AutomationDesk

Tutorial

Release 2021-A – May 2021

dSPACE

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

Contents

About This Document

Contents

This tutorial gives you basic information before you start working with AutomationDesk. It gives you step-by-step instructions for implementing automation tasks.

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
DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.
Tip	Indicates tips that can make your work easier.

Symbol	Description
	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files

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

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

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

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

To access the Web version, you must have a *mydSPACE* account.

PDF files You can access PDF files via the  icon in dSPACE Help. The PDF opens on the first page.

Introduction to AutomationDesk

Introduction

AutomationDesk is a software product for organizing and creating automation tasks. It provides graphical editors for describing control-flow-based and signal-based automation tasks. A wide range of automation functions is offered by extendable libraries. The project management features make it easy to handle a large number of automation tasks. You can see all automation sequences, data objects, execution results and formatted reports at a glance.

Limitations

For information on restrictions while working with AutomationDesk, refer to General Limitations (AutomationDesk Basic Practices).

Where to go from here

Information in this section

The following lessons assume that you are familiar with the following basics:

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.

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.

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.

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.

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.

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.

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.

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.

How to Start AutomationDesk

Precondition

AutomationDesk is installed.

Method

To start AutomationDesk

1 Suppose that <x.y> is the version of the installed dSPACE AutomationDesk. On the **Start** menu, choose **All Programs - dSPACE AutomationDesk <x.y>**, and click **dSPACE AutomationDesk <x.y>**.

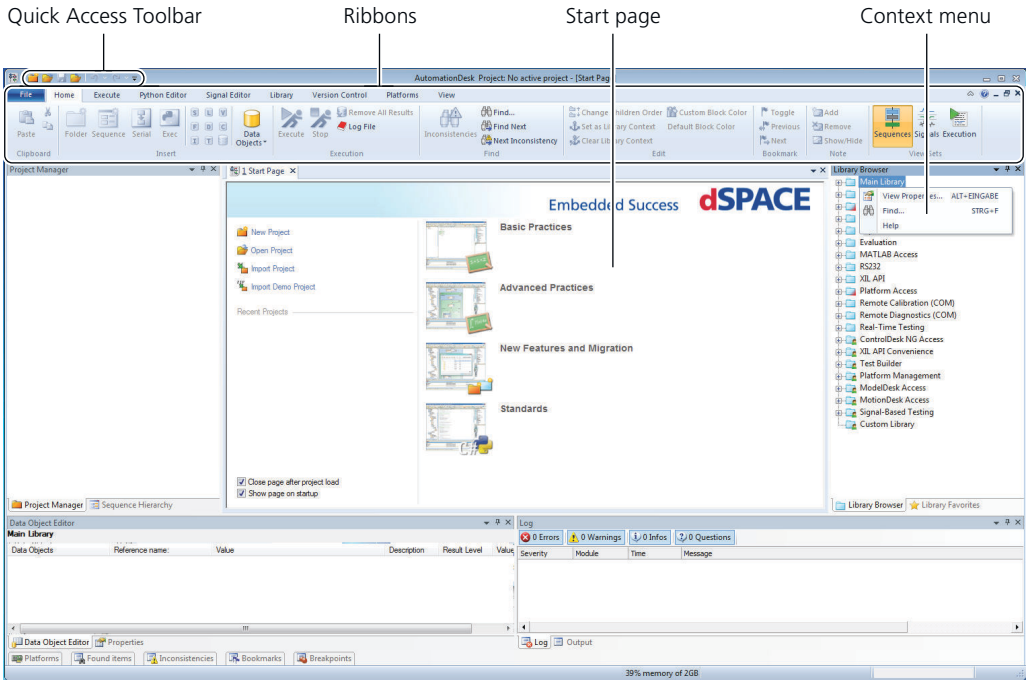
AutomationDesk opens.

Result

You started AutomationDesk.

You can invoke AutomationDesk's commands using:

- The Quick Access Toolbar
- The provided ribbons
- The AutomationDesk Start Page
- The context menus of the displayed panes and of the elements they contain



Tip

The installation process automatically creates a shortcut on the desktop. You can double-click to start AutomationDesk.

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. Some panes are arranged into view sets that match a task you want to perform.

Independently of the panes, some AutomationDesk components open object-specific pages in the working area at the center of the screen.

For some object types, AutomationDesk provides specific edit dialogs.

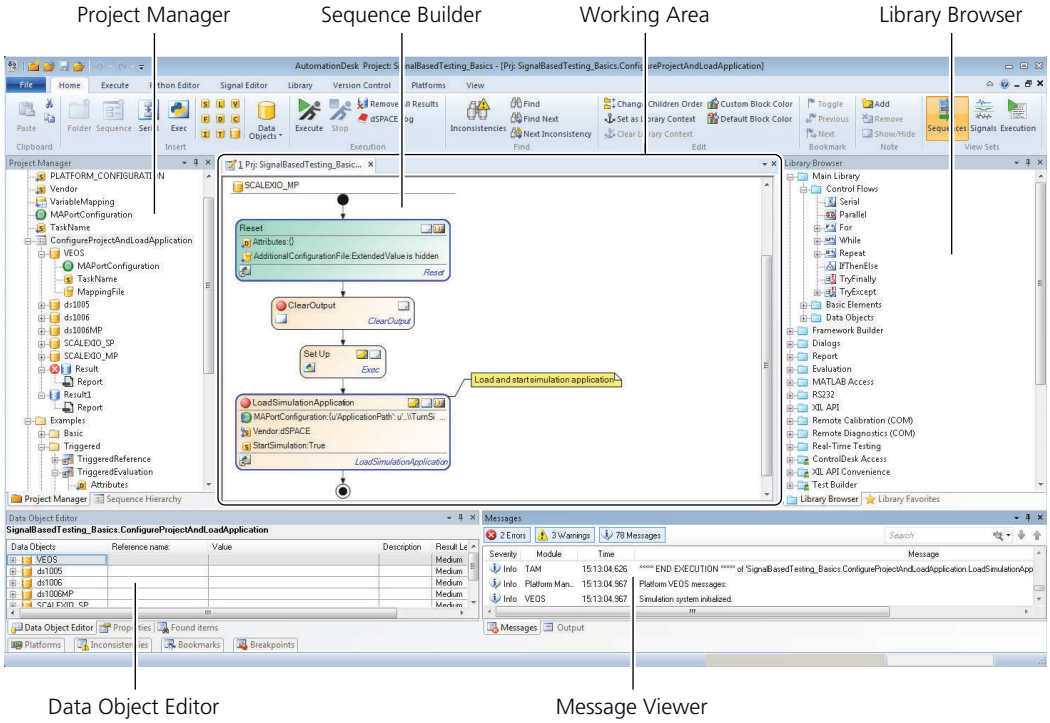
For general information on handling the user interface and detailed instructions on customizing the screen layout, refer to Customizing AutomationDesk's User Interface (AutomationDesk Basic Practices).

Overview of the default view sets

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

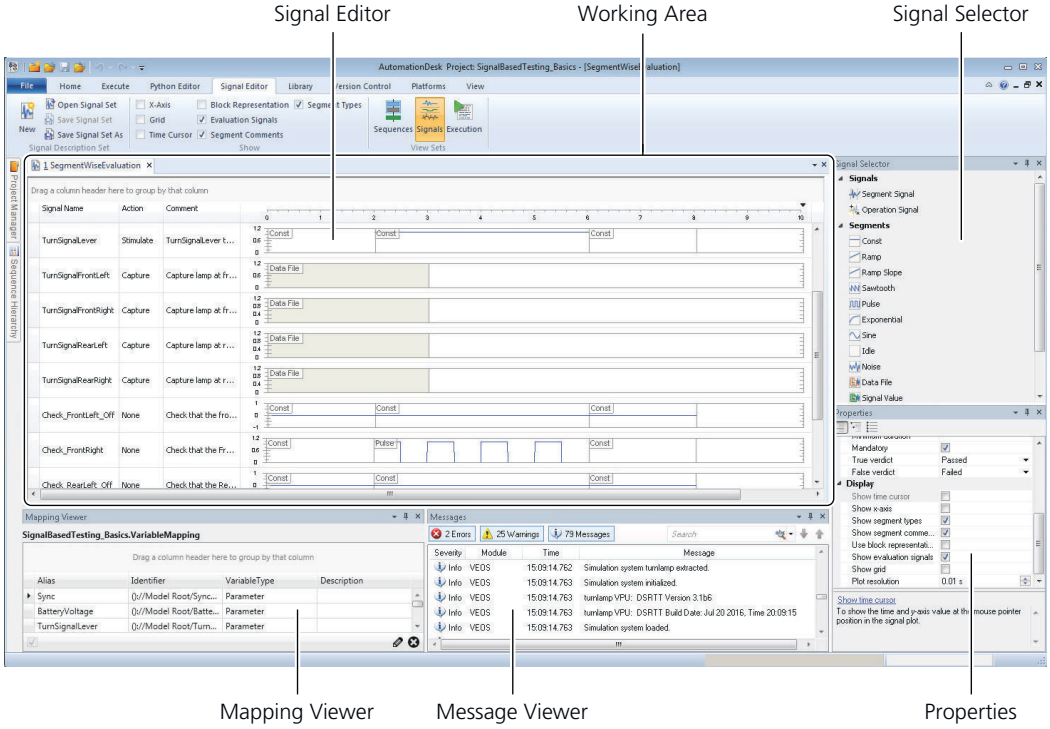
Sequences view set The Sequences view set is suitable for implementing and configuring automation tasks.

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.



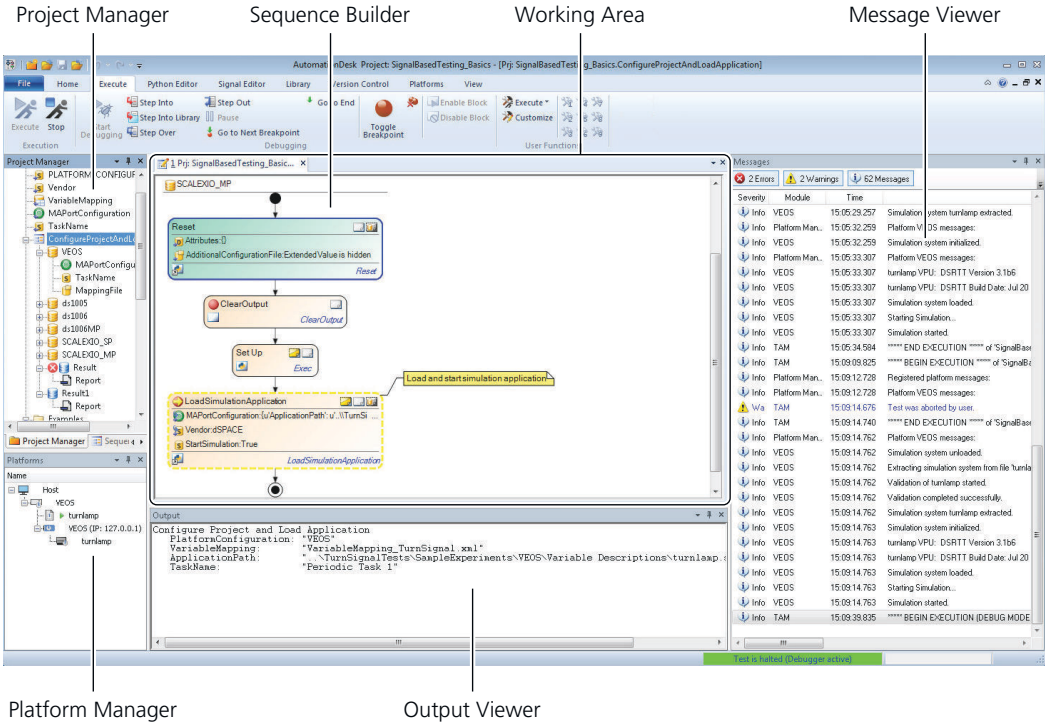
Signals view set The **Signals** view set is suitable for configuring signal description sets.

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 is used to manage automation projects. You can:

- Structure large projects by using folder elements
- Create new automation sequences

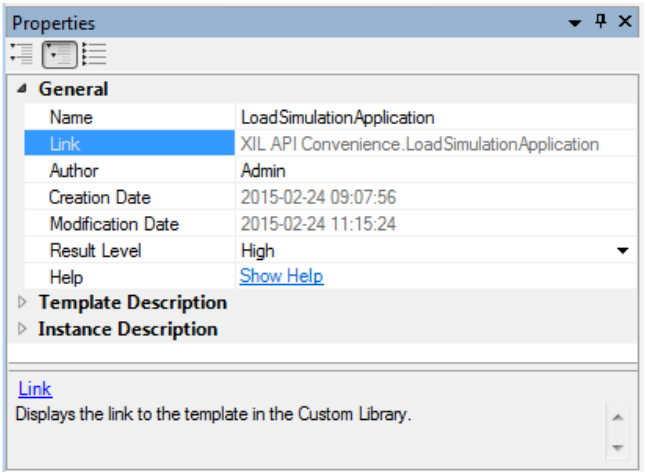
  A sequence contains an executable program flow.
- Create project-specific data objects

  The project-specific data objects are created in the Project Manager and can be referenced by other data objects, for example, by a local data object of an automation block.
- Execute automation sequences, starting from different hierarchy levels
- Change the execution order of the automation sequences
- Edit the properties of projects, folders and sequences
- View the results
- Generate reports in HTML or PDF format

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.

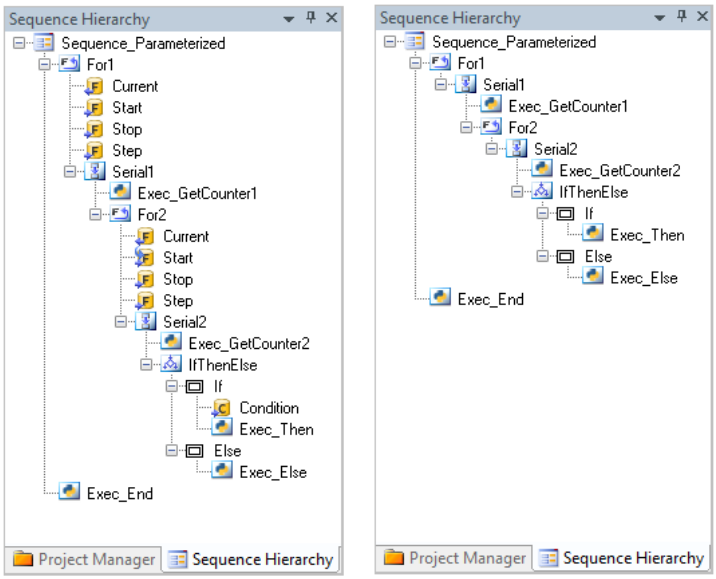
Sequence Builder The Sequence Builder is used to build an automation sequence. You can:

- Create automation sequences using the drag & drop mechanism
- Edit properties of an automation block
- Parameterize data objects of an automation block

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 The Sequence Hierarchy Browser is used to display the structure of the sequence that is open in the Sequence Builder. 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. The left of the following illustrations shows a sequence with displayed data objects and the right illustration shows the same sequence with hidden data objects.



You can use the Sequence Hierarchy Browser to do tasks such as:

- Expand and collapse the sequence tree by clicking the tree nodes.

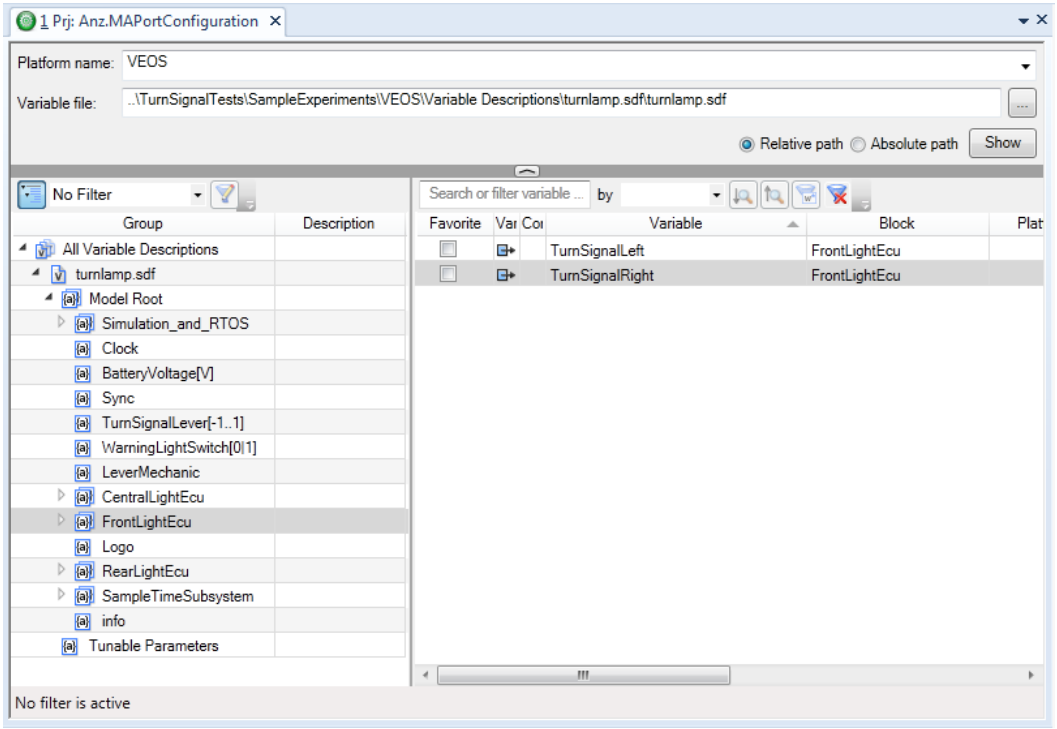
  The automation blocks in the Sequence Builder are collapsed or expanded simultaneously.
- Rename, move, and delete automation blocks.
- Create custom library elements by dragging a sequence element to the custom library folder of the Library Browser.

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.



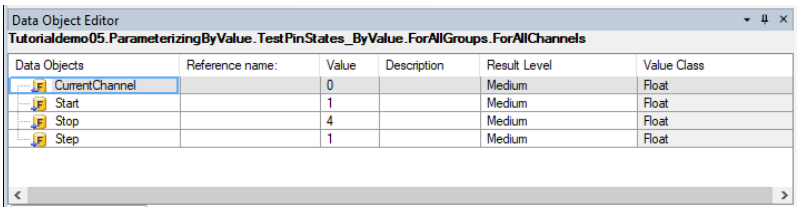
Panes for parameterizing projects and sequences

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

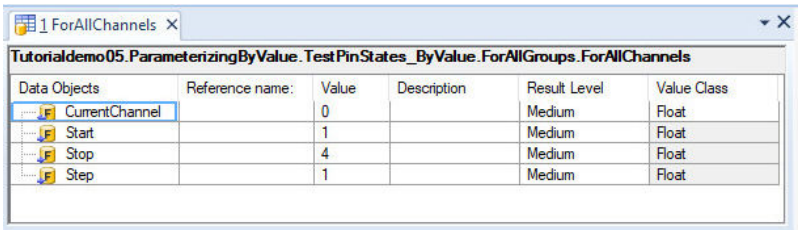
  In single view mode, the Data Object Editor is displayed on the Data Object Editor page of the Tool Window. It is activated by the **`View Data Objects`** command or by clicking the page. The content of the Data Object Editor is automatically synchronized with the currently selected project element or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

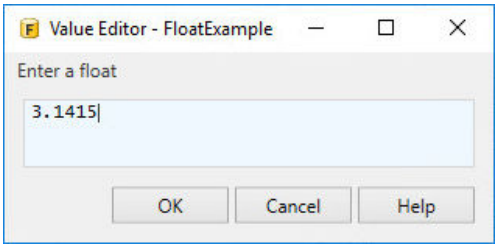


You can switch the display mode of the Data Object Editor on the General page of the AutomationDesk Options dialog. For more information, refer to Using the Data Object Editor (AutomationDesk Basic Practices 🕮).

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:

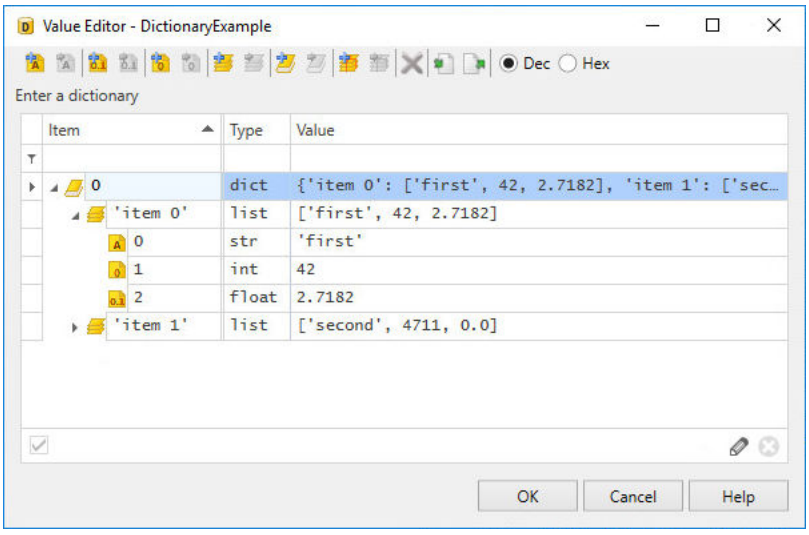
- Integer, float, string

  The Value Editor provides one edit field:



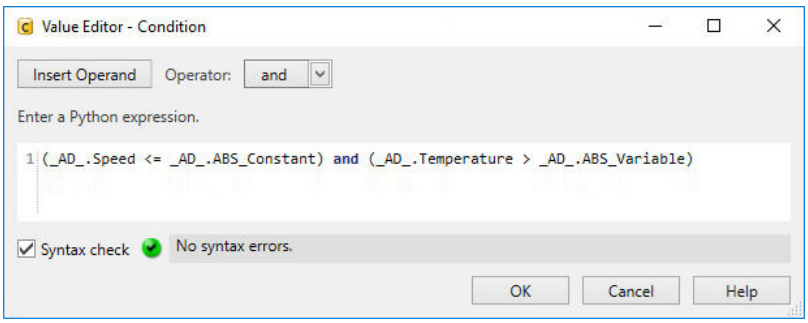
- Tuple, list, dictionary

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

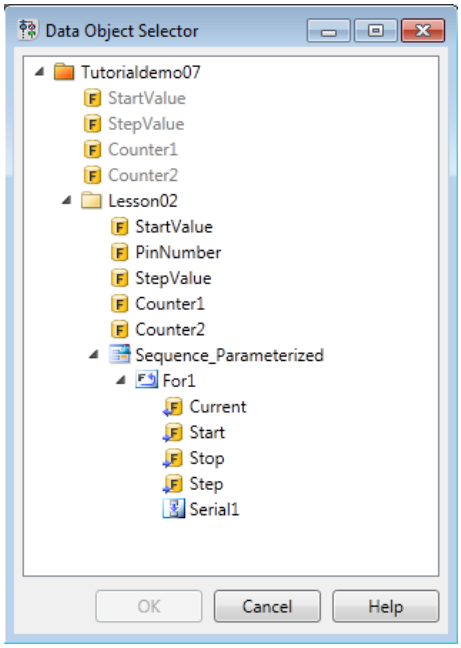


- Condition, Python expression

  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.



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.

