks After the tasks that are performed on the platform are finished, the used resources are released. For this, the XIL API Convenience library provides the ReleaseMAPort block.



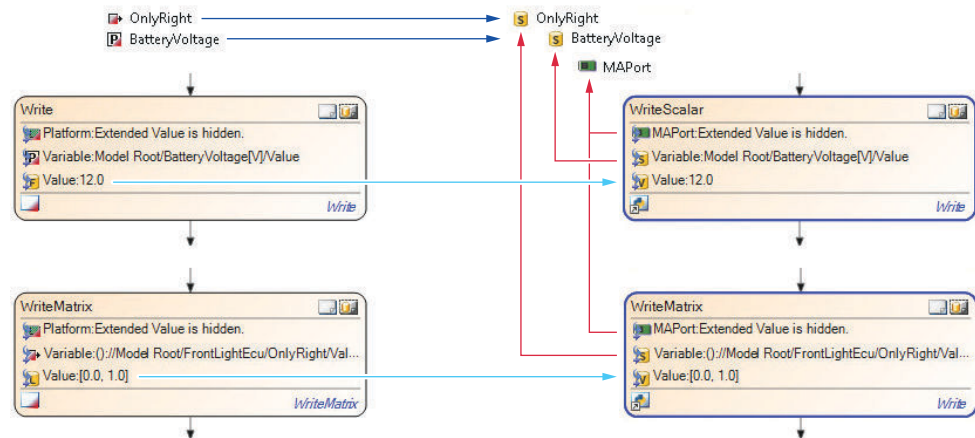
Migrating the access on single simulator variables

To specify the variables to be read or written, you can use the String data objects that are generated automatically from the former Variable data objects.

Migrating Read and ReadMatrix blocks For reading simulator variables, both scalar and matrix, the XIL API Convenience library provides the Read block, independently of the data type to be read. The data type is determined by the type of the data object that is referenced by the block's Value Variant data object.



Migrating Write and WriteMatrix blocks For writing simulator variables, both scalar and matrix, the XIL API Convenience library provides the **Write** block, independently of the data type to be written. The data type is determined by the type of the data object that is referenced by the block's Value Variant data object.



Migrating the capturing of data

Unlike the Platform Access library, where the mapping of a variable's name to its model path is stored separately in each **Variable** data object, the XIL API Convenience library introduces a variable pool that holds the mapping information for all relevant variables of a project. A variable pool can be implemented as a data container or as a **Mapping** data object. You must add the **Mapping** data object manually at the highest project level.

For a fast migration result, you can gather the strings that are generated automatically from the former **Variable** data objects in a data container to implement the variable pool.

Tip

A convenient way to copy the contents of the data container to the **Mapping** data object is to export the data container to an XML file via **Export XIL API Mapping** and then import this XML file via **Import XIL API Mapping** in the context menu of the **Mapping** data object.

Unlike in the Platform Access library, where the variables to be captured are added to the blocks as dynamic **Variable** data objects, in the XIL API Convenience library the variables to be captured are specified via a **List** data object that contains the variable names.

Tip

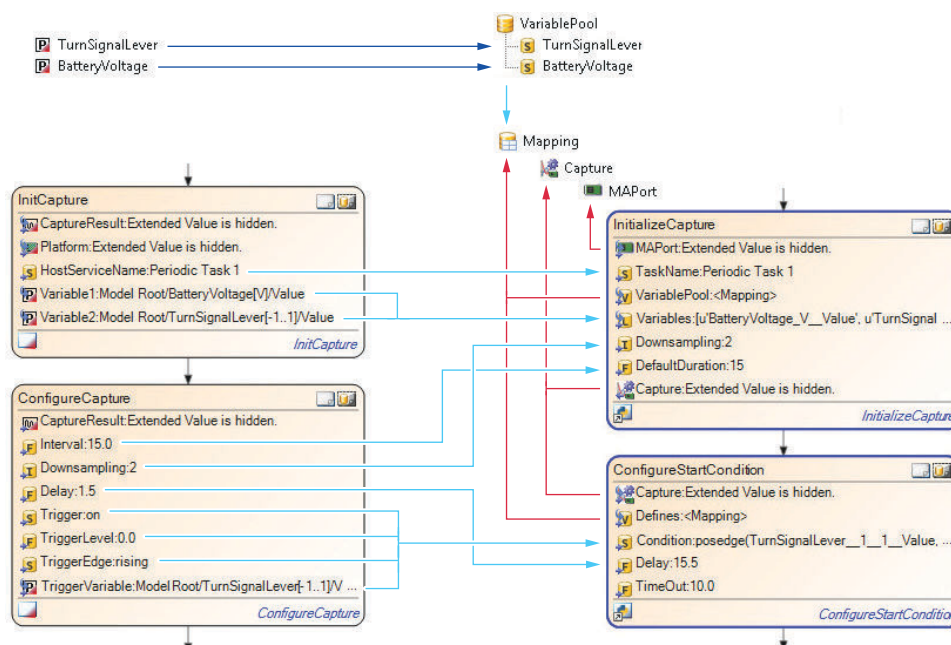
A convenient way to add a variable name to a **List** data object is to drag it from the **Mapping Viewer**.

You must add the XIL API library-specific Capture data object manually. It is used to reference the configured capture.

Migrating InitCapture and ConfigureCapture blocks For configuring which variables are captured, the XIL API Convenience library provides the InitializeCapture block.

If a trigger to start capturing is configured, you also need a ConfigureStartCondition block for migration. You must translate the block's trigger parameterization into a Condition string in ASAM General Expression Syntax (GES). For more information, refer to Edit (Condition String in ASAM GES).

The following illustration shows the migration of extended capturing blocks:



Migrating StartCapture, FetchCaptureData and GetCaptureStatus blocks For starting the capture, the XIL API Convenience library provides the StartCapture block.

For getting the status of a specified capture, the XIL API Convenience library provides the GetCaptureState block. This lets you specify when to get the captured data via a GetCaptureResult block.

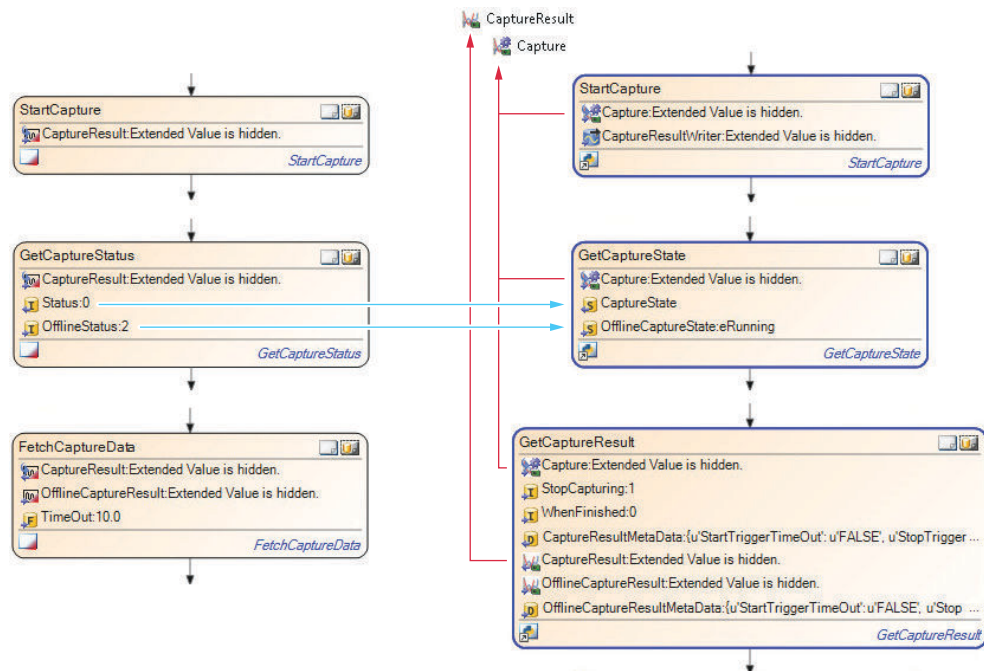
The Status Integer data object correlates to the CaptureState String data object as shown in the following table.