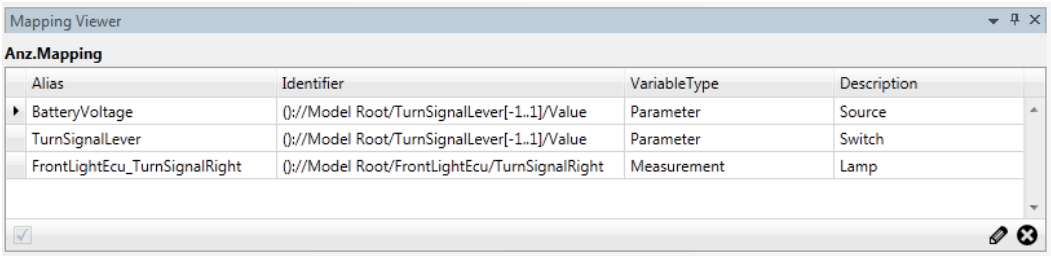


Furthermore, you can use the **Data Object Selector** to specify references to data objects in custom libraries and in projects by selection. This avoids typing errors. To provide the additional tree structure in the **Data Object Selector**, you can set the library or project as the current library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



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:

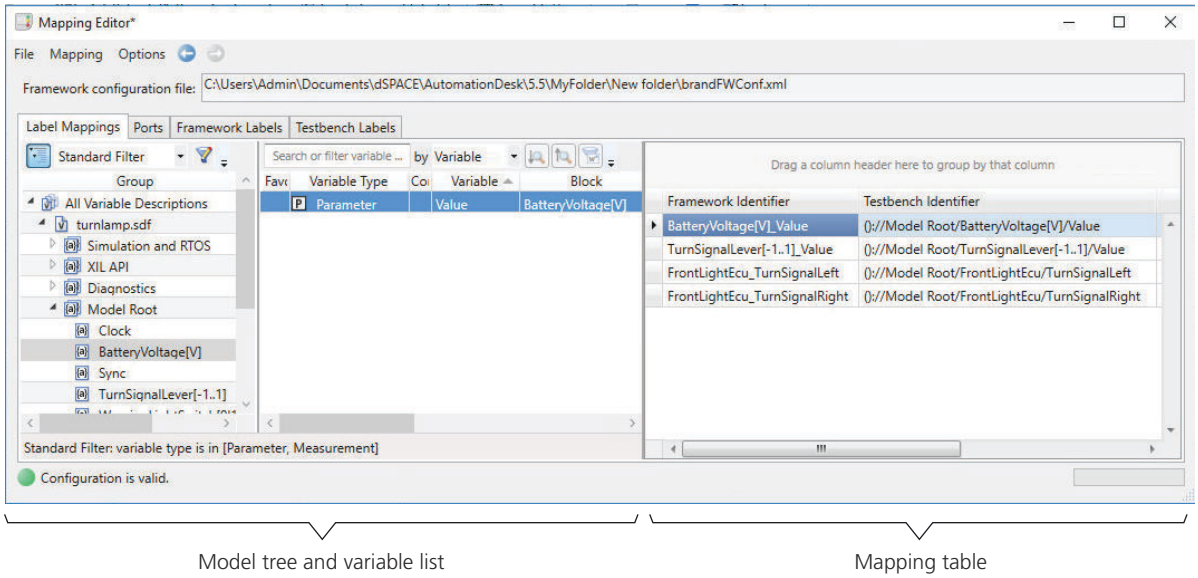
Signal Editor The Signal Editor is used to specify the shape of signals for stimulation and the conditions to evaluate captured signals. You can do the following:

- Add signals and signal segments to a signal description set via drag & drop
- Edit the properties of the signals and segments

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.

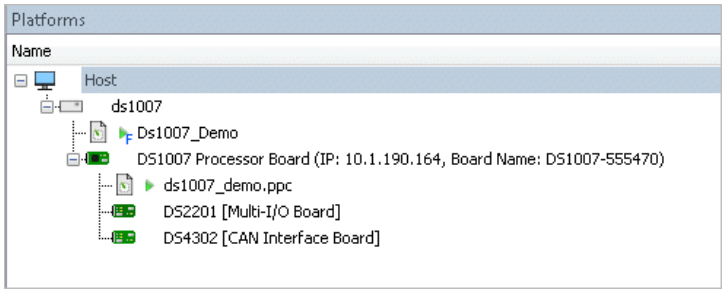
The configuration information is organized in pages. For example, the **Label Mappings** page contains the variable mapping. The available variable paths are provided by the model tree and the variable list. The mapping of paths to variable aliases is contained in the mapping table.



Panes for executing and debugging projects and sequences

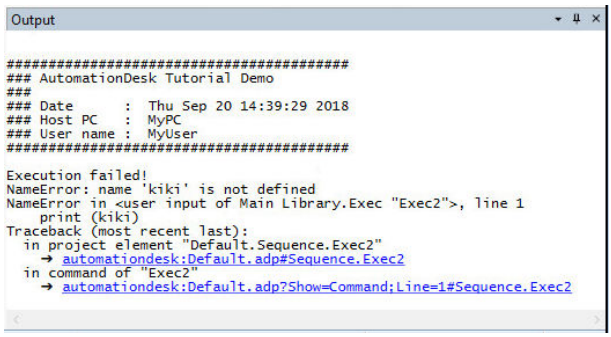
The following panes are used to execute or debug projects, sequences, and blocks:

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.

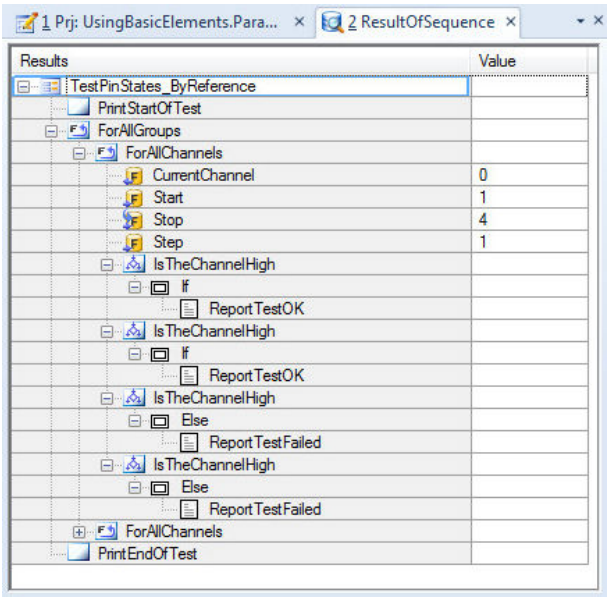
Output Viewer In the Output Viewer, you can see the specified output from print commands and error messages of the Python interpreter. It can contain colored text and hyperlinks. This can be used for debugging purposes, for example.



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.



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.

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.

Symbol	Element	Description
	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	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.: ▪ In the Project Manager: Projects and sequences. ▪ In the Library Manger: Custom libraries, sequence templates, block templates and Python modules.
	Modified subelement	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.

Symbol	State	Description
		
	Linked element	Indicates that the sequence is an instance of a custom library template. The link mode is also displayed: ▪ Dynamic link mode ▪ Static link mode
	Disabled element	Indicates that the element (folder, sequence or automation block) is excluded from execution.
	Library operation mode	Indicates that the library is in online, offline, or online recording mode.
	Discontinued element	Indicates that the element belongs to a discontinued built-in library.

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**	
		The standard symbol of the element is displayed, if: ▪ 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.
		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: ▪ 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.

AutomationDesk's 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 Bowser. 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 🕮) and XIL API Convenience (Electrical Error Simulation) (AutomationDesk Simulating Electrical Errors 🕮).

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.

	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 🕮) .
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 🕮).
ControlDesk Access	The ControlDesk Access library provides library elements to access ControlDesk. This allows you to automate the following features: ▪ Calibration and measurement ▪ Diagnostics For more information, refer to Automation Blocks (AutomationDesk Accessing ControlDesk 🕮).
ModelDesk Access	The ModelDesk Access library provides library elements to access ModelDesk. 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 🕮).
MotionDesk Access	The MotionDesk Access library provides library elements to access MotionDesk. 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 🕮).
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 🕮).

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 🕮).

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 🕮).

Custom Library

The custom library with the name **Custom Library** makes it possible to integrate user-defined automation blocks 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 🕮).

Scope of Project-Specific Data Objects

Introduction

The data object of an automation block can be parameterized in different ways.

Basics

The data object of an automation block can be set by a local value or by a reference. While a directly set value (local value) belongs to only the current automation block, a source data object can be referenced by any automation block that is within the current hierarchy path. By using project-specific data objects for data referencing, you can parameterize your sequences centrally in the Project Manager. Project-specific data objects can also be accessed via the AutomationDesk API.

Data object set by value

You can parameterize the value of an automation block's data object in the **Value** column of the Data Object Editor. This value belongs to only the current automation block.

Data object set by reference

You can parameterize the value of an automation block's data object by specifying a data object name in the **Reference name** column of the Data Object Editor. Then, the value is resolved with the value of the first data object with the specified name within the current object hierarchy path.

The root of the object hierarchy is always the current project. Below that follows the optional folder structure that you can create in the Project Manager. The next lower level contains the interface of the sequence the current automation block resides in. Below the interface, you can add more levels by using the automation blocks that implement the control flow of the sequence, such as a **Serial** block.

AutomationDesk searches for the referenced data object from lower to higher levels until it finds a data object with the specified name.

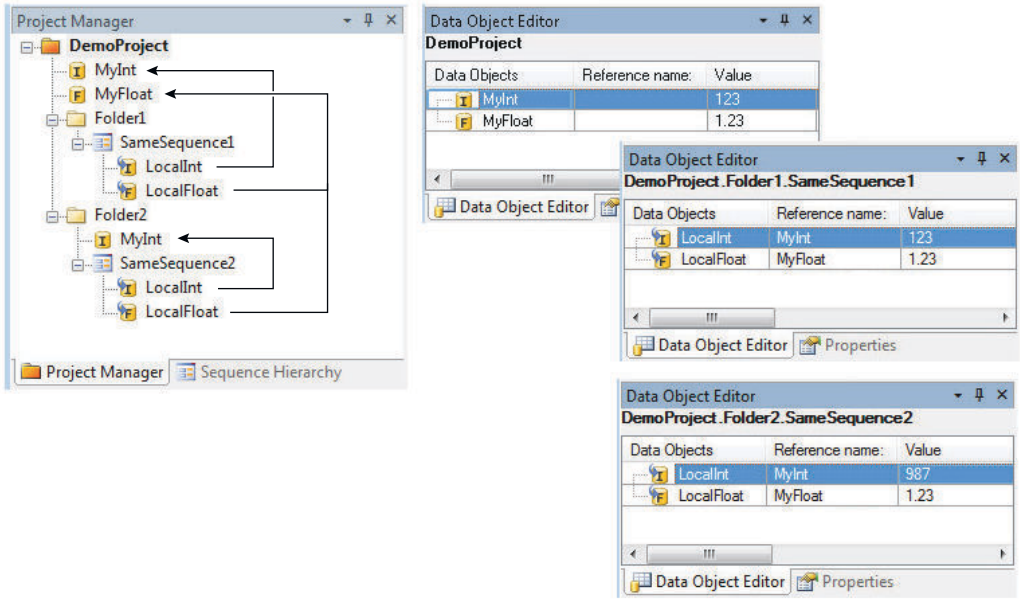
Data objects that are visible in the Project Manager are called project-specific data objects. They can also be accessed via the AutomationDesk API.

You can also use a data object reference for an output data object. The referenced data object will contain the current value of the block's output during execution and can be used by other automation blocks as an input value.

When you work with referenced data, you must know the scope of data objects. This is described in the following example.

Example

You have created a project with two identical sequences that reside in two project folders. The **LocalInt** data object and the **LocalFloat** data object in the sequences' interfaces refer to data objects that are named **MyInt** and **MyFloat** as shown in the following illustration.



When SameSequence1 is executed, AutomationDesk first tries to resolve the reference names MyInt and MyFloat in Folder1. Because this fails, the reference resolution is tried in the next higher level, where it succeeds.

When SameSequence2 is executed, AutomationDesk first tries to resolve the reference names MyInt and MyFloat in Folder2, where it succeeds for MyInt. Because it fails for MyFloat, the reference resolution is tried in the next higher level, where it succeeds.

Automation Tasks Used in This Tutorial

Introduction

This tutorial guides you through working with AutomationDesk. It is divided into several lessons explaining the individual steps involved in using AutomationDesk to create and work with simple automation tasks.

Where to go from here

Information in this section

Overview of the Example Tasks

Introduction

To introduce the example tasks that are used in this tutorial to show the typical application area of AutomationDesk.

Tasks

The following example tasks are used in this tutorial:

- Example Task for Using the Basic Elements of AutomationDesk on page 36

This task shows you how to transfer your automation task into an AutomationDesk project and to transfer your test routine into an AutomationDesk sequence that represents its control flow. No platform access is required.

- Example Task for Using Python in AutomationDesk on page 38.

  This task shows you how to enhance AutomationDesk by using Python code that you implement yourself or that is provided by others. No real-time hardware is required.

- Example Task for Implementing Tests on page 38

  This task shows you how to implement your test routine in AutomationDesk depending on the test strategy: control-flow-based testing or the signal-based testing. To complete this task, access to a platform, i.e., to a real-time hardware or a VEOS, is required.

Prepared Demo

At the end of each lesson you can check the correctness of the executed steps. In the `<DocumentsFolder>\Tutorial Demos` folder you will find an associated demo project, which includes the result you should have achieved during the lesson. To compare your result with the demo project result, you can open the demo project in parallel. Since it has a different name, both projects can be used at the same time without causing conflicts.

A demo project can also be used as a starting point. If you want to work through only a certain lesson, you can open the corresponding demo project, save it under a different name (to keep the original demo project unchanged), and start working through the tutorial using the copy.

Duration

Working through the entire tutorial will take you about 4 hours.

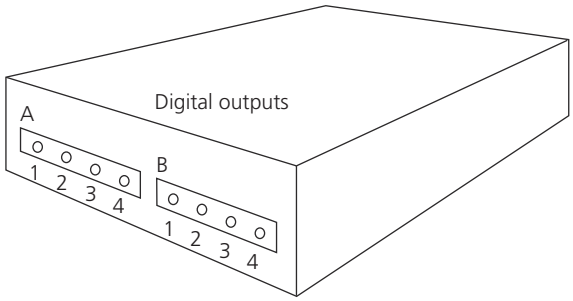
Example Task for Using the Basic Elements of AutomationDesk

Introduction

To learn how to transform your automation task to an AutomationDesk project and your test routine to an AutomationDesk sequence that represents its control flow. No platform access is required.

Your first automation task

Your first automation task is a simple program to read data within a loop and print messages which correspond to a specified condition. Suppose you have an electronic device with 8 digital outputs, arranged in 2 groups of 4 bits each.



In any experiment setup, the digital outputs get some input, and in response to this input they generate a TTL signal (high or low level, equals value 1 or 0). The setpoint of all outputs is 1.

Note

To simplify matters, the output signals will not be read out directly from connected hardware. Instead, the values will be simulated by a 2x4 data array.

The output signals are to be analyzed group by group. A message is to be printed for the bit's value. Finally, when all the signals have been read, an end-of-program message must be output.

As the output signals are to be analyzed in groups, you will use 2 For loops to iterate over the elements of the two-dimensional list of values. The outer For loop will be applied to the groups, the inner For loop will be used to pass through the bits.

Lessons

To fulfill the first task, you have to work through the following lessons:

- Lesson 1: Creating Your First Project on page 41
  You will create a structured project containing an automation sequence.
- Lesson 2: Parameterizing the Automation Sequence on page 57
  You will parameterize the elements of an automation sequence.
- Lesson 3: Executing the Automation Sequence on page 77
  You will execute an automation sequence and generate a report on the execution results.
- Lesson 4: Creating Custom Libraries on page 89 (only necessary for library developers)
  You will save automation tasks as reusable library elements.
- Lesson 5: Working With the Custom Library on page 103
  You will build an automation sequence using custom libraries.

Example Task for Using Python in AutomationDesk

Introduction

To learn how to implement tasks in AutomationDesk by using Python scripts.

Your automation tasks

The following tasks are used as examples in this tutorial:

- As an example for including a Python script that uses only local variables and does not access project-specific data objects, you implement a sequence to output information, such as the start time of the execution, the host name and the user name.
- As an example for including a Python script that accesses project-specific data objects, you implement a sequence that returns the radius of a circle when it is called with a specified circle area.
- As an example for including a Python module in a custom library, you implement a block template that returns the area of a ring when it is called with a specified inner and outer radius.

Lessons

To complete the task, you have to work through the following lesson:

- Lesson 6: Using Python Scripts on page 115

  You will integrate Python scripts in AutomationDesk.

Example Task for Implementing Tests

Introduction

To learn how to implement your test routine according to one of the following test strategies:

- Control-flow-based testing

  You implement your test routine by specifying the routine's control flow. This is the same strategy as used for the previous example tasks.
- Signal-based testing

  You implement your test routine by configuring a standardized test scenario via the involved signals.

Your automation task

This is the first task in this tutorial that accesses your platform. If you use dSPACE real-time hardware, it must be installed and connected to your host PC. If you use VEOS, VEOS must be installed and running on your host or a connected PC.

Your task is to automate the downloading of a simulation application that represents a turn signal control in a car.

When the simulation is started, the battery voltage has to be set. Then the turn lever is switched to the right for a specified amount of time and back again.

During this scenario, the state of the front lights is captured.

A report of the test is to be generated that shows plots of how the involved simulator variables changed during the test.

When you work with the signal-based test strategy, you also automate the evaluation of the captured data. At the end of the test AutomationDesk checks, whether the captured signals comply with specified reference signals. Depending on that, a verdict is generated whether the test is passed or failed.

Lessons

You can complete the task in two alternative ways by working through one of the following lessons:

- Lesson 7: Control-Flow-Based Testing on a Simulation Platform on page 131

  You will implement your test routine in a sequence that represents the routine's control flow and accesses your platform via automation blocks of the **XIL API Convenience** library.
- Lesson 8: Signal-Based Testing on a Simulation Platform on page 159

  You will implement your test as an instance of a sequence template that is provided by the **Signal-Based Testing** library. This sequence implements a standardized test scenario that is configured via the description of the signals which are used for stimulating and capturing the simulator variables during the test. You can also specify references for the captured signals and evaluate the test result.

Lesson 1: Creating Your First Project

Introduction	You learn how to create a structured project containing a simple automation sequence.

Introduction to Lesson 1

Task to be performed	This lesson describes how to perform the example task that introduces basic elements of AutomationDesk. For more information, refer to Example Task for Using the Basic Elements of AutomationDesk on page 36.
Before you begin	To understand the principles of AutomationDesk and the terminology used, you are recommended to read Introduction to AutomationDesk on page 9.
What you will learn	This lesson shows you how to create a structured project containing a simple automation sequence. ▪ You will create a new project. ▪ You will add a folder to the project. ▪ You will create a sequence that represents the automation task. ▪ You will build a control flow in AutomationDesk's graphical editor. ▪ You will learn how to navigate in a sequence. ▪ You will save your project and close it.
Duration	Working through this lesson will take you about 20 minutes.
Summary	To check the correctness of the executed steps, refer to Results of Lesson 1 on page 54.

Starting point

Now you can start with the first step, see below.

Related topics

Basics

Step 1: How to Create a New Project

Introduction

An AutomationDesk project contains all the relevant parts and information that belong to an automation task. These are the project elements:

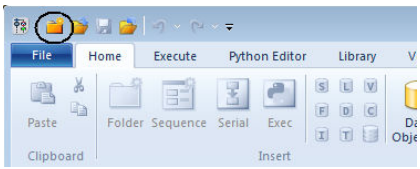
- Automation sequences
- Data objects
- Structure elements
- Execution results
- Reports

All the elements are specified in the AutomationDesk project file (ADPX file) and its corresponding subfolder. AutomationDesk's Project Manager displays the contents of the project file and can be used to add new project elements and work with them.

Method

To create a new project

1 On the Quick Access Toolbar, click New Project to open the New Project dialog.



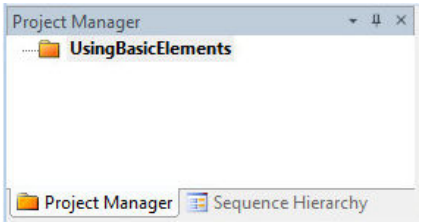
Tip

You can also open the New Project dialog. There are the following ways to do this:

- In the Project Manager, right-click a blank field to open the context menu and select New Project.
- On the AutomationDesk Start Page, select New Project.
- On the File ribbon, select New - New Project.

2 In the New Project dialog, create a new folder on your local disk, such as `<DocumentsFolder>\MyWorkingFolder`. Change to the folder and type `UsingBasicElements.adpx` in the File name edit field, then click Create.

The new project now is displayed as the root element of a project tree in the Project Manager.



What's next

You can structure the project in the next step.

Step 2: How to Structure a Project

Introduction

A project's hierarchical structure is built by adding project folders to the project tree and by renaming them.

In the first five lessons of this tutorial, we will successively implement a folder structure for your project that lets you reconstruct the main learning steps.

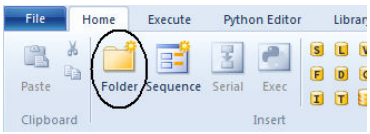
You will now add a folder named *BaseControlFlow* to your project.

Part 1

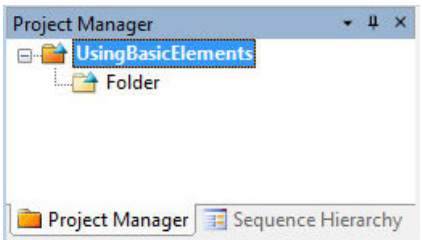
To add a new project folder to a project

1 In the Project Manager, click UsingBasicElements to select your project.

2 On the Home ribbon, click Insert - Folder to create a new folder.



The new project folder is displayed as a child of the project tree's root element.



Continue with next part

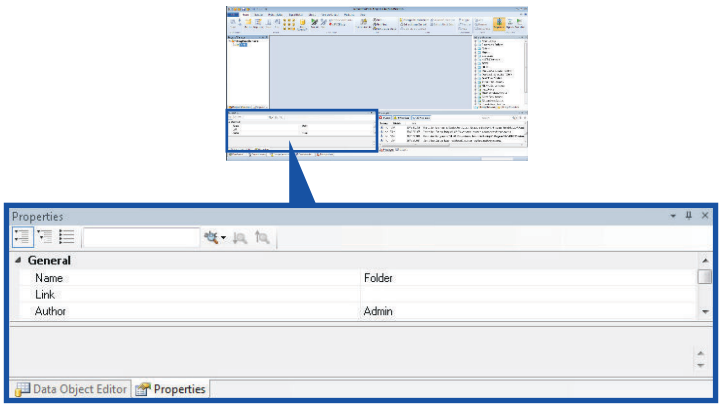
You added a new folder to the project. In the next part, you rename the project folder.

Part 2

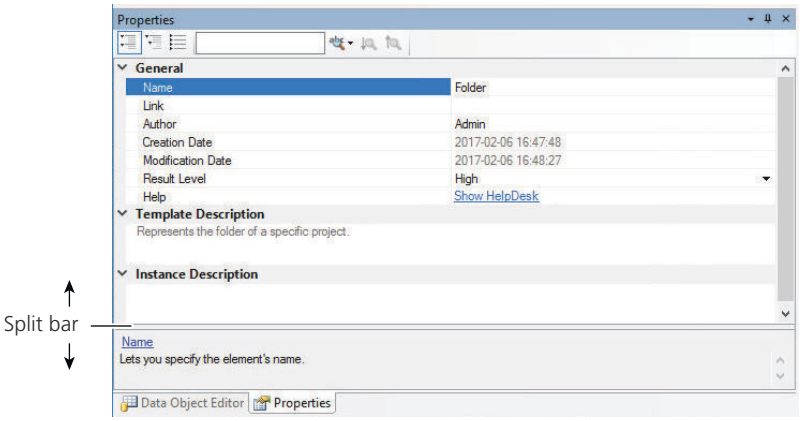
To rename a folder

1 Right-click Folder in the Project Manager to open the context menu of the new folder and choose View Properties.

If not already open, the Properties pane opens to display the properties of the new folder. By default, the Properties pane is located in the bottom left corner of the AutomatinDesk UI.



2 Adjust the size of the pane to display all properties.

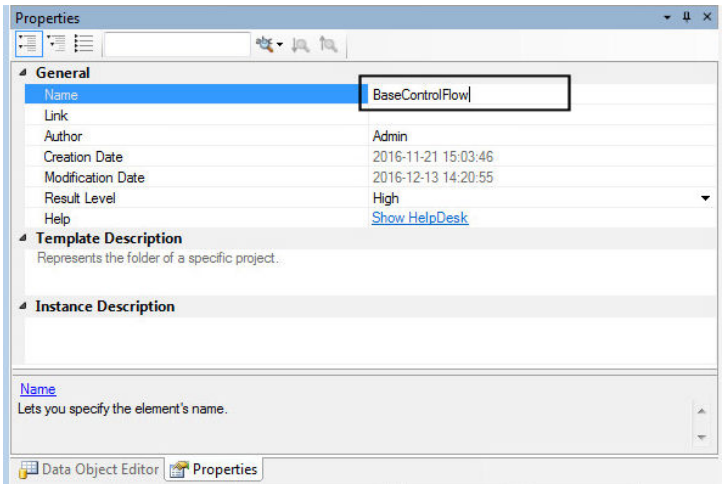


The Properties pane is divided horizontally into two parts. The upper part provides the properties grouped by categories. If you select a specific property here, a short description of the property is displayed in the lower part of the pane.

Tip

Via a split bar, you can adjust an appropriate partitioning of the pane.

3 In the Name edit field, type `BaseControlFlow` to rename the folder.



After you pressed Enter or changed the focus to another element the modified name displays in the Project Manager.

Alternative method

Tip

Alternatively, you can rename a project folder by selecting it in the Project Manager, pressing **F2** and overwriting the existing name.

What's next

You can add a new sequence to the project in the next step.

Step 3: How to Create a New Sequence

Introduction

In this step you will learn how to create a new sequence. The sequence represents your automation task and will later contain the control flow.

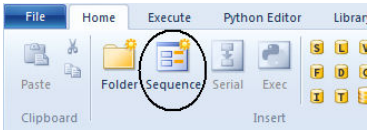
Having defined the *BaseControlFlow* folder for the *UsingBasicElements* project, you can now create a new sequence in it.

Part 1

To create a new sequence

1 In the Project Manager click BaseControlFlow to select your folder.

2 On the Home ribbon, click Insert - Sequence to create a new sequence.



The new sequence appears as a child of the BaseControlFlow folder in the project tree.

Continue with next part

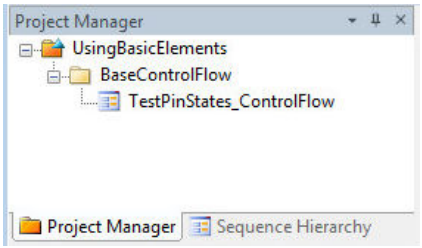
You added a new sequence to the BaseControlFlow folder. You can now rename the sequence. Continue with the next part.

Part 2

To rename a sequence

1. Right-click Sequence in the Project Manager to open the context menu of the new sequence and choose View Properties.
2. In the Properties pane, type `TestPinStates_ControlFlow` in the Name edit field to rename the new sequence.

   After you pressed Enter or changed the focus to another element the modification is displayed in the Project Manager.



Alternative method

Tip

Alternatively, you can rename a selected sequence in the Project Manager directly by pressing **F2** and overwriting the existing name.

What's next

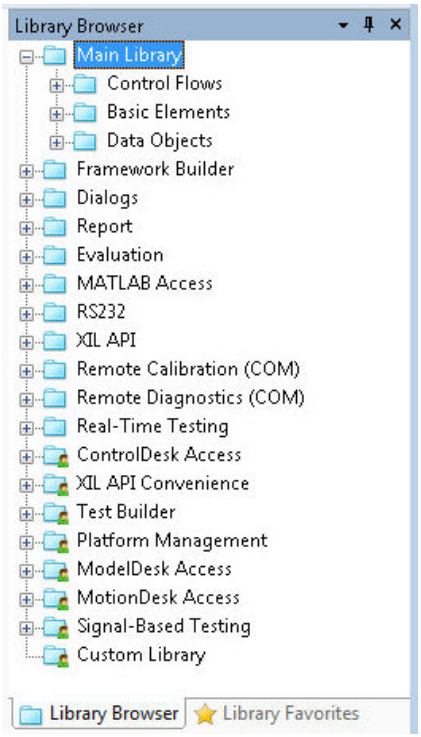
You can now build a control flow in the next step.

Step 4: How to Edit a Sequence

Introduction

This step shows you how to open a newly created sequence and use AutomationDesk's Sequence Builder, which is the graphical editor for building automation sequences.

The typical functions needed to handle automation tasks are provided graphically in the form of automation blocks. Each block is placed in a structured library that can be managed via AutomationDesk's Library Browser. The Library Browser contains all the reusable automation steps stored in both predefined and user-defined libraries. You can find the Library Browser on the right side.



A control flow is created by dragging the required automation elements successively from the Library Browser to the Sequence Builder.

To design the base control flow for your first sequence, reconsider the task to be implemented: Eight digital output channels of an electronic device are to be tested whether they are high or low. The relevant pins are arranged in two groups of four bits each.

To implement this task, the appropriate control-flow consists of two nested *for loops* and a conditional part that depends on the pin state high or low.

You will now open the *TestPinStates_ControlFlow* sequence in the Sequence Builder and build this control-flow in your first automation sequence.

Part 1

To open a sequence

1 Double-click TestPinStates_ControlFlow in the Project Manager.

Tip

Alternatively, you can open the sequence in the Sequence Builder by right-clicking TestPinStates_ControlFlow in the Project Manager and selecting the **`Open Sequence Builder`** command.

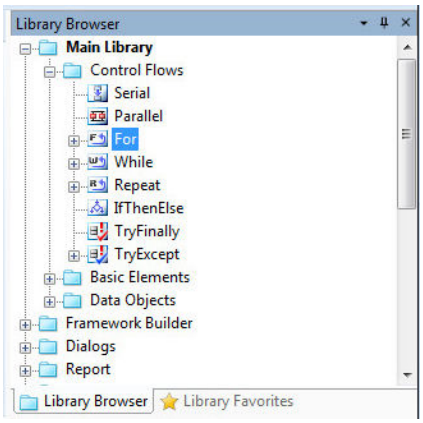
Continue with next part

You opened the TestPinStates_ControlFlow. You can now build the base control-flow in the automation sequence.

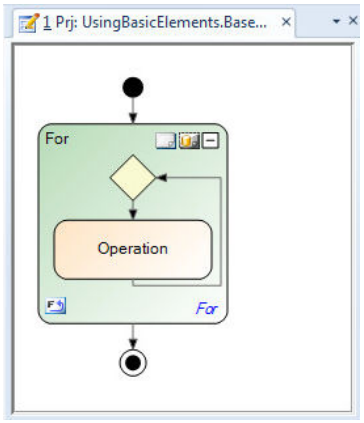
Part 2

To build the base control-flow in the automation sequence

1 In the Library Browser, open the Control Flows folder of the Main Library and select the For automation block.



Drag the For block to the Sequence Builder. A new instance of the For block is displayed in the Sequence Builder.

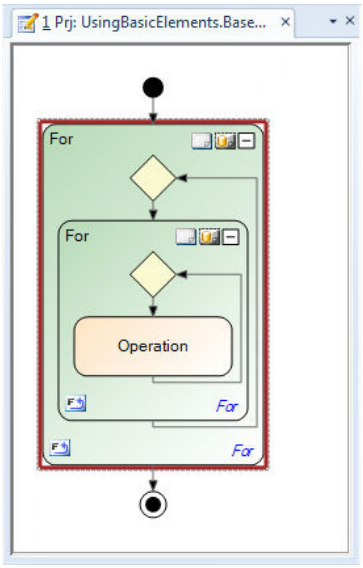


This For block implements the *for loop* for the groups.

Tip

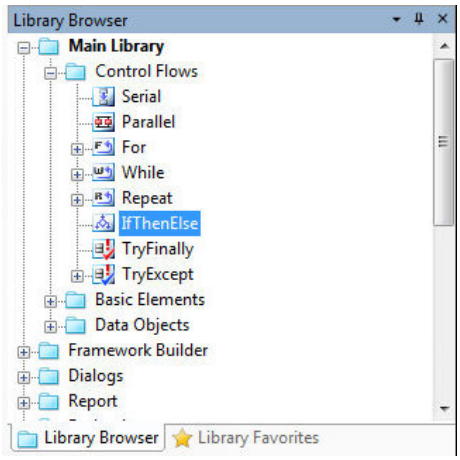
AutomationDesk indicates a valid drop position by a small red marker.

2 In the Library Browser, select the For block from the Control Flows folder and drag it to the Sequence Builder. Drop the block onto the Operation field of the already added For block.



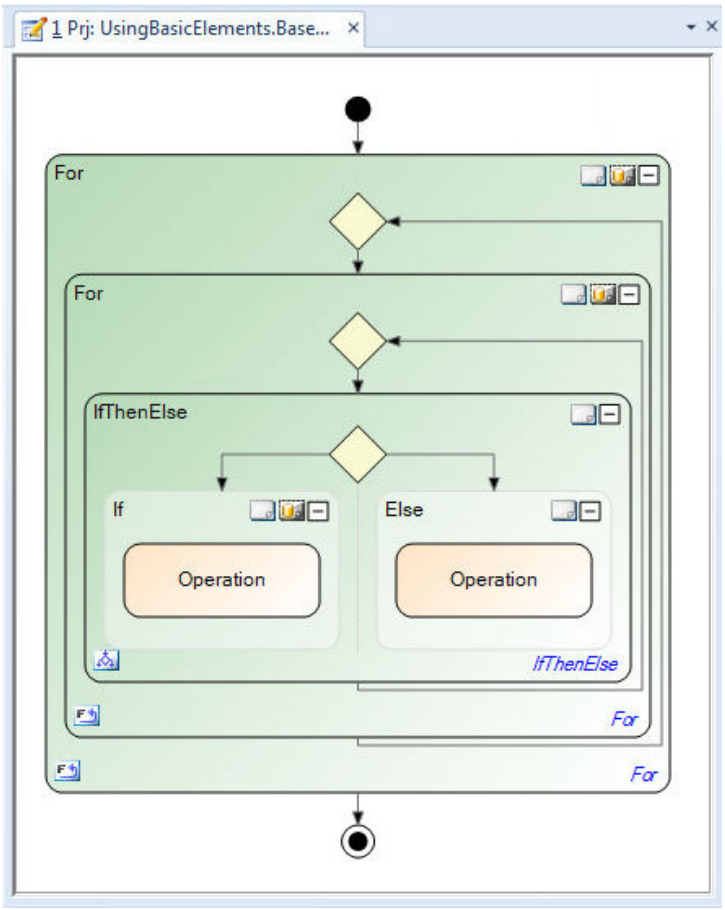
The two For blocks are nested. The new inner For block implements the *for loop* for the pins within a group.

3 In the Library Browser, select the IfThenElse block from the Control Flows folder and drag it to the Sequence Builder.



Drop the block onto the Operation field of the inner For block.

The IfThenElse block allows you to specify a condition that analyzes the value of the digital I/O pins. It provides two operation blocks that can be filled with other automation blocks. Which path is executed depends on whether or not the condition is met.



What's next

You have built the base control-flow in the automation sequence.

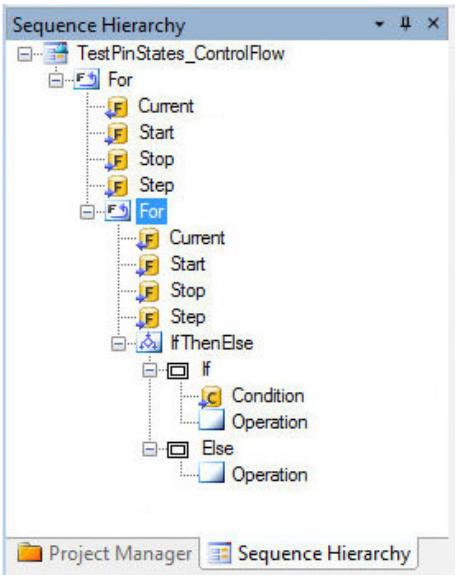
You can navigate in the sequence in the next step.

Step 5: How to Navigate In the Sequence

Introduction

In this step you will learn how to navigate in a sequence.

A detailed hierarchical view of the sequence you are editing in the Sequence Builder is provided in the Sequence Hierarchy Browser. It shows the internal structure of your control flow. To display the Sequence Hierarchy Browser, click the **Sequence Hierarchy** page.



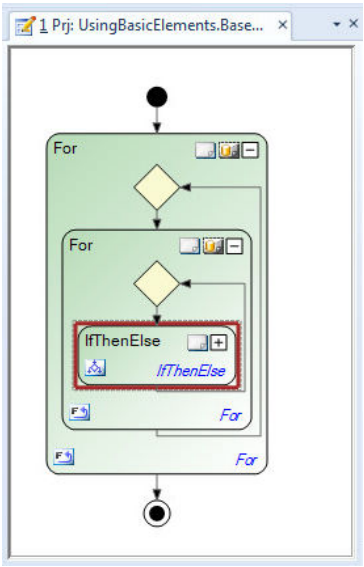
To focus on the essential parts of the sequence, you can reduce its complexity by collapsing subordinated elements. To recover the detailed view, you can expand it again.

Part 1

To collapse an element

1 In the Sequence Hierarchy Browser, collapse the node of the element you want to get an abstract view of, for example, the IfThenElse block.

The collapsed element is presented with a ⊞ icon in the Sequence Builder.



Tip

You can also collapse an element in the Sequence Builder. There are two ways to do this:

1. Double-click the element you want to mask in the Sequence Builder.
2. Click ⊟ in the command bar of the element you want to collapse.

Continue with next part

You collapsed an element in your sequence. You can now expand it again with in the next part.

Part 2

To expand an element

1 In the Sequence Hierarchy Browser, expand the node of the element you want to get a detailed view of.

Tip

You can also expand an element in the Sequence Builder. There are two ways to do this:

1. Double-click the collapsed element in the Sequence Builder.
2. Click ⊞ in the command bar of the element you want to expand.

What's next

You can save your project in the next step.

Step 6: How to Save a Project

Introduction

Having created your first project and automation sequence during the previous steps, you will now learn how to save your project.

Method 1

To save a project

1 Select the project root node UsingBasicElements in the Project Manager.

2 On the Quick Access Toolbar, click Save Project to save the project.

Note

It is not possible to rename a project. You must save it with a new name using the **Save As** command.

Using multiple projects

In principle, you can work with several projects in parallel, but only one project is active at a time. All project-specific commands apply only to the active project.

Method 2

To set the active project

1 Select the project you want to be the active one in the Project Manager.

What's next

You can close your project in the next step.

Step 7: How to Close a Project

Introduction

This step shows you how to close a project. When it is closed, the project tree will disappear from the Project Manager. All related charts and views in the Sequence Builder will also be closed.

Tip

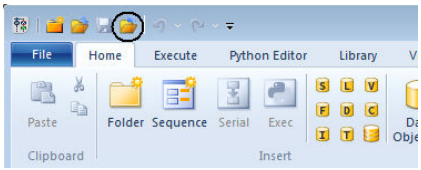
Before you close your project, you should compare your workings with the result of this lesson. Refer to Results of Lesson 1 on page 54.

Method 1

To close a project

1 Select UsingBasicElemets in the Project Manager.

2 On the Quick Access Toolbar, click Close Project to close the active project.



The active project and all its elements are closed.

Closing all open projects

AutomationDesk enables you to close all opened projects, both the active one and the nonactive ones in a single step.

Method 2

To close all projects

1 On the File ribbon, click Close All Projects to close all the opened projects in parallel.

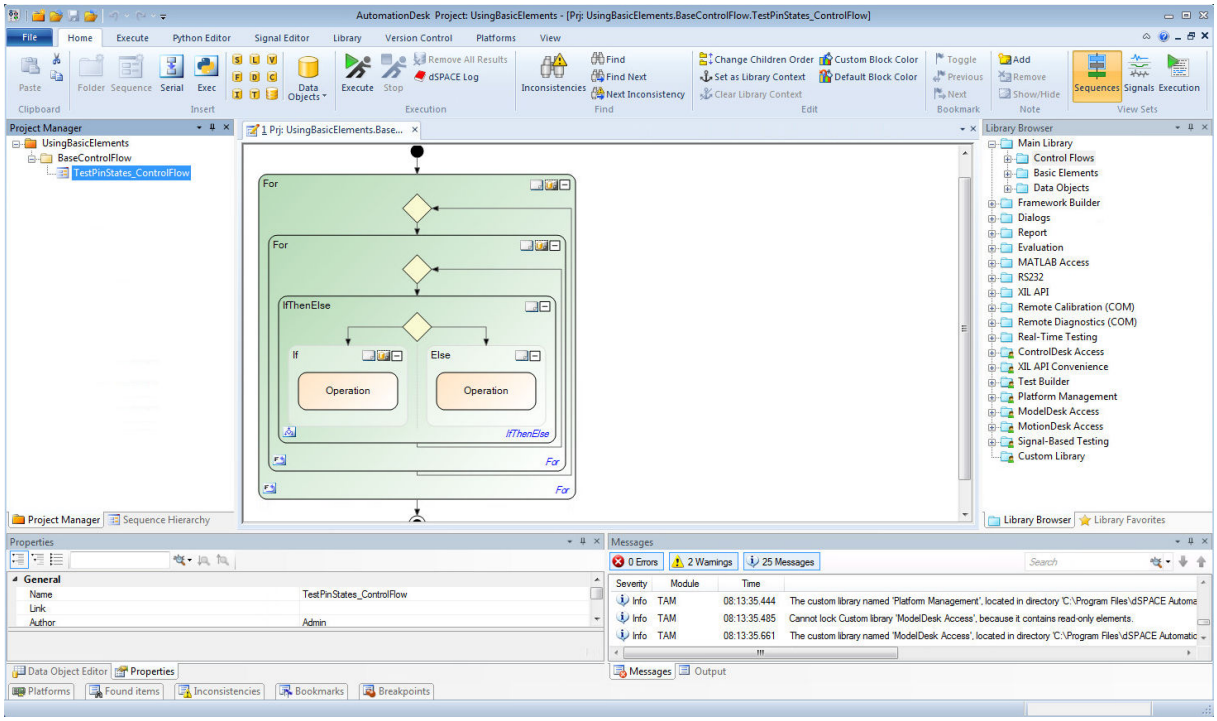
What's next

Now you can compare your work with the result you should have achieved. Refer to Results of Lesson 1 on page 54.

Results of Lesson 1

Results

If you have performed all the steps in this lesson, your AutomationDesk project should look like this:



In this lesson, you learned how to create an AutomationDesk project. First you generate a new project. To structure it, you can add a new folder to it. Having created a new sequence, you edit the sequence. You build the control flow in the graphical editor Sequence Builder by dragging automation blocks from the Library Browser. To get an abstract or a detailed view of the sequence, you can collapse or expand subordinated elements. To keep your project available for further AutomationDesk sessions, you must save it. To complete your work, you close the project.

Prepared Demo

The result of this lesson is stored in the prepared demo project *Tutorialdemo01.adpx*. You can find the demo file at `<DocumentsFolder>\Tutorial Demos`.

Further information

- For detailed information about the commands used, refer to Commands And Dialogs (AutomationDesk Basic Practices 🕮).
- For detailed information about the automation blocks used, refer to Automation Blocks (AutomationDesk Basic Practices 🕮).
- For information about basic principles of AutomationDesk and detailed descriptions for handling automation projects, refer to Managing Projects (AutomationDesk Basic Practices 🕮).

What's next

The next lesson (refer to Lesson 2: Parameterizing the Automation Sequence on page 57) shows you how to parameterize your automation sequence.

Lesson 2: Parameterizing the Automation Sequence

Introduction	You learn how to parameterize an automation sequence.

Introduction to Lesson 2

Task to be performed	This lesson describes how to parameterize the basic task prepared in lesson 1. For more information, refer to Example Task for Using the Basic Elements of AutomationDesk on page 36.
Before you begin	You have to know how to edit an automation sequence (refer to Lesson 1: Creating Your First Project on page 41).
What you will learn	This lesson shows you how to parameterize an automation sequence. ▪ You will copy an existing automation sequence. ▪ You will learn how to specify general properties of an automation block. ▪ You will parameterize your automation sequence using different methods.
Duration	Working through this lesson will take you about 40 minutes.
Summary	To check the correctness of the executed steps, refer to Results of Lesson 2 on page 74.
Starting point	The starting point of this lesson is the AutomationDesk project you created in lesson 1.

If you have not created your own AutomationDesk project as shown in the previous lesson, you can use the prepared demo project *Tutorialdemo01.adp*. It can be found at `<DocumentsFolder>\Tutorial Demos`.

Now you can start with the first step, see below.

Related topics

Basics

Lesson 1: Creating Your First Project....41
Results of Lesson 2....74

Step 1: How to Open an Existing Project

Introduction

The project you created in lesson 1 will now be enlarged. First you have to open it.

Part 1

To open an existing project

1. On the File ribbon, click Open - Project.
2. In the Open dialog, change to your working folder, for example, `<DocumentsFolder>\MyWorkingFolder`, and select the UsingBasicElements.adpx file before you click Open.

   The project, including the BasicControlFolder folder and the TestPinStates_ControlFlow sequence, is displayed in the Project Manager and can be used for further processing.

Continue with next part

When opened in the Project Manager, the project looks the same as if it was closed. By default, the Sequence Builder shows no associated view. In the next part, you open the project's sequence.

Part 2

To open an existing sequence

1. Double-click TestPinStates_ControlFlow in the Project Manager.

What's next

You can add a new folder to your project and copy the existing sequence to the new folder in the next step.

Step 2: How to Use an Existing Sequence

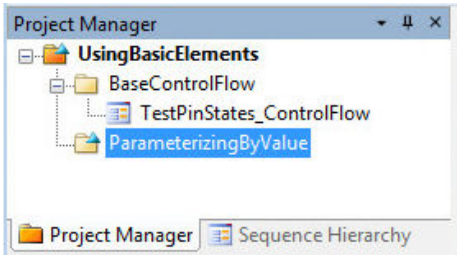
Introduction

This step explains how to use an existing sequence. You will generate a new project folder and copy an existing sequence into it.

Part 1

To add another folder to your project

1. In the Project Manager, click UsingBasicElements to select your project.
2. On the Home ribbon, click Insert - Folder. The new Folder element appears as a child of the project tree's root element in the Project Manager.
3. Select Folder, press **F2** and overwrite the folder's name with `ParameterizingByValue`.
4. Press **Enter** to apply the new name.



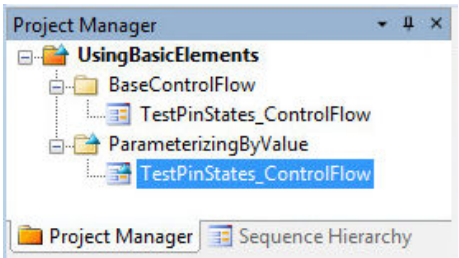
Continue with next part

You added another folder to your project. You can now copy and rename the existing sequence in the next part.

Part 2

To copy an existing sequence to a folder

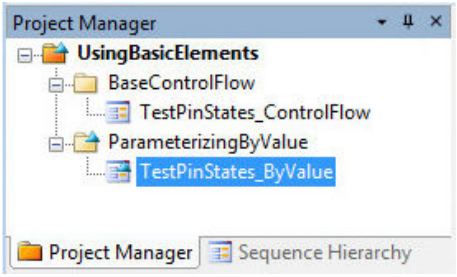
1. Select the TestPinStates_ControlFlow element in the Project Manager. Hold down the **Ctrl** key and drag the sequence from the BaseControlFlow folder to the ParameterizingByValue folder. The ParameterizingByValue folder now also contains the TestPinStates_ControlFlow sequence.



Tip

Alternatively, you can use the copy & paste shortcut keys to copy the sequence. Select TestPinStates_ControlFlow in the Project Manager. Press **Ctrl+C** to copy the sequence to the Clipboard. Select the ParameterizingByValue folder in the Project Manager. Press **Ctrl+V** to paste the sequence into the folder.

2 In the Project Manager, rename the sequence to `TestPinStates_ByValue`.



What's next

You can modify the properties of an automation block in the next step.

Step 3: How to Modify Automation Block Properties

Introduction

This step explains how to modify the general properties of an automation block. You can change its name, which is used in the Sequence Builder, to make the sequence clearer. You can store a brief description of the block for information.

Tip

- By copying the sequence, the properties of the source's automation blocks are assigned to the automation blocks of the copy.
- Besides the name and description properties, there are other automation block properties which will be described later on in the tutorial.

Now you will rename several automation blocks in your sequence. You will get to know different ways of doing this.

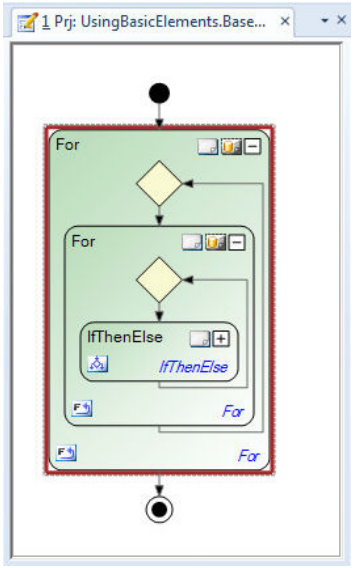
Part 1

To rename automation blocks using the Properties pane

1 In the Project Manager, double-click the TestPinStates_ByValue sequence to open it in the Sequence Builder and the Sequence Hierarchy Browser.