Status	CaptureState
0	eCONFIGURED
1	eACTIVATED
2	eRUNNING
3	eFINISHED

Status	CaptureState
4	— ¹⁾
5	eACTIVATED

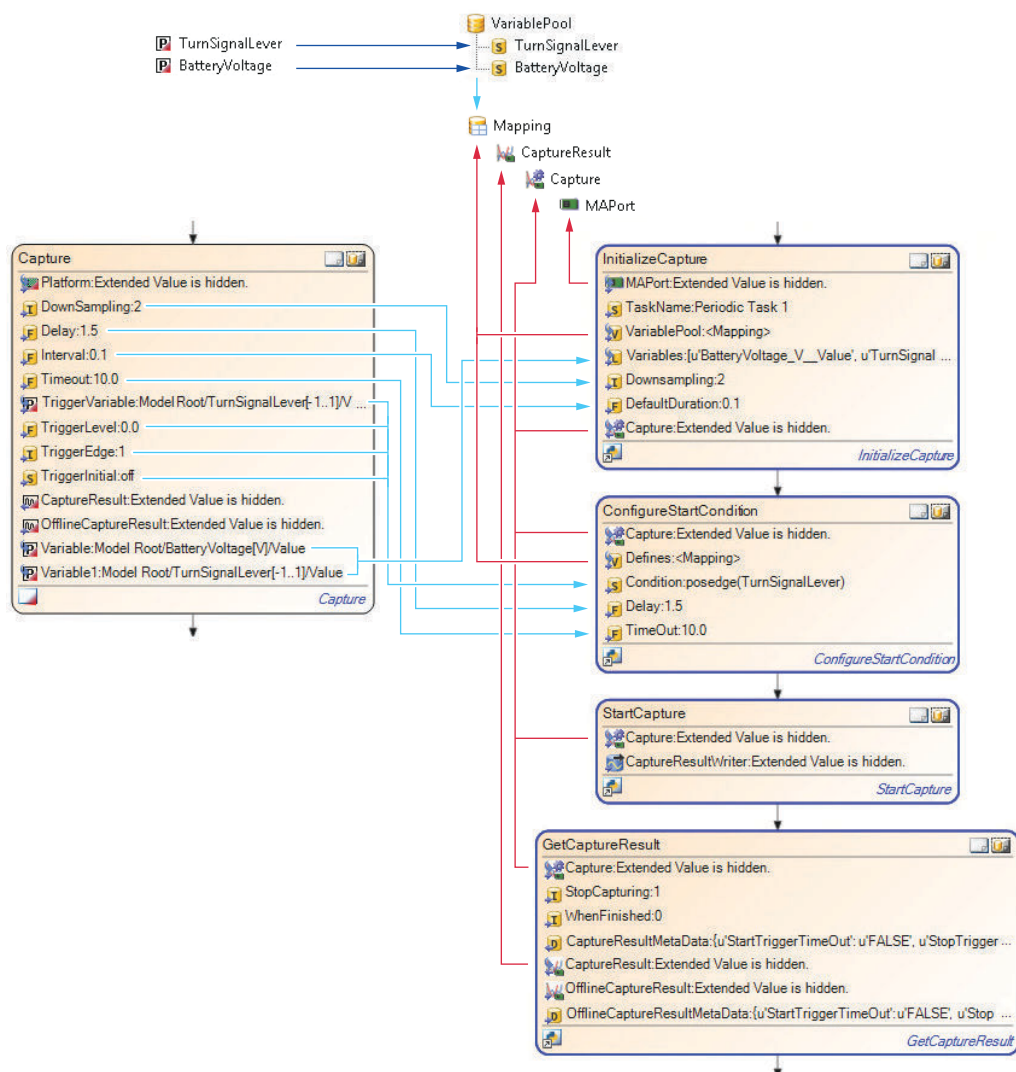
¹⁾ When you use the XIL API, an exception occurs if data capturing is aborted.

You must add the XIL API library-specific **CaptureResult** data object manually. It is used to reference the captured data, as shown in the following illustration:



Migrating Capture blocks A Capture block implements a complete capturing process. To migrate it, you need the same XIL API Convenience blocks as for migrating blocks of the Extended Capturing library folder. To wait until the capture is in **eFINISHED** state, set the **WhenFinished** parameter of the **GetCaptureResult** block to **1**.

The following illustration shows the migration of a Capture block with a configured trigger:



If no trigger is activated via the TriggerInitial data object, the ConfigureStartCondition block is not needed.