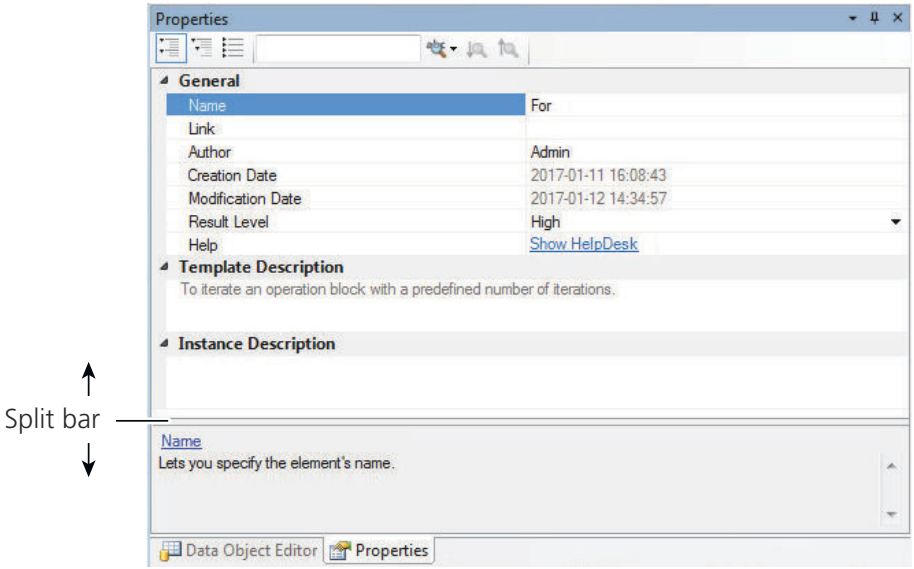
2 In the Sequence Builder, select the outer For block of the sequence.

The selected block is marked with a red frame.



In the Properties pane, the properties of the selected automation block are shown.

3 Adjust the size of the pane to display all properties of the For block.



4 In the edit field for the General - Name property, overwrite the current name of the automation block by typing `ForAllGroups`. When you press **Enter** the change is applied to the Sequence Builder and the Sequence Hierarchy Browser.

Continue with next part

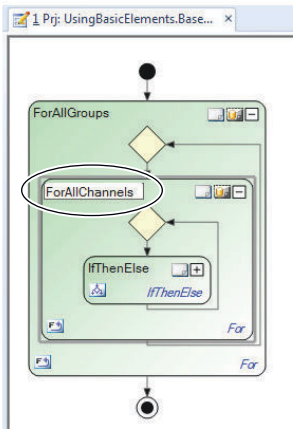
You renamed the outer For block. You can now rename the inner For block in the next part.

Part 2

To rename automation blocks by clicking the block name in the Sequence Builder

1 In the Sequence Builder, select the inner For block of the sequence.

2 Click the block name and change it to `ForAllChannels`.



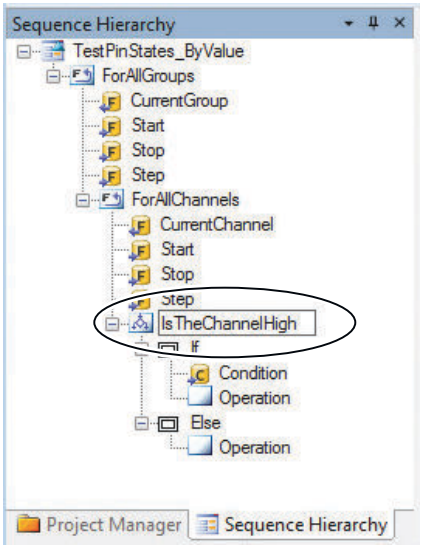
Continue with next part

You renamed the inner For block. You can now rename the IfThenElse block.

Part 3

To rename an automation blocks using F2 in the Sequence Hierarchy Browser

1 In the Hierarchy Browser, select the IfThenElse element, press **F2** and rename it to `IsTheChannelHigh`.



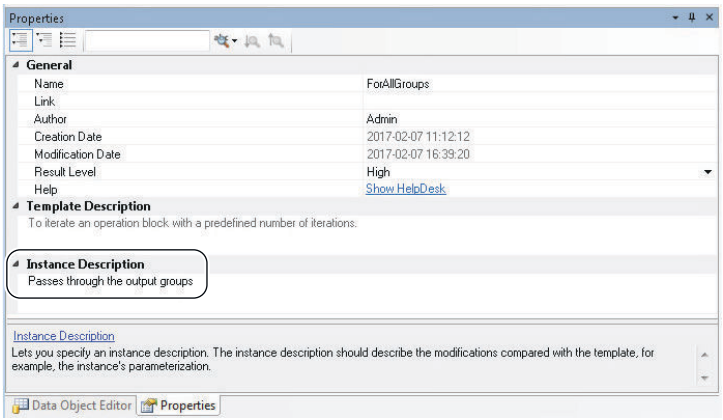
Continue with next part

Having renamed several automation blocks in your sequence, you will now store brief descriptions of some blocks in the next part.

Part 4

To specify an instance description of an automation block

1. In the Sequence Builder, select the ForAllGroups block.
2. In the Properties pane, enter `Passes through the output groups` in the Instance Description edit field.



Modifying other elements

If you want, you can edit the descriptions of the other sequence elements on your own.

What's next

You can now start parameterizing the automation blocks in your sequence in the next step.

Step 4: How to Parameterize Automation Blocks by Value

Introduction

Having created the control flow, you will now learn how to parameterize the automation blocks in your sequence.

There are two methods of doing this:

- By value

  The parametrization is specified locally in data objects that are provided by the automation blocks. This method is shown in the current step.
- By reference

  The parameterization is specified via a reference to a project-specific data object. This method is shown later in the next step.

For both methods, parameterization is specified using the Data Object Editor.

First the *For* blocks are parameterized with the number of iterations necessary to process the groups and channels of the test task.

Part 1

To parameterize the control flow blocks by value

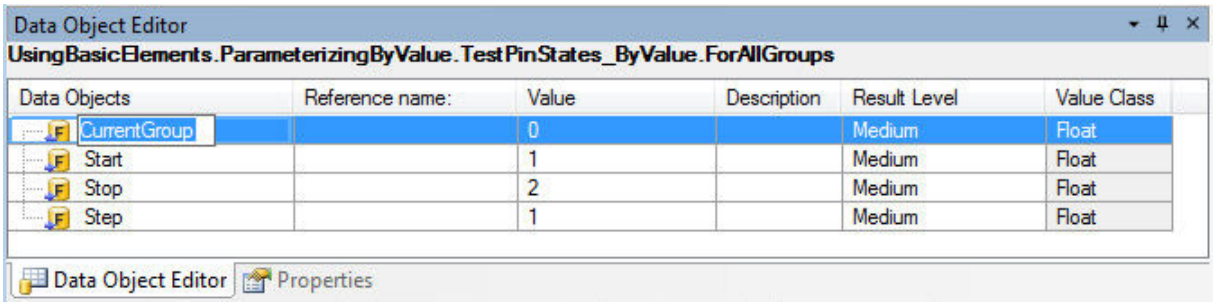
1 In the Sequence Builder, select the ForAllGroups block.

In the Data Object Editor, the parameterization of the selected block is displayed.

The default values in the Value column fit to the test case of two channel groups:

- Start = 1
- Stop = 2
- Step = 1

2 In the Data Object Editor, select the Current data object, press **F2**, and rename the data object to `CurrentGroup`.



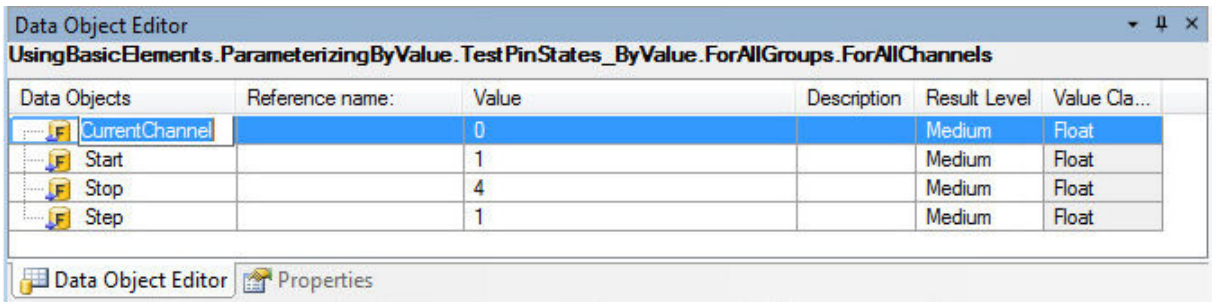
This distinguishes the Current data objects of the inner and the outer For block.

3 In the Sequence Builder, select the ForAllChannels block.

4 In the Data Object Editor, overwrite the default values in the Value edit field to parameterize that each group consists of 4 channels:

- Start = 1
- Stop = 4
- Step = 1

5 In the Data Object Editor, select the Current data object, press **F2**, and rename the data object to `CurrentChannel`.



Continue with next part

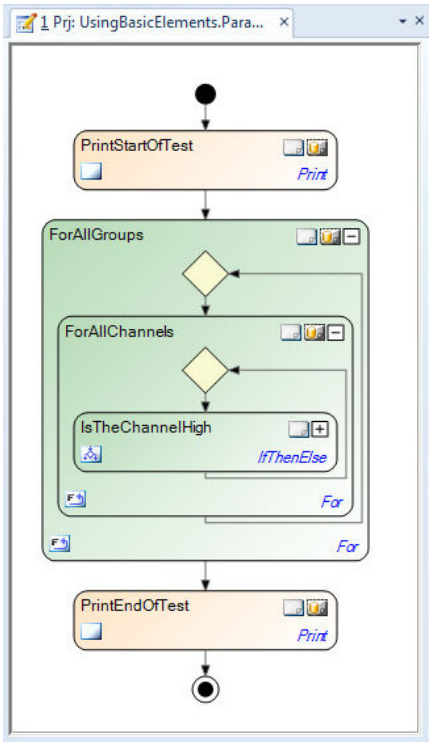
You parameterized the *For* blocks of your sequence by value.

In the next part, you add blocks to the sequence that announce the start and the end of sequence processing in the Output Viewer.

Part 2

To add blocks for printing

1. Open the TestPinStates_ByValue sequence in the Sequence Builder.
2. In the Library Browser, select the Print block in the Basic Elements folder of the Main Library.
3. From the Library Browser, drag a Print block to the Sequence Builder and drop it above the ForAllGroups block.
4. Rename the new Print block to `PrintStartOfTest`.
5. In the same way, add a Print block below the ForAllGroups block and rename it to `PrintEndOfTest`.



Continue with next part

You added blocks to the sequence that print messages. In the next part, you parameterize these blocks by value with the text to be printed and the color to be used for this.

Part 3

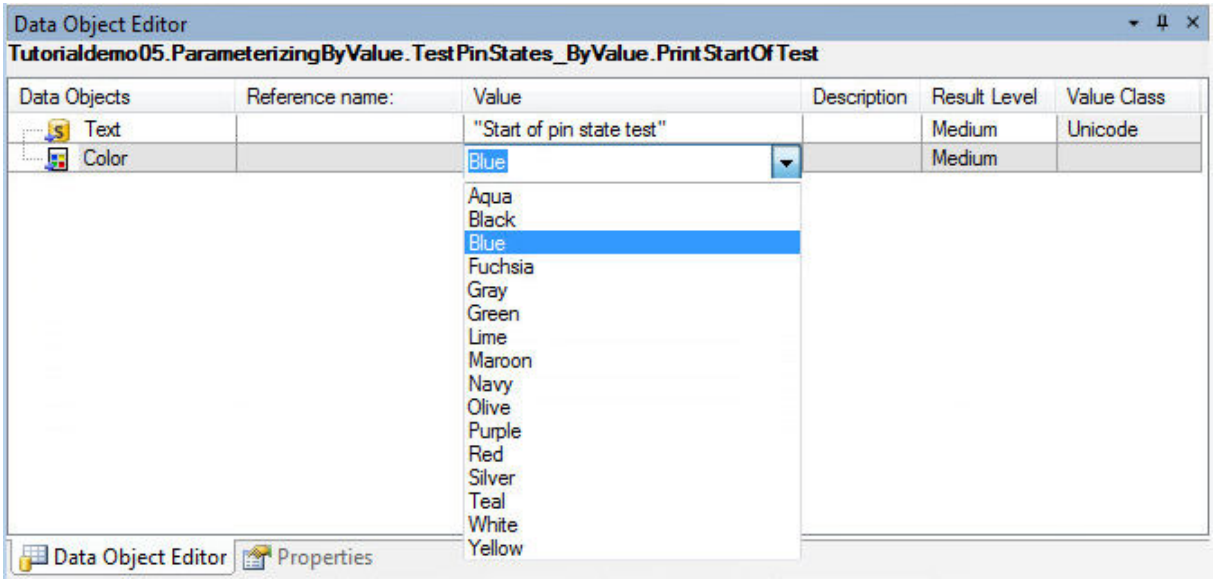
To parameterize the blocks for printing

1. In the Sequence Builder, select the PrintStartOfTest block.

   In the Data Object Editor, the parameterization of the selected block is displayed.
2. In the Data Object Editor, specify the message to be printed by entering the following text in the Value edit field of the Text data object:

```
Start of pin state test
```

3 In the Value edit field of the Color data object, select Blue from the list to specify the color of the message to be printed.



After you pressed **Enter** or changed the focus to another element the RGB color code of blue, `(0, 0, 255)`, is displayed in the edit field.

4 For the PrintEndOfTest block, specify the following message to be printed in blue:

```
End of pin state test
```

Tip

As an alternative for specifying a value in the Data Object Editor, you can double-click the data object's element to open the data object in the Value Editor. The appearance of the Value Editor depends on the type of data object that you edit.

What's next

In this step, you parameterized automation blocks by the value of their local data objects.

In the next step, you learn how to parameterize blocks by referencing to project-specific data objects.

Step 5: How to Parameterize Automation Blocks by Reference

Introduction

This step shows how to define project-specific data objects and then parameterize automation blocks with those data objects by reference. To

parameterize by reference means that you do not specify a value for a data object, but the name of another data object that holds the value.

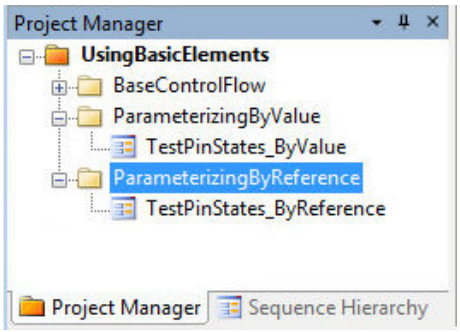
First you duplicate the ParameterizingByValue project folder to keep the current version of your sequence. This allows you to reconstruct the main learning steps again.

Part 1

To duplicate the project folder

1. In the Project Manager, right-click the ParameterizingByValue folder and select Copy from the context menu to copy the folder to the Clipboard.
2. Right-click the UsingBasicElements project and select Paste from the context menu to insert the folder from the Clipboard to the project.

   A folder named ParameterizingByValue1 is added to the project.
3. Rename the new folder to `ParameterizingByReference` and the sequence it contains to `TestPinStates_ByReference`.



Continue with the next part

Next, you generate a two-dimensional list to represent the grouped output channels of the electronic device to be tested. Remember that a value of `1` represents a *High* level and `0` a *Low* level. The list is parameterized with arbitrary `0` and `1` values.

Furthermore, two integer data objects are needed to parameterize the number of groups and the number of channels of the device's output.

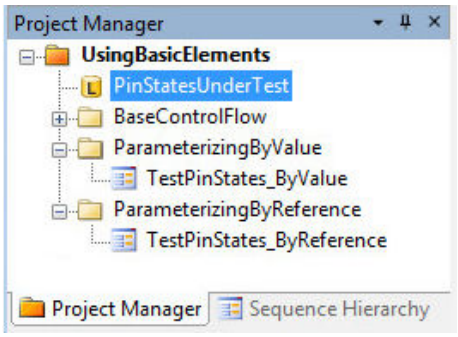
Part 2

To define project-specific data objects

1. Right-click the UsingBasicElements project and select New Data Object - List from the context menu to create a new data object of list type.

   The List data object displays at the top level of the UsingBasicElements project.

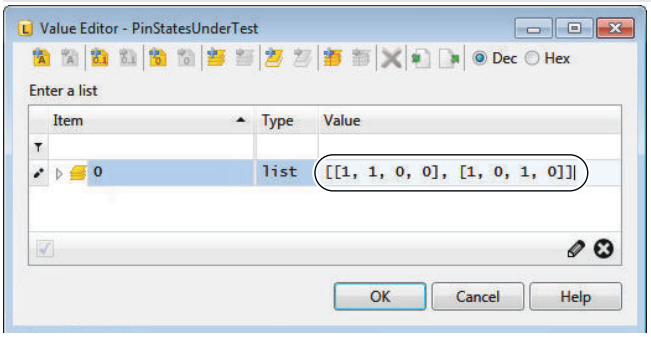
2 Rename the new data object to `PinStatesUnderTest` by using **F2**.



3 Choose Edit from the context menu of the PinStatesUnderTest element to open the Value Editor.

4 Specify a 2x4 matrix by adding two nested lists with four integers each to the PinStates data object. Replace the [] in the Value edit field with the following and click OK to close the dialog:

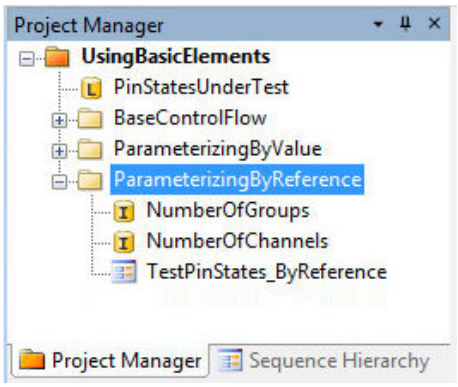
```
[[1, 1, 0, 0], [1, 0, 1, 0]]
```



5 In the Project Manager, right-click the ParameterizingByReference folder and select New Data Object - Integer from the context menu to create a new integer data object.

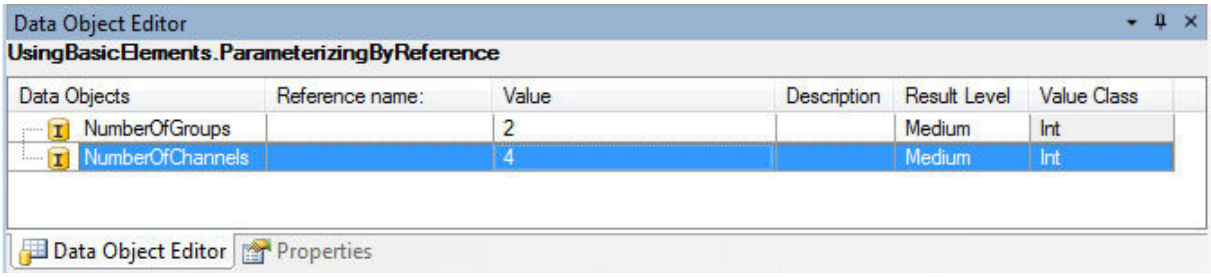
6 On the Home ribbon, in the Insert ribbon group click to use an alternative way for creating another integer data object in the selected folder.

7 Rename the two new integer data objects to `NumberOfGroups` and `NumberOfChannels`.



8 Select the ParameterizingByReference folder to show its data objects in the Data Object Editor.

9 Edit the values of the integer data objects:
- NumberOfGroups = 2
- NumberOfChannels = 4



Continue with next part

You created project-specific data objects that hold values.

In the next part, you use the names of those data objects as the reference names to parameterize data objects in an automation block.

You can specify reference names via the related edit fields in the Data Object Editor. A more convenient way is to select them from the available data objects provided by the Data Object Selector.

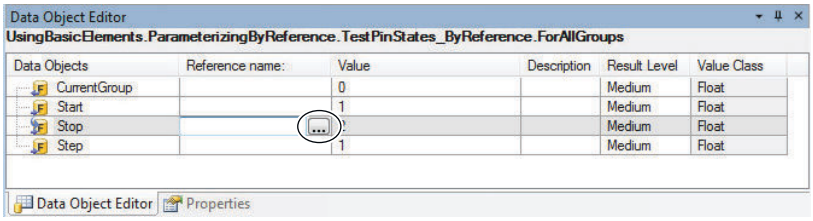
Part 3

To parameterize automation blocks by referencing project-specific data objects

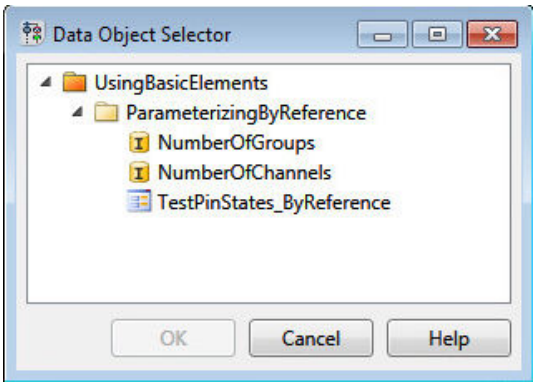
1 In the Sequence Builder, select the ForAllGroups block to show its data objects in the Data Object Editor.

2 In the Data Object Editor, select the cell of the Stop data object in the Reference Name column to make the cell editable.

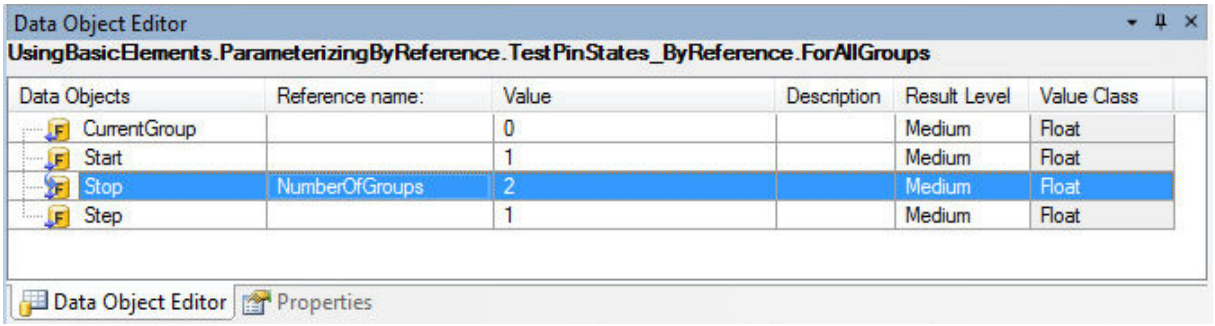
3 Click the Browse button in the cell to open the Data Object Selector.



The Data Object Selector opens.



4 Select the NumberOfGroups data object and click **OK** to close the Data Object Selector.

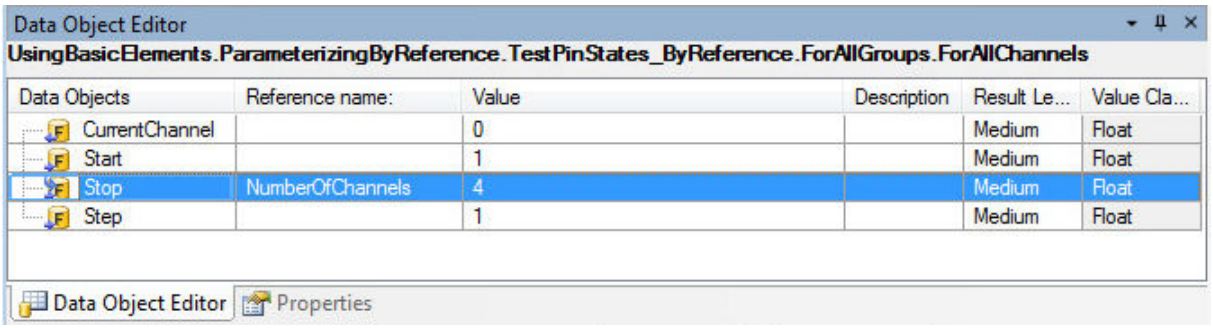


Now the name of the project-specific NumberOfGroups data object is specified as the reference of the Stop data object of the ForAllGroups block. This means, that you can change the stop value of the ForAllGroups block by editing the NumberOfGroups data object.

A small blue arrow beside the corresponding symbol indicates that Stop is parameterized by reference.

5 In the Sequence Builder, select the ForAllChannels block to show its data objects in the Data Object Editor.

6 In the Data Object Editor, set the NumberOfChannels data object as the reference for the block's Stop data object.



Revoking references

Tip

To revoke a reference, open the Data Object Editor. Select the element whose reference you want to cancel and delete its Reference Name entry.

What's next

You defined project-specific data objects and used them to parameterize blocks in your sequence by reference.

In the next step, you access the value of data objects in Python.

Step 6: How to Access the Value of Data Objects via the _AD_ Alias

Introduction

To access the value of a data object in a Python expression, you can prefix the name of the data object with the _AD_ alias that is provided by AutomationDesk.

Specifying conditions

In this step, you will learn how to specify the condition of an IfThenElse block. The condition has to be defined as a Python expression.

In our automation sequence, the condition of the IfThenElse block involves checking if the current output signal is of high level. The value we have to check is defined by its position in the list array. The position is specified by the current counters of the two *for-loops*.

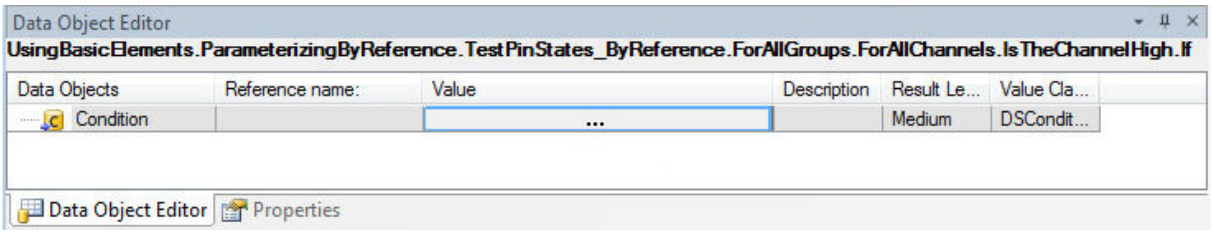
Part 1

To specify the condition of the IfThenElse block

1 Open the TestPinStates_ByReference sequence in the Sequence Builder.

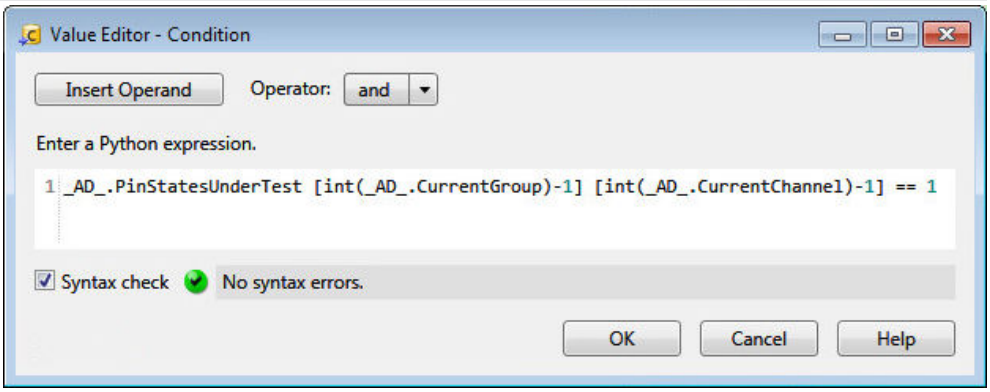
2 Select the If branch of the IfThenElse block to show its data objects in the Data Object Editor.

3 Click the **Browse** button in the Value column of the Condition data object to open the Condition Editor.



4 In the Condition Editor, replace `False` by the following Python expression:

```
_AD_.PinStatesUnderTest [int(_AD_.CurrentGroup)-1] [int(_AD_.CurrentChannel)-1] == 1
```



The project-specific two-dimensional list `PinStatesUnderTest` represents the actual pin states. The indices to access the list are determined by the current values of the nested *for-loops*. The variables are accessed via the _AD_ alias.

5 Press **OK** to close the Condition Editor.

Continue with the next part

You specified the condition of the *IfThenElse* block. It checks whether the value at the current position in the two-dimensional list of the output channels is equal to **1**, which represents a **High** level.

In the next part, you add blocks to the sequence, that report the found output level.

Part 2

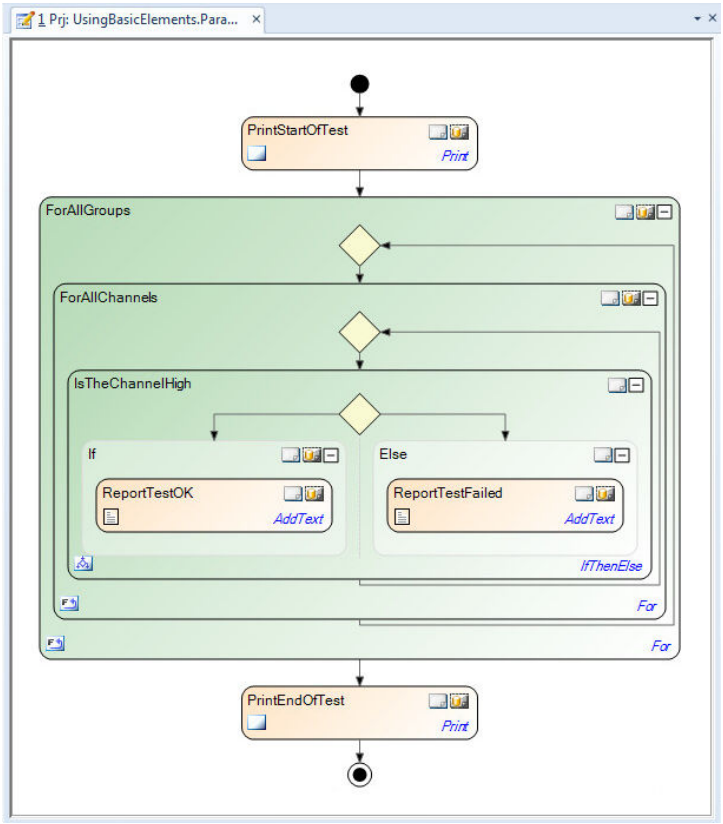
To add blocks for reporting

1 In the Library Browser, select the AddText block in the Report library.

2 Drag an AddText block to the Sequence Builder and drop it to the Operation field of the If path of the IsTheChannelHigh block

3 Rename the new block to `ReportTestOK`.

4 Drag another AddText block to the Operation field of the Else path and rename it to `ReportTestFailed`.



Continue with the next part

The sequence now contains the blocks for reporting.

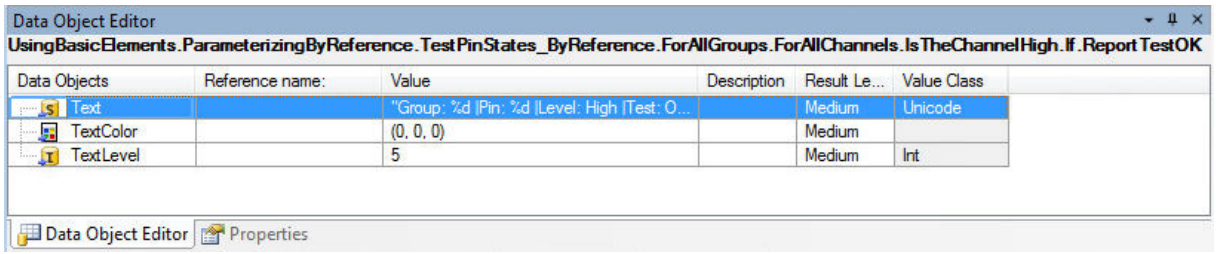
In the next part, you specify the text to be reported via a Python expression with a format string. The current position in the two-dimensional list is determined via the _AD_ alias.

Part 3

To integrate data object values in a text

1 In the Sequence Builder, select the ReportTestOK block.

In the Data Object Editor, the parameterization of the selected block is displayed.



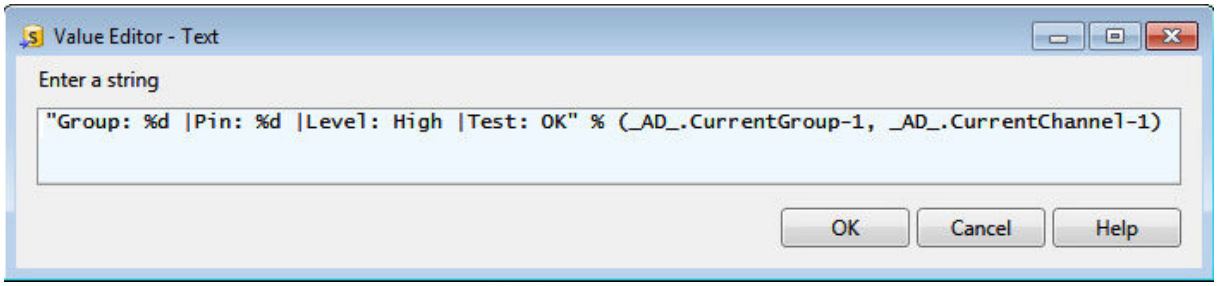
2 In the Data Object Editor, double-click the **Data Objects** column of the **Text** data object.

The **Text** data object is opened in the Value Editor

3 Enter the following Python expression and click **OK**:

```
"Group: %d |Pin: %d |Level: High |Test: OK" %
(_AD_.CurrentGroup-1, _AD_.CurrentChannel-1)
```

The text to be added to the report is formatted via a format string. The contained variables are accessed via the _AD_ alias.



4 In the same way, enter the following Python expression for the **Text** data object of the **ReportTestFailed**:

```
"Group: %d |Pin: %d |Level:  Low |Test: Failed" %
(_AD_.CurrentGroup-1, _AD_.CurrentChannel-1)
```

What's next

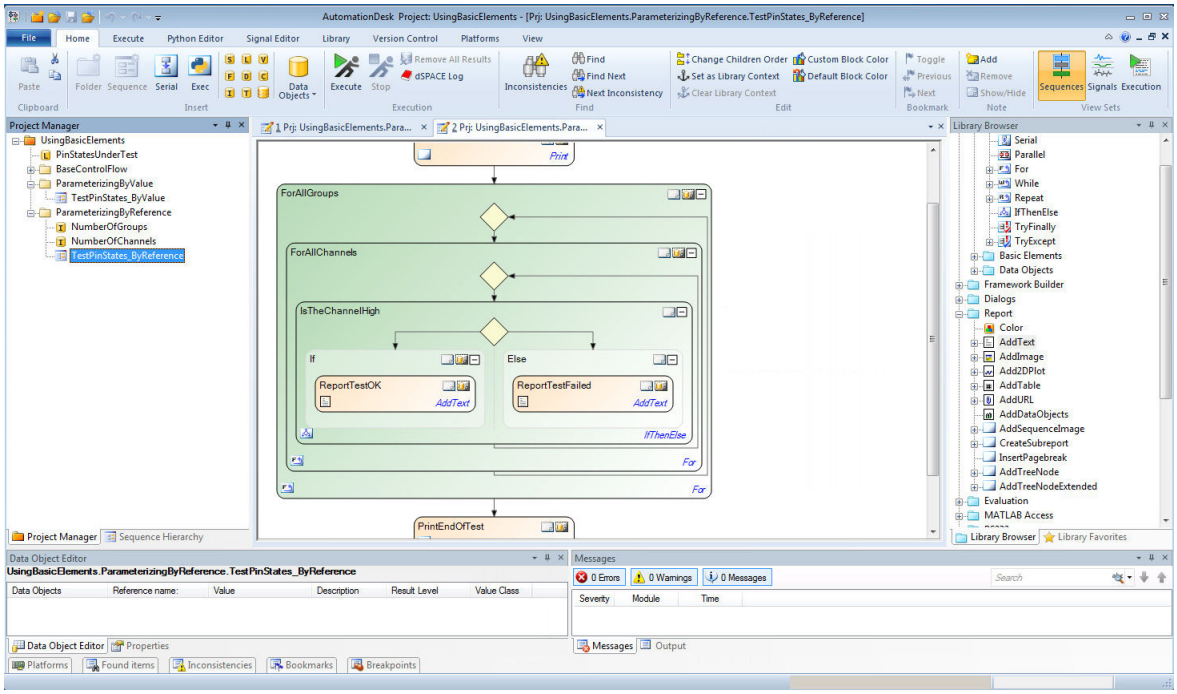
In your sequence, you parameterized an automation block via Python expressions. The sequence is now ready to be executed.

To compare your work with the result you wanted to achieve, refer to Results of Lesson 2 on page 74.

Results of Lesson 2

Results

If you performed all the steps in this lesson, your AutomationDesk project contains the new *ParameterizingByValue* and *ParameterizingByReference* folders, which contain different development stages of the workflow's parameterization.



You have learned how to modify the general properties of automation blocks for the purpose of information and to get a clear view of the sequence.

You have also learned how to parameterize an automation sequence. There are different ways of parameterizing an automation block. To parameterize by value means to use local data objects. To parameterize by reference means to define project-specific data objects and to parameterize an automation block by referencing them.

You learned to access project-specific data objects in Python expressions via the _AD_ alias. This is used to parameterize the automation block that you utilize to print messages to the output and to the report during execution.

Prepared Demo

If you had problems parameterizing your own AutomationDesk project, or if you want to compare your results with the result we wanted to achieve in this lesson, you can look at the prepared demo project *Tutorialdemo02.adpx*. You can find it at `<DocumentsFolder>\Tutorial Demos`.

Further information

- For detailed information about the commands used, refer to Commands And Dialogs (AutomationDesk Basic Practices 🕮).
- For detailed information about the automation blocks used, refer to Automation Blocks (AutomationDesk Basic Practices 🕮).
- For information about the basic principles of AutomationDesk and a detailed description of parameterizing automation sequences, refer to Building Automation Sequences (AutomationDesk Basic Practices 🕮).

What's next

The next lesson (refer to Lesson 3: Executing the Automation Sequence on page 77) shows you how to execute an automation sequence. It explains how you can view the execution results and how to generate a report.

Lesson 3: Executing the Automation Sequence

Introduction	You learn how to execute an automation sequence and generate a report.

Introduction to Lesson 3

Task to be performed	This lesson describes how to execute the basic task that you prepared in lesson 1 and 2. For more information, refer to Example Task for Using the Basic Elements of AutomationDesk on page 36.
Before you begin	You have to know how to parameterize an automation sequence (refer to Lesson 2: Parameterizing the Automation Sequence on page 57).
What you will learn	To execute an automation sequence, you have to make several settings with regard to the result and the report generation. ▪ You will learn how to specify result parameters which affect the result output of an execution. ▪ You will execute an automation sequence. ▪ You will also learn how to specify report parameters which affect the report output. ▪ You will generate a report.
Duration	Working through this lesson will take you about 15 minutes.
Summary	To check the correctness of the executed steps, refer to Results of Lesson 3 on page 87.

Starting point

The starting point of this lesson is the AutomationDesk project you created in lesson 2.

If you have not created your own AutomationDesk project as shown in the previous lessons, you can use the prepared demo project *Tutorialdemo02.adp*. It can be found at `<DocumentsFolder>\Tutorial Demos`.

Now you can start with the first step, see below.

Related topics

Basics

Lesson 2: Parameterizing the Automation Sequence 57
Results of Lesson 3 87

Step 1: How to Specify Result Parameters

Introduction

In this lesson you will execute the *TestPinStates_ByReference* sequence you created during the previous lessons. Before you start the execution, you should specify the result parameters of the sequence's automation blocks. Each block and each of its data objects has an attribute called result level that decides if the element can be included in the result. During the sequence's execution, the result level is compared with the record depth, which you must specify at the start of the execution.

The result level can have 3 different values:

- None: The information of the block or the data object never appears in the result.
- Medium: The information can be included in the result.
- High: The information is always added to the result, provided that a result is generated.

By default, the result level of an automation block is specified as *High* and a data object's result level is set to *Medium*.

Note

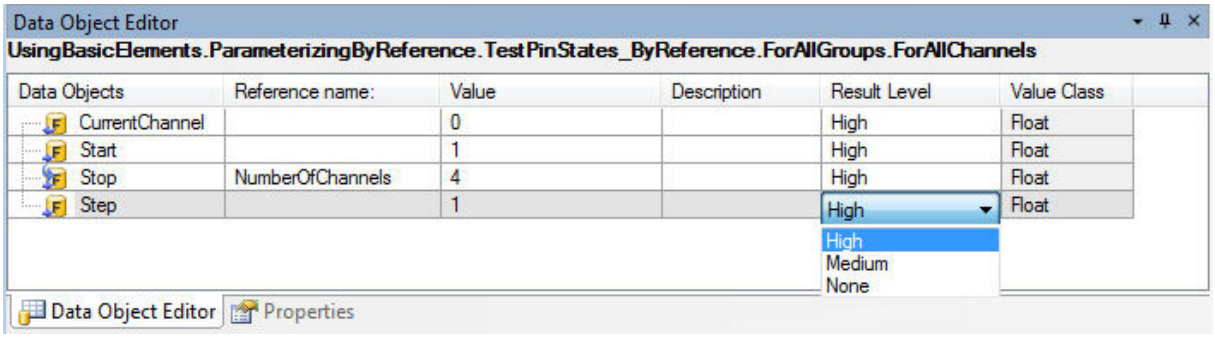
If you set the result level of an automation block to None, none of its subordinated automation blocks will be included in the result either, regardless of their own result levels.

You will now change the result level of the data objects of the *ForAllChannels* block from *Medium* to *High*. Hence, these data objects will be included in the result.

Method

To specify the result level of a data object

1. Select the ForAllChannels block to show its data objects in the Data Object Editor. Each data object has a default result level.
2. Select "High" in the Result Level drop-down list associated with the data object CurrentChannel.
3. Modify the result level of the other 3 data objects in the same way.



Specifying the result level of an automation block

Tip

The modification of an automation block's result level should be done in the Properties pane. This can be opened, for example, by choosing View Properties from the block's context menu.

What's next

You can now execute the sequence in the next step.

Step 2: How to Execute a Sequence

Introduction

Having defined the result level, you will now execute your sequence.

When starting the execution, you must define the record depth. This is the final decision on which blocks and data objects are to be included in the result. The record depth is compared with the result levels of the automation blocks and data objects. The following table shows you the relation between result level and

record depth. You can see which information will be added to the result for each result parameter setting.

		Record Depth		
		None	High only	High and medium
Result Level	High	—	✓	✓
	Medium	—	—	✓
	None	—	—	—

You can also store a brief description of the execution for information.

The execution of a sequence is started from within the Platform Manager. You can execute the sequence on different levels of the project tree, which is on the project's root element, on a folder's node, or on a sequence's node. When execution is started, all sequences that are below the execution's starting point are executed and their information is passed to the result.

You will now start the execution of your *TestPinStates_ByReference* sequence. You will execute it on the sequence's level, so only this single sequence will be executed.

Tip

You should open the Output Viewer by clicking the Output page in the Tool Window before you start the execution. At run time you can watch the output of the sequence's print commands in the Output Viewer.

Note

Automation blocks that use COM objects, for example, most of those from the ModelDesk Access library, should not be executed individually.

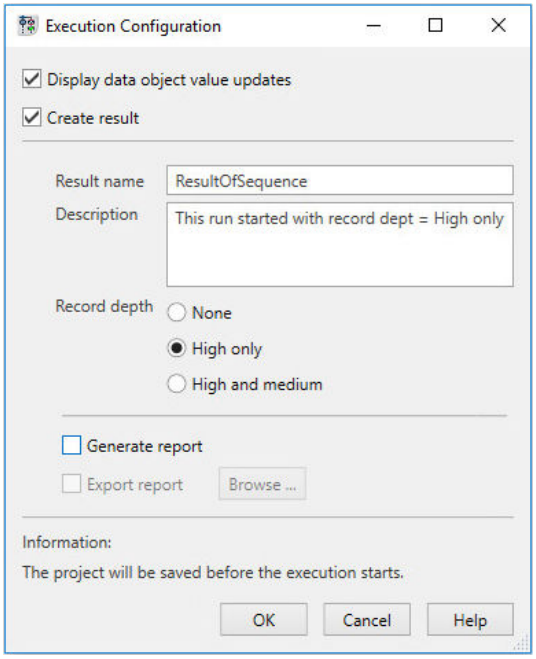
Method

To execute a sequence

1. In the Project Manager, click TestPinStates_ByReference to select your sequence.
2. On the Home ribbon, click Execution - Execute to open the Execution Configuration dialog.
3. Check the Display data object value updates option. This lets you observe the changing of the data object values in the Sequence Builder during the execution of the sequence.
4. Check the Create result option. It is set by default. If you want to execute a sequence without logging any data, you can clear this checkbox.
5. Rename the default result name by entering `ResultOfSequence` in the Result name edit field.
6. Under Record depth, select the High only checkbox.
   All blocks and data objects with a High result level are added to the result.

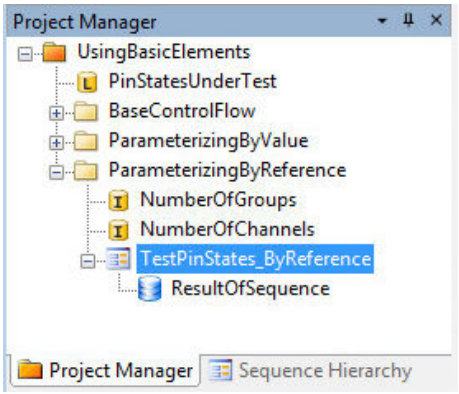
7 Type `This run was started with record depth = High only` in the Description edit field.

8 Clear the Generate report checkbox. The configuration and the generation of a report will be described in detail in the steps 4 and 5 of this lesson.



9 Click OK to confirm the parameter settings and start the execution.

At the execution's start, a result node is created as a child element of the starting point in the project tree. By default, the Result Browser is opened automatically. You can change this behavior on the General page of the AutomationDesk Options dialog.



What's next

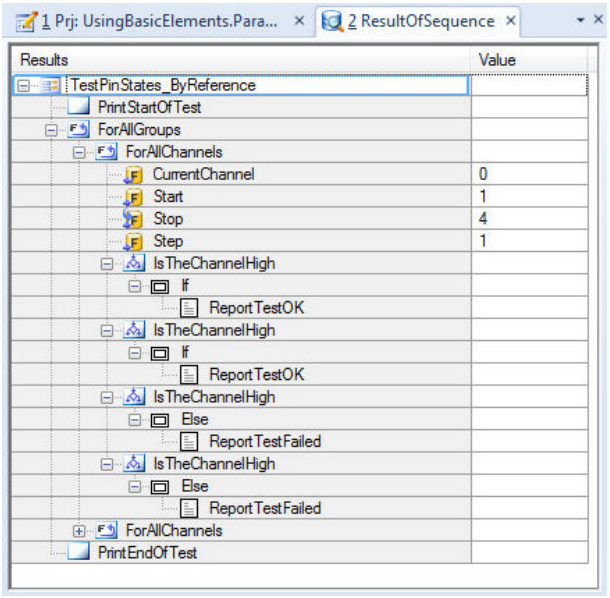
The next step explains how to view the results of the execution.

Step 3: How to View the Results

Introduction

Each executed automation task with a created result appears in the Project Manager, represented by a result node. The result node is a child element of the tree element that execution of the sequence was started from.

You can view the details of the results by using the Result Browser.



The result is displayed as a structured result tree. The result details contain the symbol, name and value of each executed automation block.

Having executed the *TestPinStates_ByReference* sequence in the previous step, you will now view the results of the execution.

Method

To view the results

1 If the Result Browser is closed, choose **View Results** from the context menu of the **ResultOfSequence** result in the Project Manager to open it.

All the automation blocks in your sequence appear in the result.

The default result level of a data object is set to Medium. Since you changed the result level of the **ForAllChannels** block's data objects to **High** in the previous step, the result contains data information on these data objects. No other data objects appear in the result tree.

Tip

You can save the result as HTML and PDF format by using the **Export** command from the result node's context menu.

What's next

You can generate a report based on the execution result. This will be done in the next step.

Step 4: How to Specify Report Parameters

Introduction

If you want to generate a report from an execution result, you have to specify some report parameters beforehand.

Output formats

A report is based on the result data stored in XML format. Depending on the style sheet used, you can generate the report in HTML or PDF format.