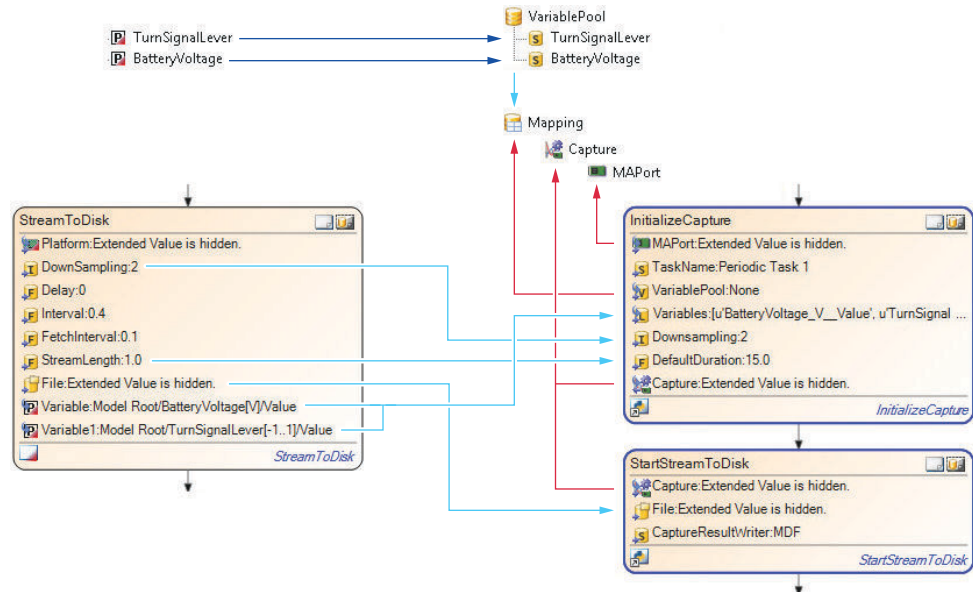
Tip

Using the XIL API Convenience library provides the following additional benefits:

- You can stop capturing explicitly via a StopCapture block or by a conditional trigger via a ConfigureStopCondition block.
- You can specify more complex condition terms that utilize more than one variable for triggering.

Migrating the streaming of captured data to the disk

Migrating StreamToDisk blocks For writing the captured data immediately to a file on the disk, the XIL API Convenience library provides the StartStreamToDisk block, as shown in the following illustration:



Unlike the Platform Access library, which uses the IDF format to store captured data in files, the XIL API Convenience library uses the MDF format.

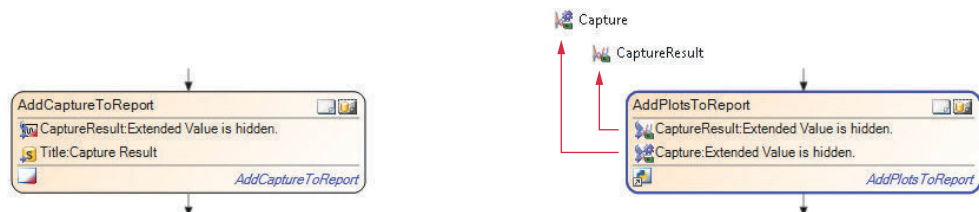
Tip

Using the XIL API Convenience library provides the following additional benefits:

- You can read in MDF and IDF files from the disk to a **CaptureResult** data object via a **GetRecordedData** block.
- You can specify triggers for streamed capturing via a **ConfigureStartCondition**, **ConfigureStopCondition**, or a **ConfigureStopDuration** block.

Migrating the reporting of captured data

Migrating AddCaptureToReport blocks For reporting the captured data, the XIL API Convenience library provides the **AddPlotsToReport** block, as shown in the following illustration:



Tip

Using the XIL API Convenience library provides the additional benefit that you can group the display of multiple variables in single diagrams via an AddCustomPlotsToReport block.