Available style sheets

The standard style sheets provide a default layout of the reports. If you want to use a different layout, you can use a custom style sheet. Examples of custom style sheets can be found at `<DocumentsFolder>\Custom Report Stylesheets`. You can change the alignment of the logo and replace the default logo with another no matter which style sheet is selected.

Report attributes

The report can contain predefined information data, such as the name of the reported element. These data are called attributes. There are standard attributes, which are provided by the execution result for a sequence, and additional attributes. With the additional attributes you can add the following information to a report:

- Project and folder information
- Content of report blocks in the executed sequence (project, folder, sequence)
- Descriptions of all elements included in the report
- Result states of all elements included in the report (passed, failed, undefined, executed)

Report parameters

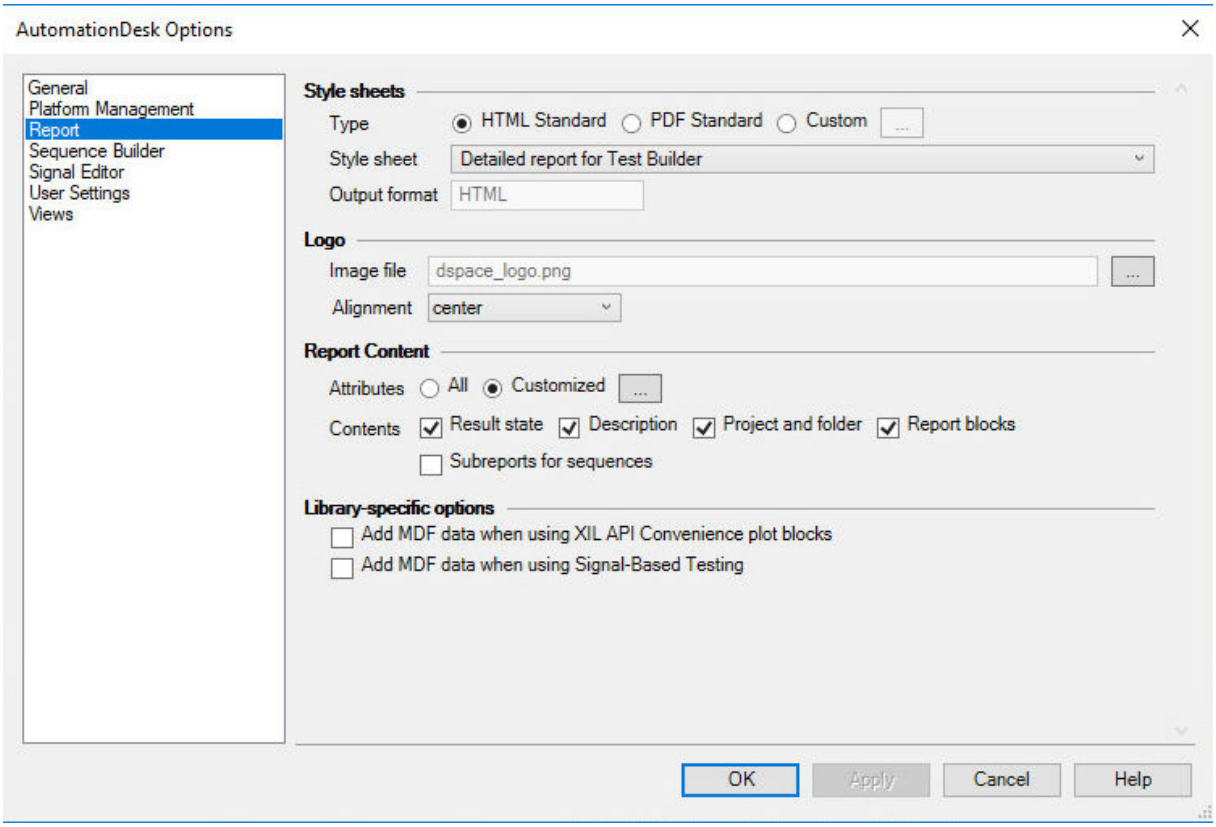
The above report parameters can be specified on the Report page of the AutomationDesk Options dialog. They are used the next time a report is generated.

Method

To specify report parameters

1 On the File ribbon, click Options to open the AutomationDesk Options dialog.

2 Switch to the Report page of the AutomationDesk Options dialog.



3 Select the HTML Standard style sheet type to generate reports with the AutomationDesk standard layout in HTML format.

4 Select Classic report as the style sheet for the report.

5 In the Report Content frame, select the Customized option and click the … Browse button.

The Define Report Header dialog opens.

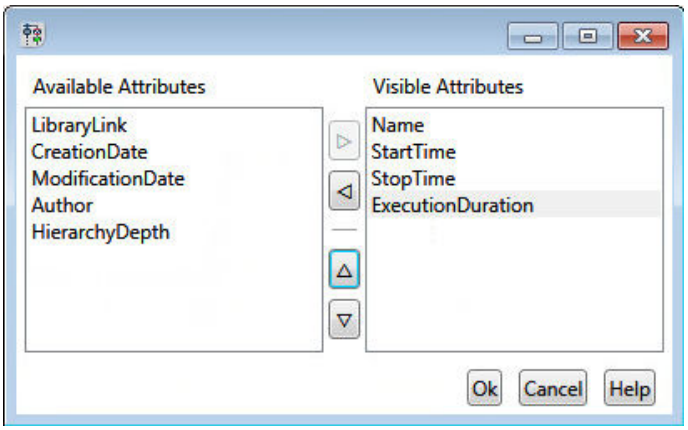
6 In the Available Attributes list, select the Start time attribute and click the ▷ button to move it to the Visible Attributes list.

In the same way, move the Stop time, Execution Duration, and Name attributes to the Visible Attributes list

If you generate a report, the selected attributes will now be displayed in it.

7 Select the Name attribute in the Visible Attributes list and click the △ button until it is displayed at the top of the list.

This setting specifies the order of the attributes in the report.



8 Click **OK** to close the dialog.

9 In the Contents group of the Report Content frame, select the Description checkbox .

The descriptions which you have entered in the Properties pane for a Project, Folder and Sequence are added to the report.

The default options for the attributes remain unchanged.

You have specified the report parameters. These settings are used for the next report generation.

What's next

You can now start the report generation in the next step.

Step 5: How to Generate a Report

Introduction

Having defined the report settings in the previous step, you can now generate a report on the execution results.

When report generation has finished, the report appears as a child element of the result in the Project Manager.

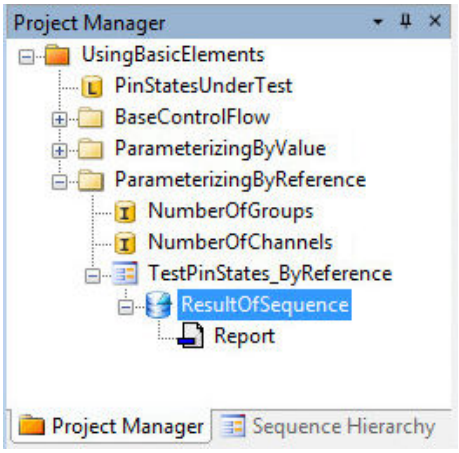
Part 1

To generate a report

1 Choose Generate Report from the context menu of the result node in the Project Manager to start the report generation.

By default, the first generated report of a result is called Report.

The newly created report appears as a child element of the result in the Project Manager.



Tip

The report is stored in the folder structure of your AutomationDesk project. You can store a copy of the generated report by exporting it via Export Report from the context menu of the report node in the Project Manager.

Continue with next part

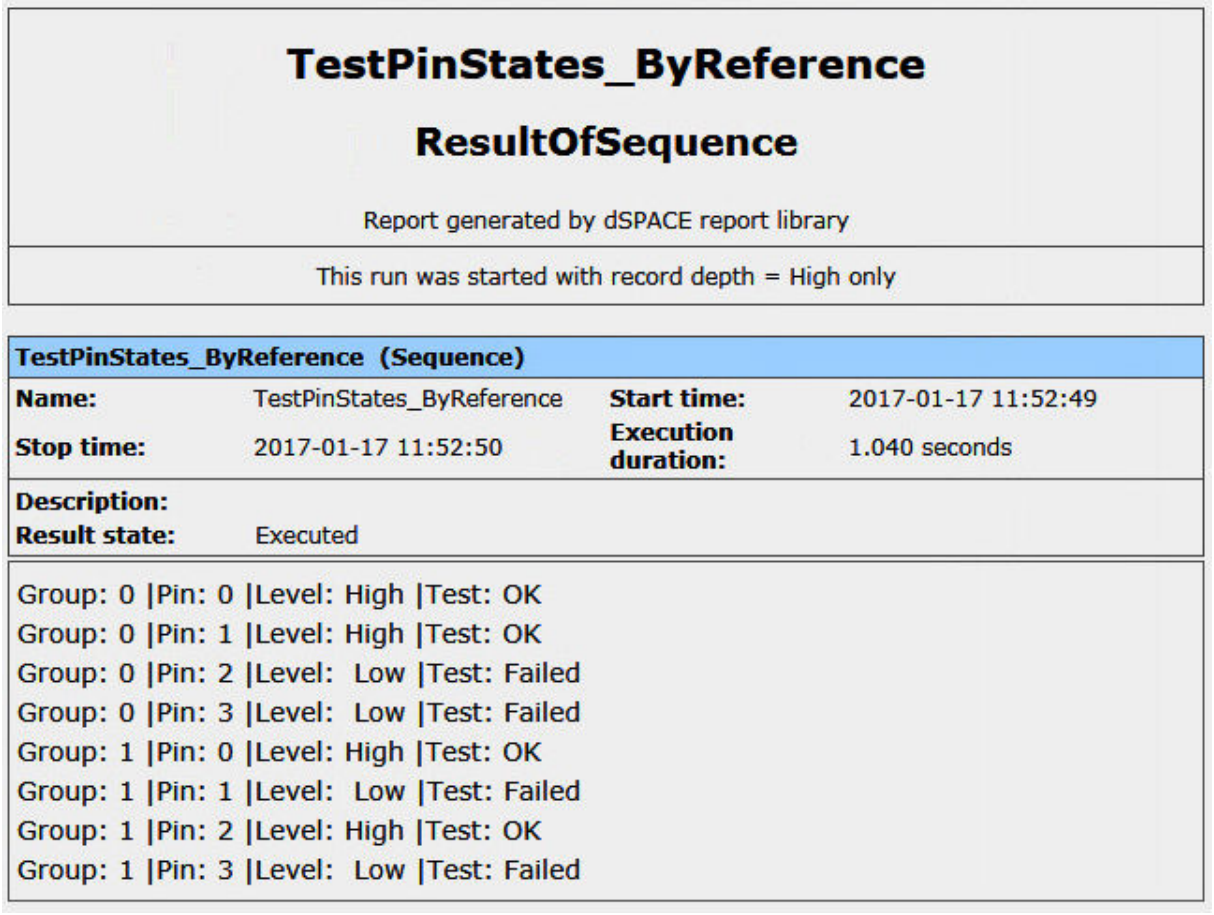
You generated a report. To open it continue with the next part.

Part 2

To open the report

1 In the Project Manager, open the context menu of the report element.

2 Choose View Report to open the report.



TestPinStates_ByReference

ResultOfSequence

Report generated by dSPACE report library

This run was started with record depth = High only

TestPinStates_ByReference (Sequence)			
Name:	TestPinStates_ByReference	**Start time:**	2017-01-17 11:52:49
Stop time:	2017-01-17 11:52:50	**Execution duration:**	1.040 seconds
Description:			
Result state:	Executed		

Group: 0 |Pin: 0 |Level: High |Test: OK
Group: 0 |Pin: 1 |Level: High |Test: OK
Group: 0 |Pin: 2 |Level: Low |Test: Failed
Group: 0 |Pin: 3 |Level: Low |Test: Failed
Group: 1 |Pin: 0 |Level: High |Test: OK
Group: 1 |Pin: 1 |Level: Low |Test: Failed
Group: 1 |Pin: 2 |Level: High |Test: OK
Group: 1 |Pin: 3 |Level: Low |Test: Failed

Tip

You can also open the report by double-clicking the report element in the Project Manager.

What's next

To compare your work with the result you should have achieved, refer to Results of Lesson 3 on page 87.

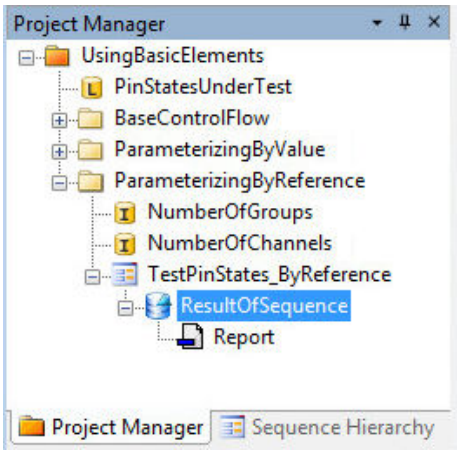
Results of Lesson 3

Results

If you have performed all the steps as described in this lesson, your AutomationDesk project has 2 new elements accessible in the Project Manager:

- The result of the sequence's execution as a child element of the *TestPinStates_ByReference* sequence.

- The report as a child element of the *ResultOfSequence* element.



You have learned how to execute an automation sequence and how to view the results of the execution. Before you can start an execution, you have to specify some result parameters. By specifying the result level and result depth, you define which elements of the automation sequence will be included in the result, and in which cases.

When the execution is finished, you can view the result in the Result Browser.

If required, you can generate a report from the results. By specifying the report parameters, you define which attributes of the executed sequence should also be included in the report. The report can be generated as an HTML file or as a PDF file.

Prepared Demo

If you want to compare your results with the result we wanted to achieve, you can look at the prepared demo project *Tutorialdemo03.adpx*. You can find it at `<DocumentsFolder>\Tutorial Demos`.

Further information

- For detailed information about the commands used, refer to Commands And Dialogs (AutomationDesk Basic Practices 🕮).
- For detailed information about the automation blocks used, refer to Automation Blocks (AutomationDesk Basic Practices 🕮).
- For information about the basic principles of AutomationDesk and a detailed description of executing automation sequences, refer to Executing Automation Sequences (AutomationDesk Basic Practices 🕮).

What's next

The next lesson (refer to Lesson 4: Creating Custom Libraries on page 89) shows you how to create custom libraries. If it is not your job to create custom libraries, you can skip that lesson and continue with the following one, where working with custom libraries is explained. Refer to Lesson 5: Working With the Custom Library on page 103.

Lesson 4: Creating Custom Libraries

Introduction	You learn how to save automation tasks as reusable library elements.

Introduction to Lesson 4

Task to be performed	This lesson describes how to make elements reusable that you created in the previous lessons. For more information, refer to Example Task for Using the Basic Elements of AutomationDesk on page 36.
Before you begin	You have to know how to edit an automation sequence (refer to Lesson 1: Creating Your First Project on page 41) and how to parameterize an automation sequence (refer to Lesson 2: Parameterizing the Automation Sequence on page 57).
What you will learn	AutomationDesk allows automation tasks to be saved as reusable library elements. ▪ You will extract a single automation block and a subsequence from the Tutorial demo project, and save them as automation templates in your custom library. ▪ You will also learn how to save a complete automation sequence as a automation template in a library. ▪ You will define the behavior of the specified data objects and descriptions. ▪ You will also learn how to make the automation templates available to your colleagues. **Tip** If it is not your job to create reusable elements, you can skip this lesson.

Duration

Working through this lesson will take you about 10 minutes.

Summary

Refer to Results of Lesson 4 on page 102.

Starting point

The starting point of this lesson is the AutomationDesk project you created in lesson 3.

If you have not created your own AutomationDesk project as shown in the previous lessons, you can use the prepared demo project *Tutorialdemo03.adpx*. You can find it at `<DocumentsFolder>\Tutorial Demos`.

Now you can start with the first step, see below.

Related topics

Basics

Step 1: How to Prepare Creating Library Elements

Introduction

In this step, you prepare elements of your project to use them as prototypes for new automation templates in your custom library.

One way to create templates is to drag these automation blocks, subsequences, or entire sequences from your project to a custom library. To optimize your work, you can place the Library Bowser next to the Project Manager or the Sequence Hierarchy Browser. For this you change the state of the pane. By default, the Library Browser pane is in the docking state. If you change the pane state to floating, you can place the Library Browser beside the Sequence Hierarchy Browser.

Part 1

To set a pane to docking or floating state

1. Double-click the title bar of the pane.
2. Drag the Library Browser pane beside the Sequence Hierarchy Browser.

Continue with next part

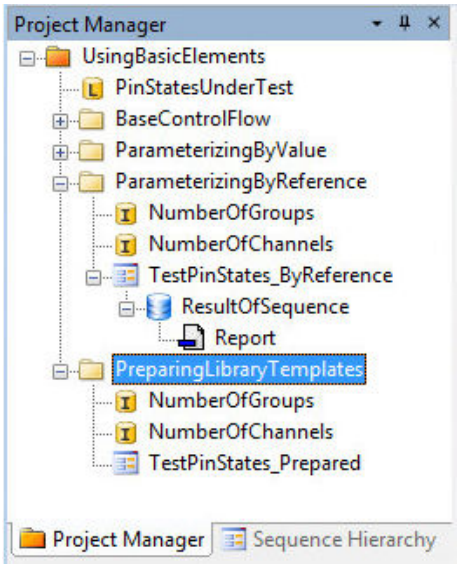
You optimized your work by placing the Library Browser next to the Sequence Hierarchy Browser.

In the next part, you create a serial block in your sequence that is used later as a subsequence template. It gathers the initial activities that are performed before the start of the actual output channel analysis.

Part 2

To create a Serial block with initial test activities

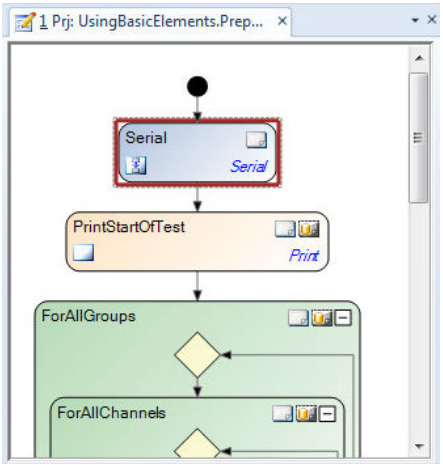
1 In the Project Manager, duplicate the ParameterizingByReference folder and rename the new folder to `PreparingLibraryTemplates` and the contained sequence to `TestPinStates_Prepared`.



2 In the Project Manager, double-click the TestPinStates_Prepared sequence to open it in the Sequence Builder.

3 In the Library Browser, select the Serial block from the Control Flows folder in the Main Library and drag it to the Sequence Builder. Drop the block to the top of the sequence's control control flow.

A new Serial block is added to the sequence.

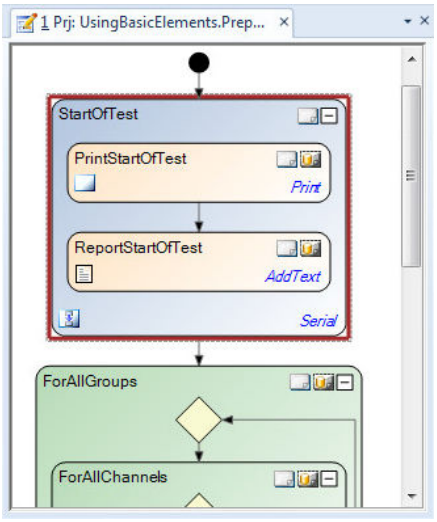


4 Rename the new block to `StartOfTest`.

5 Select the PrintStartOfTest block and move it to the StartOfTest block.

6 From the Report library, drag an AddText block to the StartOfTest block.
A new AddText block is added to the Serial block.

7 Rename the new block to `ReportStartOfTest`.



8 In the Data Object Editor, double-click the Text data object of the ReportStartOfTest block to open the Value Editor.

9 In the Value Editor, enter the following Python expression and press **OK**:

```
"Pin state test %s" % (_AD_.PinStatesUnderTest)
```

The contents of the list with the actual pin states `PinStatesUnderTest` is accessed via the _AD_ alias.

What's next

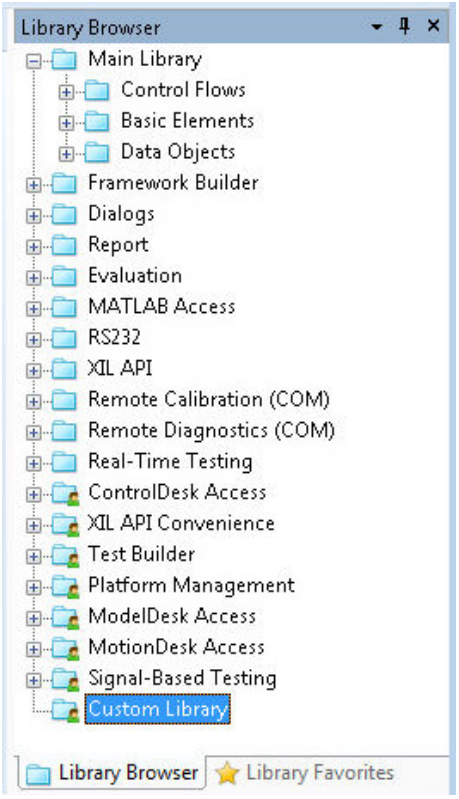
You prepared a Serial block in your project to use it as a prototype for building a subsequence template.

With the next step, you create the custom library to which you will add the new template.

Step 2: How to Create a New Custom Library

Introduction

The Library Bowser contains a custom library node by default.



You can fill the default custom library with your custom automation blocks and sequences, or you can create additional custom libraries to improve organization.

In this step, we create a new custom library, fill it with a new library folder, and save the changes.

Part 1

To create a new custom library

1. On the Library ribbon, click Custom Library - New to open the Create Custom Library dialog.
2. Navigate to your working folder, for example, `<DocumentsFolder>\MyWorkingFolder`.
3. Type `UsingBasicElementsLib` in the File name edit field and click Create.
   The new custom library is displayed in the Library Browser.

Continue with next part

Now there is a new custom library named UsingBasicElementsLib in the Library Browser.

To create library folders for your new custom library, continue with the next part.

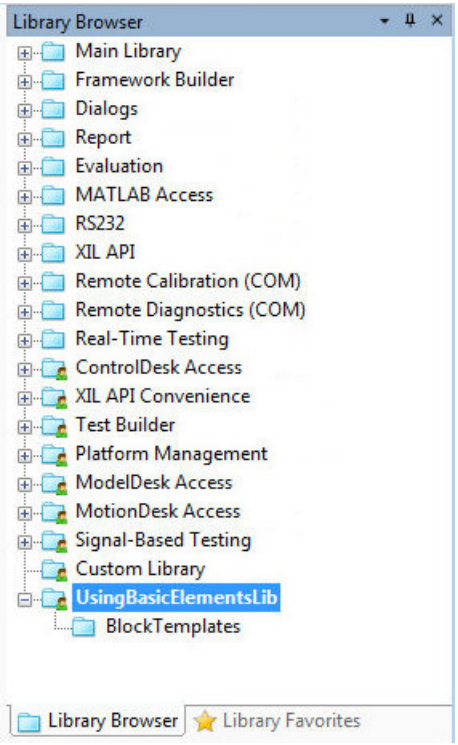
Part 2

To create library folders in the new custom library

1 Right-click the UsingBasicElementsLib node and select New Library Folder from the context menu.

A new library folder named LibraryFolder is displayed in the Library Browser.

2 Select the new library folder, press `F2`, and rename it to `BlockTemplates`.



Continue with next part

You created a new library folder.

Note

All new elements of the custom library nodes, for example, custom library folders or custom automation blocks, are inserted only temporarily if you do not save them.
When you exit AutomationDesk you are prompted to save modifications of the custom libraries.

In the next part, you save the new library.

Part 3

To save a custom library

1 In the Library Browser, from the custom library's context menu select Save.

What's next

You created a new custom library.

Now you can add reusable templates to it. The next step shows how to do this for a parameterized automation block.

Step 3: How to Create an Automation Block Template

Introduction

The created project contains blocks that can be reused for similar automation tasks.

One of these is the *PrintEndOfTest* block, which can be used as a standard action to indicate the end of a test. If you save this block in your custom library, you can use it in other sequences without parameterizing it each time.

Method

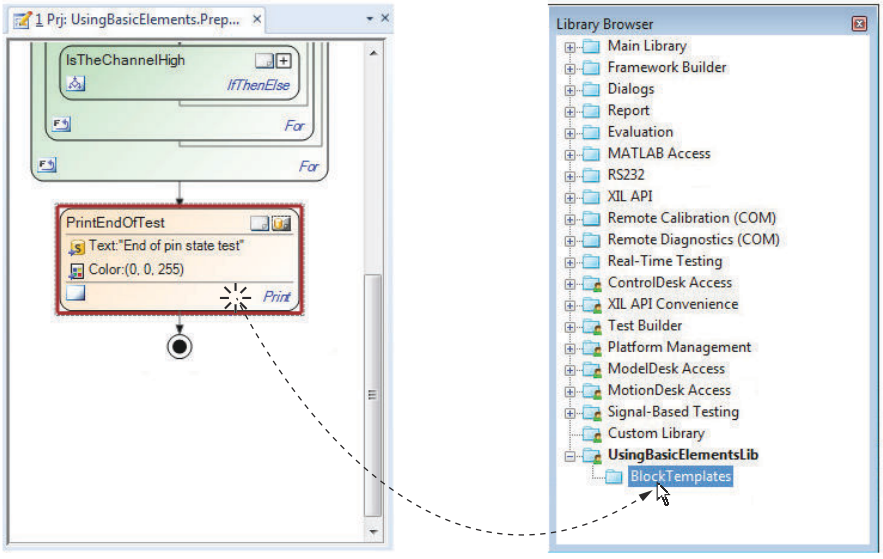
To create an automation block template

1 Open the TestPinStates_Prepared sequence in the Sequence Builder.

2 In the Sequence Builder, select the PrintEndOfTest block.

The selected block is marked with a red frame.

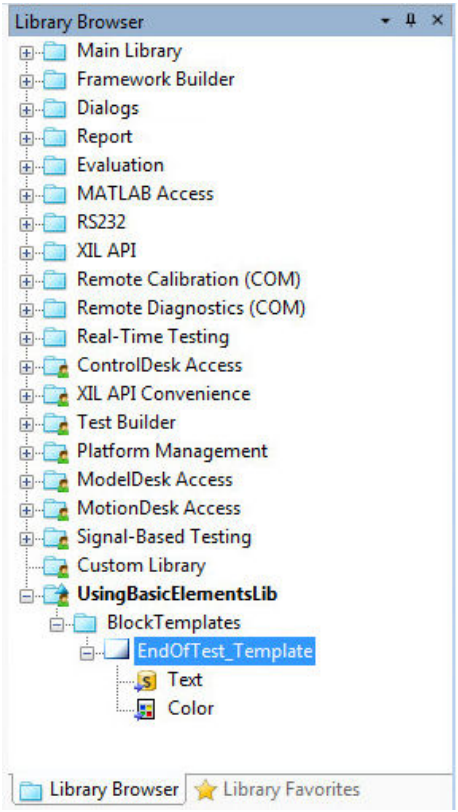
3 Drag the block from the Sequence Builder to the BlockTemplates library folder in the UsingBasicElementsLib library.



The new PrintEndOfTest template is displayed in the library.

4 In the Library Browser, rename the new template to `EndOfTest_Template`.

This is not mandatory, but descriptive naming makes it easier to distinguish templates from their prototypes.



Result

If you select the *EndOfTest_Template* block template, you can see that its parameterization is the same as the parameterization of the *PrintEndOfTest* prototype block.

What's next

You created an automation block template.

In the next step you create a template for a subsequence.

Step 4: How to Create a Subsequence Template

Introduction

If you can identify reusable subsequences in your automation task, it is possible to save them in your custom library folder. The new subsequence template contains each of the specified subordinated automation blocks.

In your project, the *StartOfTest* Serial block can be used as a standard action to indicate the start of a test. If you save this subsequence in your custom library, you can use it in other sequences without implementing it each time.

Method

To create a subsequence template

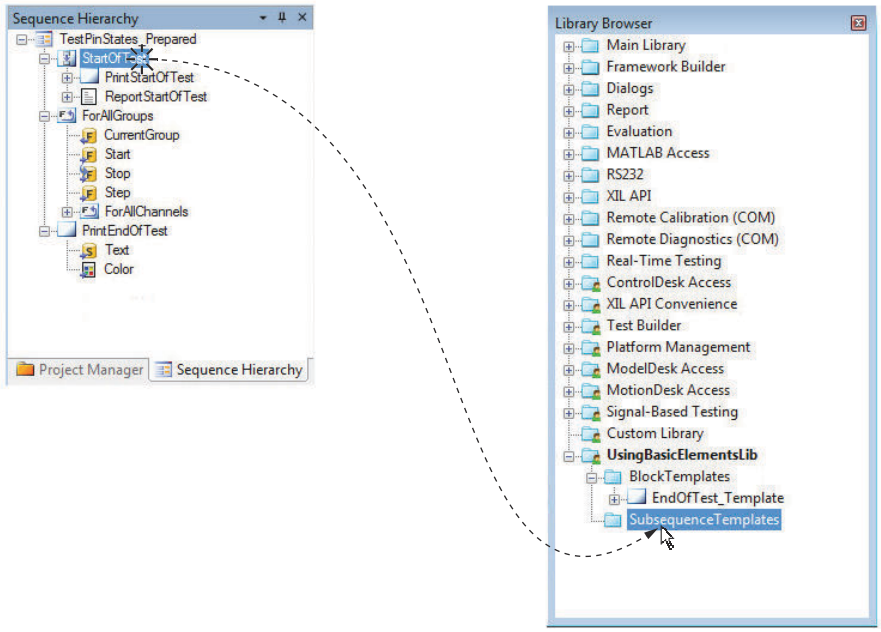
1. Open the TestPinStates_Prepared sequence in the Sequence Builder.

   If not already present, the TestPinStates_Prepared sequence is displayed in the Sequence Hierarchy Browser.

2. In the Library Browser, create a library folder in your custom library and rename it to `SubsequenceTemplates`.

3. In the Sequence Hierarchy Browser, select the StartOfTest Serial block.

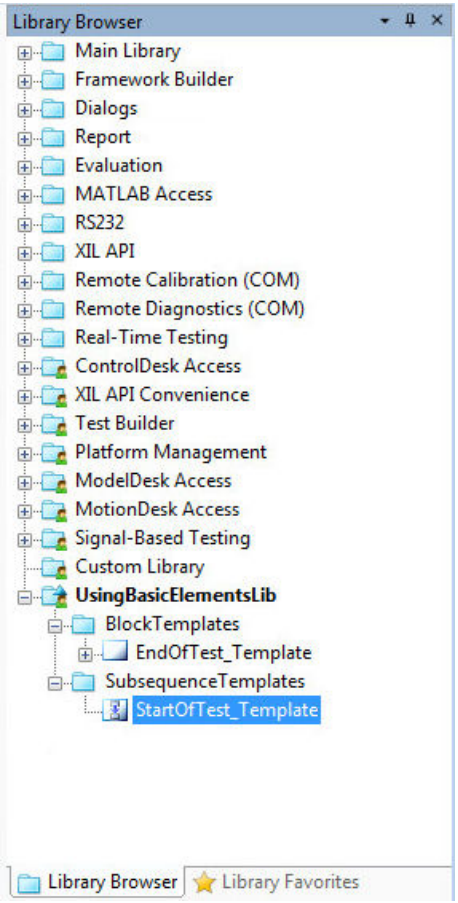
4. Drag the block from the Sequence Hierarchy Browser to the SubsequenceTemplates library folder in the UsingBasicElementsLib library.



   The new StartOfTest subsequence template is displayed in the library.

5. In the Library Browser, rename the new template to `StartOfTest_Template`.

This is not mandatory, but descriptive naming makes it easier to distinguish templates from their prototypes.



Result

If you open the *StartOfTest_Template* subsequence template, you can see that its parameterization is the same as the parameterization of the *PrintStartOfTest* prototype serial.

What's next

You can also create templates from complete sequences in the next step.

Step 5: How to Create a Sequence Template

Introduction

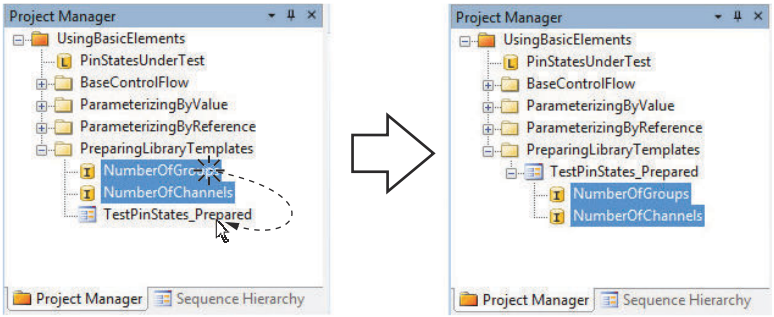
For a reusable sequence template, it is helpful, to gather the data objects that are required for parameterization in the template's interface. In your project, this applies to *NumberOfGroups* and *NumberOfChannels*.

Method

To create a sequence template

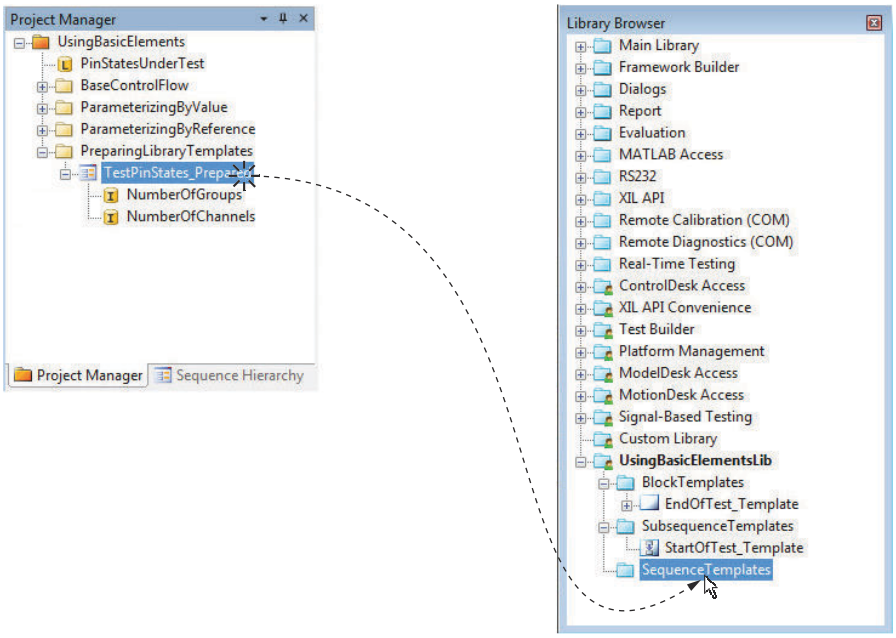
1. In the Project Manager, keep the **Ctrl** button pressed while you select the data objects NumberOfGroups and NumberOfChannels in the PreparingLibraryTemplates folder.
2. Drag the multi-selected data objects to the TestPinStates_Prepared sequence in the Project Manager.

   This changes the location of the data objects in the project's hierarchy.



   The data objects now represent the interface of the sequence.

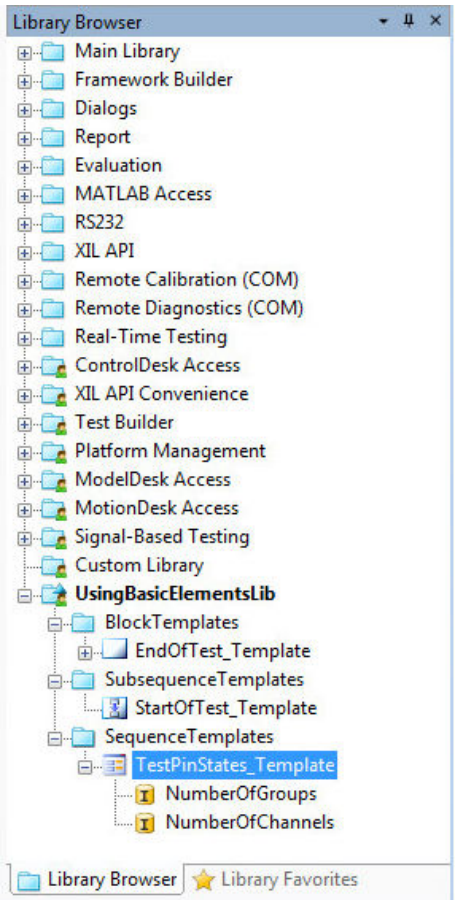
3. In the Library Browser, create a library folder in your custom library and rename it to **SequenceTemplates**.
4. In the Project Manager, select the TestPinStates_Prepared sequence and drag it to the SequenceTemplates library folder in the UsingBasicElementsLib library.



   The new TestPinStates_Prepared sequence template is displayed in the library.

5. In the Library Browser, rename the new template to **TestPinStates_Template**.

This is not mandatory, but descriptive naming makes it easier to distinguish templates from their prototypes.



6 Right-click the UsingBasicElementsLib custom library node and choose Save from the context menu.

Your modifications of the custom library are saved.

Result

If you open the TestPinStates_Template sequence template, you will see that both set and referenced data objects remain in the template sequence.

Note

All new elements of the custom library nodes, for example, custom library folders or custom automation blocks, are inserted only temporarily if you do not save them.

When you exit AutomationDesk you are prompted to save modifications of the custom libraries.

What's next

The next step shows how to make your custom library available to your colleagues.

Step 6: How to Make a Custom Library Available to Others

Introduction

Once created, custom libraries can be reused by others. If you work on a network, you can save your custom library to a network folder.

Method

To make a custom library available to others

1. Right-click the custom library node you want to save and choose Save as from the context menu.
2. Choose a network folder, enter `MyLibForCommonUse` in the File name edit field, and click Save.

   The MyLibForCommonUse custom library is now active in your Library Browser. The library is locked by you.
3. Right-click the custom library node and choose Close to give another user full access to the new custom library.
4. On the Library ribbon, click Custom Library - Open Custom Library to open the dialog.
5. Navigate to your working folder, select the UsingBasicElementsLib.adlx file and click Open.

   The UsingBasicElementsLib custom library that you created in this lesson is displayed in the Library Bowser again.

Result

You created the *MyLibForCommonUse* custom library that can be opened by others. You reopened the UsingBasicElementsLib Custom Library to proceed with it in the next lesson.

If you want to protect the copied custom library against editing, you can set the read-only attribute in the file system for all files that belong to the custom library.

There are further options for working with custom libraries:

- You can set your custom library under version control.
- You can export and import custom libraries as archives of ZIP file type.
- You can export and import custom libraries in XML format.

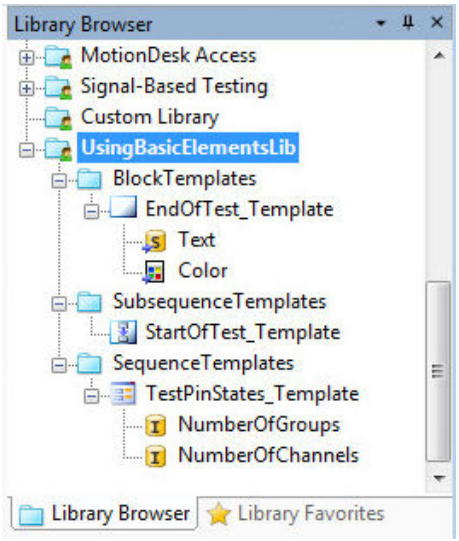
What's next

For a summary of the lesson's exercises, refer to Results of Lesson 4 on page 102.

Results of Lesson 4

Results

If you have performed all of the previous steps, your Library Browser contains a custom library called **UsingBasicElementsLib**. In its library folders the custom library provides an automation block template, a subsequence template, and a complete sequence as a template.



We have learned that you can drag automation elements from the Sequence Hierarchy Browser to a custom library folder of the Library Browser's custom library node.

- You can maintain specified properties and data objects (directly set and referenced).
- You can modify the properties of the automation templates directly.
- You must save the created custom library explicitly to make it available for further AutomationDesk sessions.
- If you want to make your custom library available to your colleagues, you must provide a copy of certain files, for example, on a network drive.

Further information

- For detailed information about the commands used, refer to Commands And Dialogs (AutomationDesk Basic Practices 🕮).
- For detailed information about the automation blocks used, refer to Automation Blocks (AutomationDesk Basic Practices 🕮).
- For information about the basic principles of AutomationDesk and a detailed description of creating user libraries, refer to Working with Custom Libraries (AutomationDesk Basic Practices 🕮).

What's next

The next lesson (refer to Lesson 5: Working With the Custom Library on page 103) shows you how to build sequences using this custom library.

Lesson 5: Working With the Custom Library

Introduction	You learn how to create a new sequence by using elements of a custom library.

Introduction to Lesson 5

Task to be performed	This lesson describes how to implement the basic task by using the reusable elements that you created in the previous lesson. For more information, refer to Example Task for Using the Basic Elements of AutomationDesk on page 36.
Before you begin	You have to know how to edit an automation sequence (refer to Lesson 1: Creating Your First Project on page 41) and parameterize an automation sequence (refer to Lesson 2: Parameterizing the Automation Sequence on page 57).
What you will learn	AutomationDesk enables you to save your standard automation tasks in custom libraries and reuse them in similar projects or variants of a sequence. In this lesson, the tutorial demo project will be modified. The task of this lesson is to analyze an electronic device with 16 digital outputs instead of 8. You can use the same sequence but with other values for the project-specific data objects. ▪ You will learn to open a custom library in AutomationDesk. ▪ You will create a new sequence using elements of this custom library.
Duration	Working through this lesson will take you about 15 minutes.
Summary	Refer to Results of Lesson 5 on page 112.

Starting point

The starting point of this lesson is the AutomationDesk project you created in lesson 3.

If you have not created your own AutomationDesk project as shown in the previous lessons, you can use the prepared demo project *Tutorialdemo03.adpx*. You can find it at `<DocumentsFolder>\Tutorial Demos`.

Now you can start with the first step , see below.

Step 1: How to Open a Prepared Custom Library

Introduction

Tip

If you created your own custom library in lesson 4, you can skip this step.

If the UsingBasicElementsLib custom library is not open yet, you can copy the library that is included in the AutomationDesk demo projects to your working folder.

First, open the prepared library.

Part 1

To open a prepared custom library

1. On the Library ribbon, click Custom Library - Open Custom Library to open the dialog.
2. Navigate to `<DocumentsFolder>\Tutorial Demos`.
3. Select the UsingBasicElementsLib.adlx file and click Open.

Continue with next part

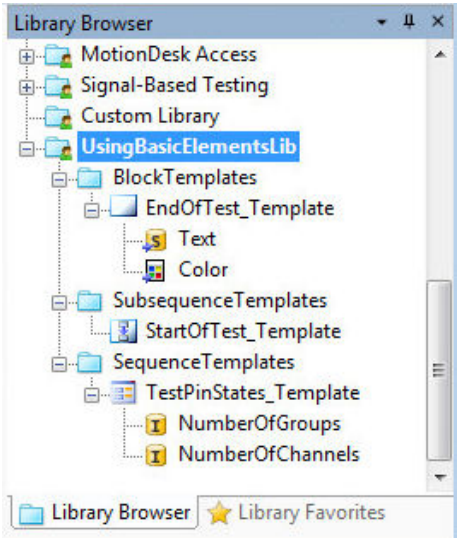
You opened a prepared custom library in the Library Browser. Now you can save it to your working folder. This avoids that you overwrite the demo version of the custom library when you proceed with this lesson.

Part 2

To save the prepared custom library for working

1. In the Library Browser, right-click UsingBasicElementsLib to open the context menu and choose Save as to open the dialog.
2. Navigate to your working folder, for example, `<DocumentsFolder>\MyWorkingFolder`.
3. Type `UsingBasicElementsLib` in the File name edit field and click Save.

The Library Browser now contains the custom library elements that you created in the previous lesson.



What's next

Now you can use the library's templates in your project. The next step shows how to use block templates in your sequences.

Step 2: How to Use a Block Template and a Subsequence Template

Introduction

In this step, you reuse the block and the subsequence that are provided by the custom library as templates. You can integrate them as linked instances in your sequences by dragging their library elements to the related position in your control flow.

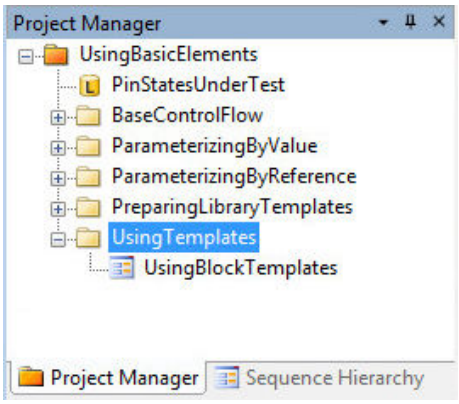
This way, you can implement similar sequences quickly and easily by assembling them from reusable templates.

Method

To use block templates and subsequence templates

1 In the Project Manager, create a new folder and rename it to `UsingTemplates`.

2 In the new folder, create a new sequence and rename it to `UsingBlockTemplates`.



3 Double-click the new sequence to open it in the Sequence Builder.

4 From the UsingBasicElementsLib custom library, drag StartOfTest_Template to the Sequence Builder

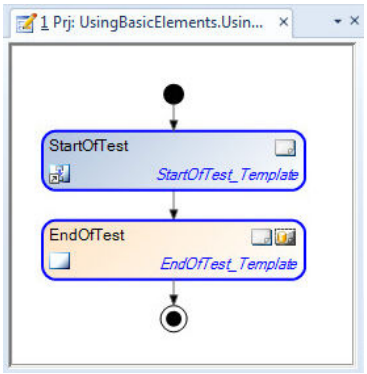
A new block of the type StartOfTest_Template is created.

5 Rename the new block to StartOfTest.

6 Drag an EndOfTest_Template to the Sequence Builder.

A new block of the type EndOfTest_Template is created.

7 Rename the new block to EndOfTest.



Result

When you execute this sequence, a start message and an end message is printed in blue to the Output Viewer.

When you generate a report, the list that represents the output pins of the device is included.

What's next

You used block templates and subsequence templates in your sequence.

In the next step, you learn how to use a template that provides a complete sequence.

Step 3: How to Use Sequence Templates

Introduction

In this step, you reuse the sequence that you implemented in the previous lessons for analyzing an electronic device with 8-bit digital outputs.

The template of this sequence provides data objects to parameterize the number of groups and channels. With these, an instance of its sequence template can also be used to analyze other digital interfaces, for example, one with a 16-bit output.

In the sequence template, the representation of the digital output is the two-dimensional list named *PinStatesUnderTest*, which is addressed via the *_AD_* alias. To specify a 16-bit variant of this list, a project-specific data object with the same name is needed. It is located in the same folder as the new instance of the sequence.

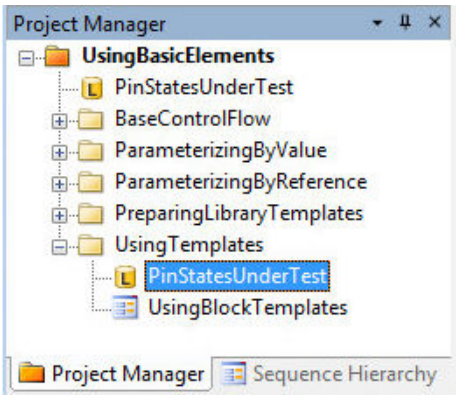
Part 1

To create the project-specific list that represents a 16-bit output

1. In the Project Manager, from the context menu of the UsingTemplates folder, select **New Data Object - List** to create a new data object of list type.

   The new data object is displayed in the folder.

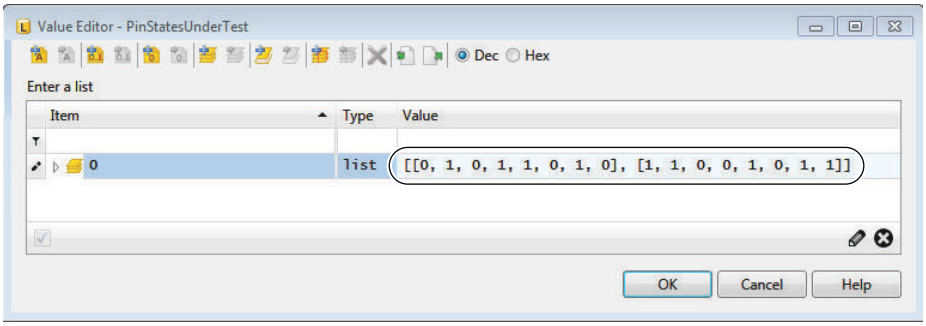
2. Rename the new data object to `PinStatesUnderTest`.