Related topics**References**

[Export XIL API Mapping \(AutomationDesk Accessing Simulation Platforms !\[\]\(83f22ed94ec5517769dd76d702c6bfd8_img.jpg\)\)](#)
[Import XIL API Mapping \(AutomationDesk Accessing Simulation Platforms !\[\]\(58518edde73d42d67a35a8ed26134c7b_img.jpg\)\)](#)
[XIL API \(Model Access\) \(AutomationDesk Accessing Simulation Platforms !\[\]\(256548e00e7fa4879dddf376cbbab973_img.jpg\)\)](#)
[XIL API Convenience \(Model Access\) \(AutomationDesk Accessing Simulation Platforms !\[\]\(4df0d38cb8d3da3858f8bf3819cfafd3_img.jpg\)\)](#)

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.....	82
A.....	82
B.....	84
C.....	84
D.....	85
E.....	86
F.....	87
H.....	87
I.....	87
L.....	88
M.....	89
O.....	90
P.....	90
R.....	91
S.....	92
T.....	94
U.....	94
V.....	94

W.....	95
X.....	95

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 Browser](#).

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 Browser](#).

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 Browser](#).

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.

D

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 Browser](#).

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 Browser](#).

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 Browser](#). Each library is organized as a tree and can be structured using [library folders](#).

There are [built-in libraries](#) and [custom libraries](#).

Library Browser 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 Browser](#), 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 Browser](#) 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](#) or 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 Browser](#) 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 Browser](#).

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 Browser](#) 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.

W

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

X

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

- AutomationDesk
 - basic concepts 12
 - Data Object Editor 25
 - Data Object Selector 27
 - demo projects 43
 - libraries 35
 - Library Browser 23
 - Library Favorites 24
 - Message Viewer 31
 - new features 51
 - Output Viewer 31
 - Platform Manager 30
 - Sequence Builder 23
 - Sequence Overview 24
 - symbols 32
 - user interface 19
- AutomationDesk GUI 19
- AutomationDesk library 12
 - ControlDesk Access 38
 - custom 40
 - Dialogs 36
 - Evaluation 36
 - Framework Builder 37
 - Main 36
 - MATLAB Access 39
 - ModelDesk Access 39
 - MotionDesk Access 39
 - Platform Management 37
 - Real-Time Testing 39
 - Remote Calibration (COM) 38
 - Remote Diagnostics (COM) 38
 - Report 36
 - RS232 38
 - Signal-Based Testing 40
 - Test Builder 37
 - XIL API 37
 - XIL API Convenience 37
- AutomationDesk project 12
- AutomationDesk sequence 12

B

- Breakpoints Viewer 32

C

- Common Program Data folder 6, 85
- ControlDesk Access library 38
- custom library 40

D

- Data Object Editor 25
- Data Object Selector 27
- demons for AutomationDesk 43
- Dialogs library 36
- Documents folder 6, 86

E

- Evaluation library 36

F

- Framework Builder library 37

L

- libraries
 - AutomationDesk 35
 - ControlDesk Access 38
 - custom 40
 - Dialogs 36
 - Evaluation 36
 - Framework Builder 37
 - main 36
 - MATLAB Access 39
 - ModelDesk Access 39
 - MotionDesk Access 39
 - Platform Management 37
 - Real-Time Testing 39
 - Remote Calibration (COM) 38
 - Remote Diagnostics (COM) 38
 - Report 36
 - RS232 38
 - Signal-Based Testing 40
 - Test Builder 37
 - XIL API 37
 - XIL API Convenience 37
- Library Browser 23
- Library Favorites 24
- Local Program Data folder 6, 88
- Log Viewer 31

M

- main library 36
- Mapping Editor 29
- Mapping Viewer 28
- MATLAB Access library 39
- migrating
 - AutomationDesk projects and custom libraries 58
 - discontinued library 66
 - library-specific aspects 60
 - Platform Access library 71
 - required preconditions 58
 - XIL API library 60
- ModelDesk Access library 39
- MotionDesk Access library 39

N

- new features
 - AutomationDesk 51

O

- Output Viewer 31

P

- Platform Management library 37
- Platform Manager 30
- Properties 22

R

- Real-Time Testing library 39
- Remote Calibration (COM) library 38
- Remote Diagnostics (COM) library 38
- Report library 36
- RS232 library 38

S

- Sequence Builder 23
- Sequence Overview 24
- Signal Selector 29
- Signal-Based Testing library 40
- symbols
 - AutomationDesk 32

T

- Test Builder library 37

U

- user interface
 - AutomationDesk 19

V

- Variables pane 24

X

- XIL API Convenience library 37
- XIL API library 37