3. Double-click the **PinStatesUnderTest** data object to open it in the Value Editor.

4. In the **Value** edit field, replace the `[]` with the following two-dimensional list:

```
[[0, 1, 0, 1, 1, 0, 1, 0], [1, 1, 0, 0, 1, 0, 1, 1]]
```



5 Click **OK** to close the Value Editor.

Continue with the next part

You created the list that represents the 16-bit digital output.

In the next part, you create the new sequence for analyzing the list and parametrize the new sequence with the new number of groups and channels.

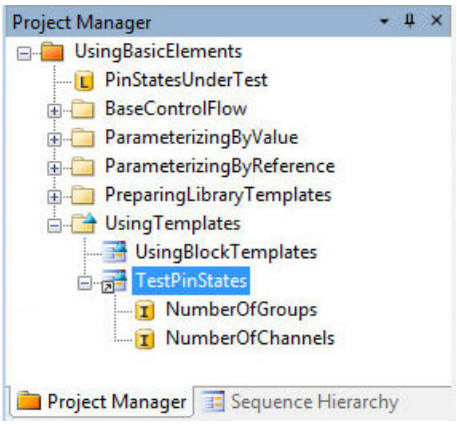
Part 2

To create a new sequence using a sequence template

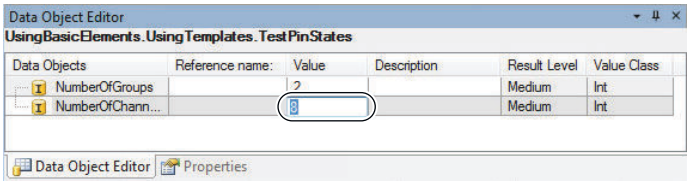
1 From the UsingBasicElementsLib custom library, drag the TestPinStates_Template sequence template to the UsingTemplates folder in the Project Manager.

A new sequence with the same name as its sequence template is added to the folder. The icon extension of the sequence indicates that it is linked dynamically to its template in the Custom Library.

2 Rename the new sequence to `TestPinStates`.



3 In the Data Object Editor, change the Value of the NumberOfChannels data object to **8**.



Executing the sequence

When you execute this sequence, the control flow as implemented in the template is processed. Due to the parameterization of the data objects in the sequence's interface, the analysis is performed for the complete 16-bit digital output as show in the report below.

TestPinStates (Sequence)			
Name:	TestPinStates	**Start time:**	2017-01-19 10:42:54
Stop time:	2017-01-19 10:42:54	**Execution duration:**	0.278 seconds
Description:			
Result state:	Executed		

Pin state test [[0, 1, 0, 1, 1, 0, 1, 0], [1, 1, 0, 0, 1, 0, 1, 1]]

```
Group: 0 |Pin: 0 |Level:  Low |Test: Failed
Group: 0 |Pin: 1 |Level: High |Test: OK
Group: 0 |Pin: 2 |Level:  Low |Test: Failed
Group: 0 |Pin: 3 |Level: High |Test: OK
Group: 0 |Pin: 4 |Level: High |Test: OK
Group: 0 |Pin: 5 |Level:  Low |Test: Failed
Group: 0 |Pin: 6 |Level: High |Test: OK
Group: 0 |Pin: 7 |Level:  Low |Test: Failed
Group: 1 |Pin: 0 |Level: High |Test: OK
Group: 1 |Pin: 1 |Level: High |Test: OK
Group: 1 |Pin: 2 |Level:  Low |Test: Failed
Group: 1 |Pin: 3 |Level:  Low |Test: Failed
Group: 1 |Pin: 4 |Level: High |Test: OK
Group: 1 |Pin: 5 |Level:  Low |Test: Failed
Group: 1 |Pin: 6 |Level: High |Test: OK
Group: 1 |Pin: 7 |Level: High |Test: OK
```

What's next

You created and parameterized an instance of a sequence template.

Until the automation blocks of a sequence are linked to the corresponding library element, you can synchronize the automation blocks after modifying library elements. The next step will describe how to do this.

Step 4: How to Synchronize a Sequence Created With Custom Library Elements

Introduction

By default, the linked instances of templates that you use in your project are dynamically linked to the related templates in the custom library. Modifications in the templates' control flows take effect immediately. Synchronization is necessary only if a template's interface was modified.

In larger projects, it can be helpful to locate the linked template in its custom library.

Synchronization ranges

Synchronization can be started on different hierarchy levels in the Project Manager. For example, if you execute the **Synchronize** command on a folder element, all subordinated sequences update their custom templates according to their library links.

The **Synchronize** command is also available in the Sequence Hierarchy Browser and in the Sequence Builder.

Part 1

To locate the linked template of an element

1. Open the UsingBlockTemplates sequence in the Sequence Builder.
2. Select the StartOfTest block.
   The block is marked with a red frame.
3. From the block's context menu, select Locate Element in Library.
   The StartOfTest_Template is highlighted in the Library Browser.

Continue with next part

You located the linked template of the *StartOfTest* block.

To show the effects of the synchronization mechanism, you can edit the subsequence template in the next part.

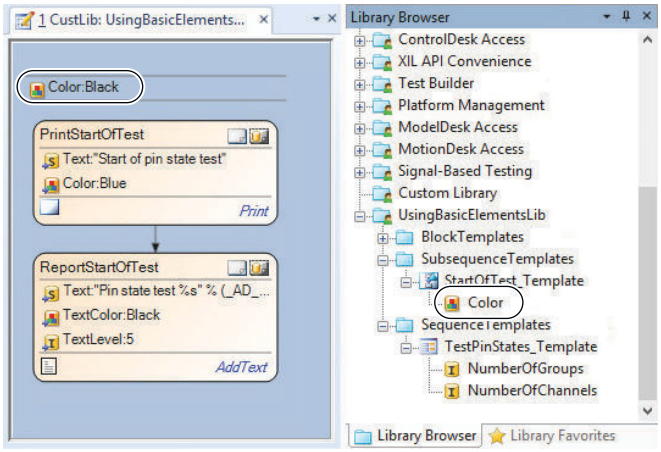
A *Color* data object is added to the subsequence's interface to parameterize the color of the message that is printed to the Output Viewer.

Part 2

To modify the interface of the subsequence template

1. In the Library Browser, double-click the StartOfTest_Template element.
   The template is opened in the Sequence Builder. To indicate that this is a template and not a sequence, the background of the Sequence Builder is colored.
2. In the Library Browser, select a Color data object from the Report library and drag it to the Sequence Builder.

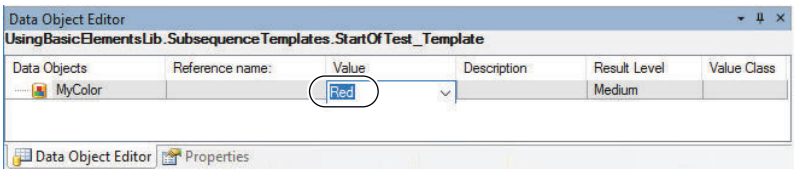
A Color data object with the default value of `(0, 0, 0)` is added to the template. It is displayed in the Sequence Builder and in the Library Browser.



The template's parameterization is displayed in the Data Object Editor.

3 In the Data Object Editor, select the Color data object and press `F2` to rename the data object to `MyColor`.

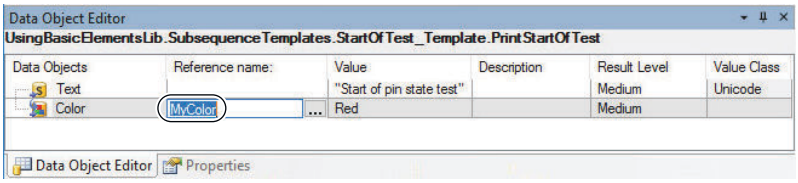
4 In the Value edit field of the MyColor data object, select Red from the provided list of colors.



5 In the Sequence Builder, select the PrintStartOfTest block.

The block is marked with a red frame. The block's parameterization is displayed in the Data Object Editor.

6 In the Data Object Editor, enter `MyColor` in the Reference name edit field of the Color data object.



A value Red is displayed in the Value column of the Color data object, which is the value of the referenced data object.

Continue with next part

Because the interface of the template is modified, an execution of the *UsingBlockTemplates* sequence would terminate with an error message. You have to synchronize the sequence with its linked templates in the custom library.

Part 3

To synchronize a sequence in the Project Manager

1. In the Project Manager, select the UsingBlockTemplates sequence.
2. On the Library ribbon, click Library Link - Synchronize to activate the update mechanism for the sequence.

What's next

You changed the interface of a subsequence template that was dynamically linked to a sequence and then you synchronized the sequence.

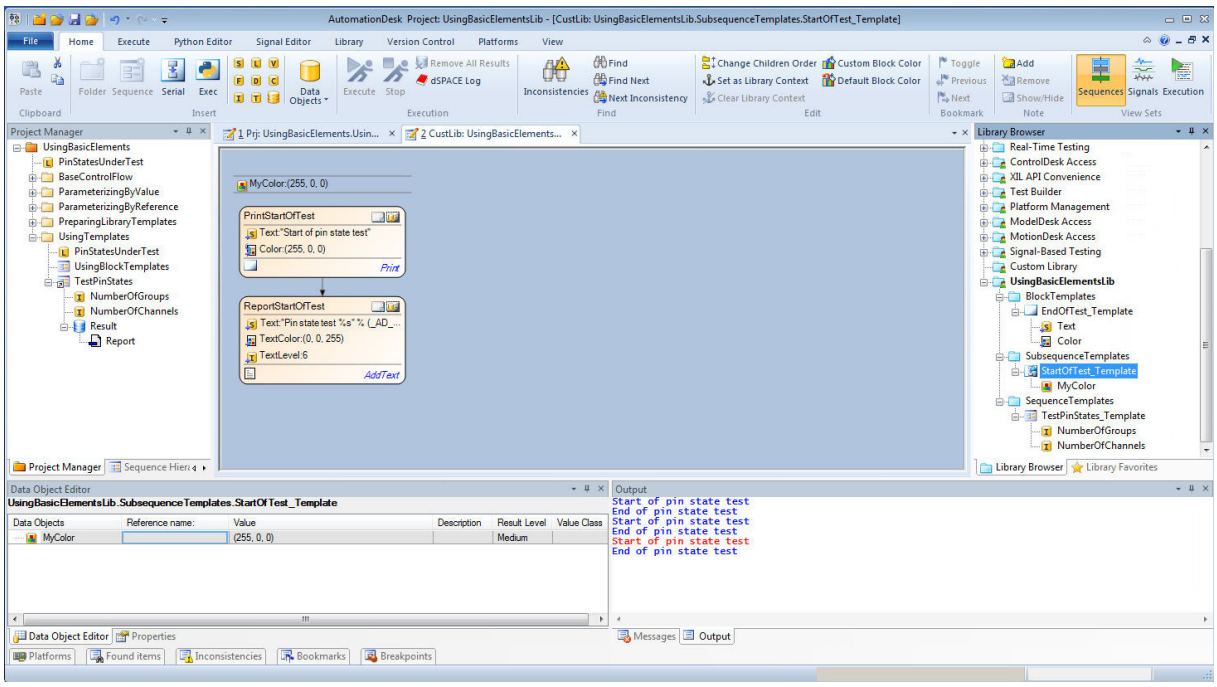
For a summary of the lesson's exercises, refer to Results of Lesson 5 on page 112.

Results of Lesson 5

Results

If you have performed all of the previous steps, the Project Manager contains a new folder called *UsingTemplates*. It contains a 16-bit variant of the *PinStatesUnderTest* list, a sequence that uses block templates and a parameterized instance of a the *TestPinStates_Template*.

If you execute the *UsingBlockTemplates* sequence after the synchronization, the start message of the test is printed in red.



If you are working with custom libraries, it is necessary to know about the update mechanism. Until the sequence elements are linked to the corresponding custom library elements, you can update your sequence with modified interfaces.

- Synchronization covers new and removed data objects of a library element, and modified subsequences.
- Specified properties of the block remain unchanged.

Prepared demo

If there were problems in editing your own AutomationDesk project, or if you want to compare your results with the result we wanted to achieve, you can look at the prepared demo project *Tutorialdemo05.adpx*. You can find it at `<DocumentsFolder>\Tutorial Demos`.

Further information

- For detailed information about the commands used, refer to Commands And Dialogs (AutomationDesk Basic Practices 🕮).
- For detailed information about the automation blocks used, refer to Automation Blocks (AutomationDesk Basic Practices 🕮).
- For detailed information about the static and dynamic link modes of custom library blocks, refer to Basics on Link Modes (AutomationDesk Basic Practices 🕮).
- For information about the basic principles of AutomationDesk and a detailed description of applying user-defined library elements, refer to Working with Custom Libraries (AutomationDesk Basic Practices 🕮).

What's next

The next lesson (refer to Lesson 6: Using Python Scripts on page 115) gives you an overview of how to use the Python programming language in AutomationDesk.

Lesson 6: Using Python Scripts

Introduction	You learn how to enhance AutomationDesk by using Python scripts.

Introduction to Lesson 6

Task to be performed	This lesson describes how to perform the example task for enhancing AutomationDesk using Python code. For more information, refer to Example Task for Using Python in AutomationDesk on page 38.
Before you begin	You have to know how to edit an automation sequence (refer to Lesson 1: Creating Your First Project on page 41) and to parameterize an automation sequence (refer to Lesson 2: Parameterizing the Automation Sequence on page 57). To save custom automation blocks as a library element, you should have read Lesson 4: Creating Custom Libraries on page 89. For more information on Python, refer to the Python website http://www.python.org.
What you will learn	AutomationDesk's libraries are based on the Python programming language. Because of this, it is easy to use Python functions or Python modules in AutomationDesk. You will learn how to integrate Python scripts with and without the need to exchange data between AutomationDesk and the Python workspace. You learn also to create your own automation block for an existing Python function. AutomationDesk provides two blocks that you can use for these purposes.
Duration	Working through this lesson will take you about 30 minutes.

Summary

Refer to Results of Lesson 6 on page 128.

Starting point

This lesson is independent of the previous work.

You can start with creating a new AutomationDesk project in the first step.

Related topics

Basics

Step 1: How to Use Python Without Accessing Data Objects

Introduction

The task in this step is to print a header with information about the current test run to the Output Viewer. For this, no access to a project-specific data object is required. AutomationDesk provides the automation blocks **Exec** and **ExecFile**.

For each **Exec** block and **ExecFile** block, you can specify which Python namespace is used for name resolution, for example, for variable names:

- If the *local* namespace is used, variables are accessible only in their specific blocks.

  It is recommended to use the local namespace whenever possible. This avoids unwanted impact on other elements of your project.

  The memory of allocated local variables is freed automatically when the block's execution is finished.
- If the *global* namespace is used, variables are accessible across blocks and projects in the AutomationDesk session.

  Once a global variable is set, you can access it in all currently opened AutomationDesk projects. Ensure that you have initialized the variable before you use its value.

  To avoid memory shortage in large projects, you must free the memory of allocated global variables that are no longer required by setting them to *None*. For more information on what to consider when working with objects in the global namespace, refer to UsingCOMInAutomationDeskApplicationNote.pdf.

  Unlike AutomationDesk data objects, the values of the global variables are not stored when you save a project.

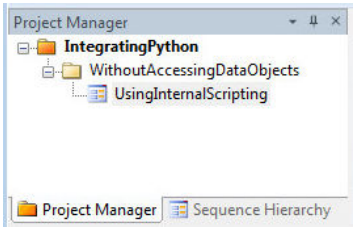
Using an Exec block

An **Exec** block lets you specify Python code internally in AutomationDesk.

Method 1

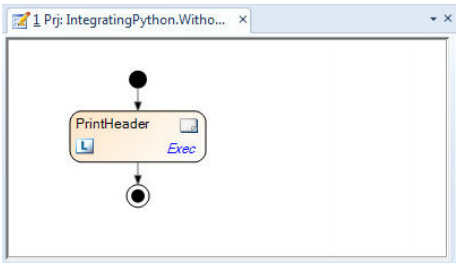
To use Python code via an Exec block

1. In the Project Manager, create a new AutomationDesk project named `IntegratingPython`.
2. Add a new folder to the project and rename it to WithoutAccessingDataObjects.
3. Add a new sequence to the folder and rename it to UsingInternalScripting.



4. Double-click the UsingInternalScripting sequence to open it in the Sequence Builder.
5. On the Home ribbon, click Insert - Exec to add an Exec block to the sequence and rename the block `PrintHeader`.
6. From the block's context menu, select Set Python Namespace – Local.

   The block icon changes to indicate that for its script execution the local namespace is used.



7. In the Sequence Builder, double-click the PrintHeader block.

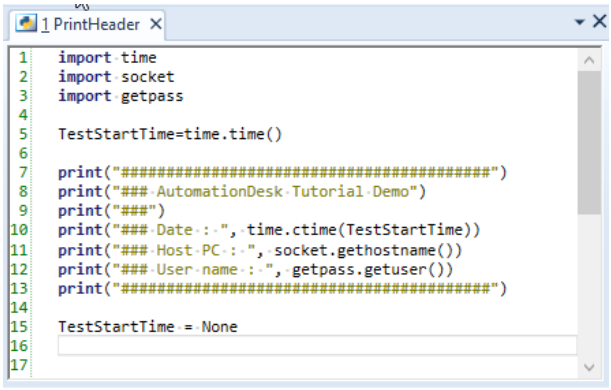
   In the working area, AutomationDesk's Python Editor opens.

   The default script contains a `pass` statement that represents the Python code.
8. In the Python Editor, replace the `pass` statement with the following Python code:

```
import time
import socket
import getpass
TestStartTime=time.time()
print("##########################################")
print("### AutomationDesk Tutorial Demo")
print("###")
print("### Date      : ", time.ctime(TestStartTime))
print("### Host PC   : ", socket.gethostname())
print("### User name : ", getpass.getuser())
print("##########################################")
TestStartTime = None
```

This script first imports the Python modules that are used later to determine the current time, the PC host name, and the login name of the user that executes the script. Then, a time stamp is assigned to a variable and a header is printed to the output. Finally, the variable set to `None`.

The Python Editor provides syntax highlighting as shown below:



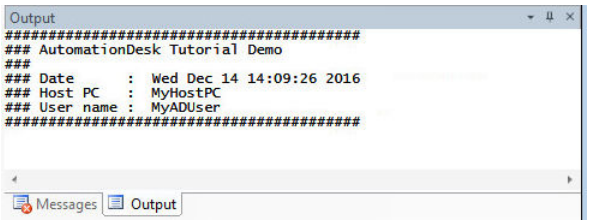
Tip

AutomationDesk provides the Python Editor ribbon that contains commands that support you when scripting Python.

9 Close the Python Editor to apply your input and save your project.

Executing the sequence

When you execute this sequence, AutomationDesk performs the Exec block's Python code. The specified header that contains a time stamp, the host PC name and the name of the current user is displayed in the Output Viewer.



Note

- The Exec block's namespace contains Python built-in modules such as *time*. The modules' definitions can be used without explicitly importing them.
- Python variables, such as *TestStartTime* in the previous example, are added to the local namespace that is valid until the block's execution terminates.

Using an ExecFile block

An ExecFile block lets you specify a PY file that is executed in AutomationDesk's namespace.

Prerequisites

Create a text file with the Python source code described above and save it as *PrintHeader.py* in your working folder

Method 2

To use an external Python file via an ExecFile block

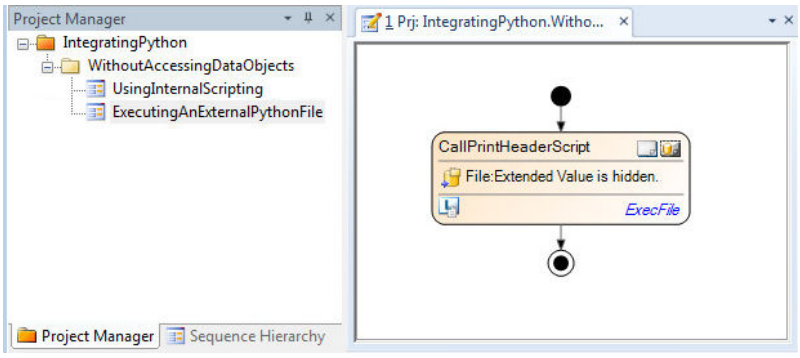
1 In the Project Manager, add a new sequence to the WithoutAccessingDataObjects folder and rename it to ExecutingAnExternalPythonFile.

2 Double-click the ExecutingAnExternalPythonFile sequence to open it in the Sequence Builder.

3 From the Main Library in the Library Browser, drag a Basic Elements - ExecFile block to the sequence and rename the block to `CallPrintHeaderScript`.

4 From the block's context menu, select Set Python Namespace – Local.

The block icon changes to indicate that for its script execution the local namespace is used.



5 Select the ExecFile block in the Sequence Builder to show its data objects in the Data Object Editor.

6 Click the ... button in the Value column to open the File dialog.

Tip

You can choose how the Python file is referenced in this dialog. This can be useful if you want to export your AutomationDesk project to another PC.

- Absolute Path means that the complete path of the Python file including the drive letter is stored.
- Relative Path means that the path of the Python file relative to the location of your AutomationDesk project is stored.

7 Select `PrintHeader.py` from your working folder and click OK to close the dialog.

8 Save the project.

Executing the sequence

When you execute this sequence, AutomationDesk executes the source code of the specified Python file. The same header as for the previous method is displayed in the Output Viewer.

Note

- Dependencies on environment variables can be avoided if you include Python functions to get environment variables. Then you can, for example, create the string for the local file name of the Python script dynamically using an Exec block.
- If you store the file name in a project-specific File data object, you can reference it from the ExecFile block.
- The Python file is not included in the AutomationDesk project. If you want the content of a Python script to be managed by AutomationDesk, you have to use Exec blocks or you integrate your Python module in a custom library as shown later.
- To make the test execution reproducible, you can add the Python file to the generated report by inserting an **AddURL** block from the Report library in your sequence. You can decide whether the link in the report refers to the original file location or to a copy of it. The copied file is created in the corresponding result folder of the AutomationDesk project.

What's next

Now you know how to use Exec and ExecFile blocks to execute Python source code without accessing AutomationDesk data objects. In the next step you will learn how to realize data exchange.

Step 2: How to Access AutomationDesk Data Objects in Python

Introduction

To parameterize sequences and automation blocks, you must use AutomationDesk data objects.

To access AutomationDesk data objects in Python, you can apply the _AD_ alias to a data object's name. This lets you address all data objects within the current object hierarchy path of the Exec block that contains the Python script. The name of a data object is resolved from lower to higher hierarchy level.

You can structure data objects by using data containers.

Your task

The task in this step is to calculate the radius of a circle specified by its area. The formula is:

$$r = \sqrt{\frac{A}{\pi}}$$

The calculated result could be needed in the following automation tasks, so it must be written to a project-specific data object.

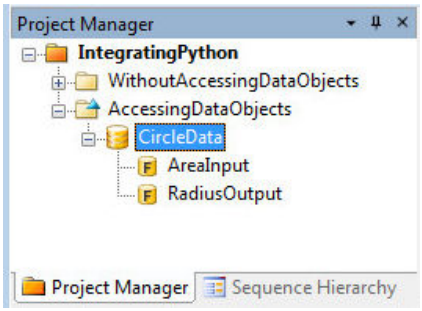
Method

To access AutomationDesk data objects in Python

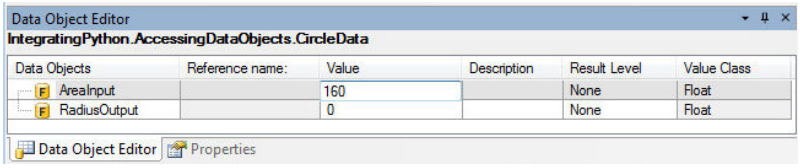
1 Add a new folder to the IntegratingPython project and rename it to AccessingDataObjects.

2 Click Home - Insert - DataContainer to add a data container to the folder and rename the data container to `CircleData`.

3 Select the CircleData data container and add two Float data objects to it. Rename them to `AreaInput` and `RadiusOutput`.



4 In the Data Object Editor, specify `160.0` as the value of the AreaInput Float data object.



Optionally, you can specify `m` as the unit of measure in the Description edit field.

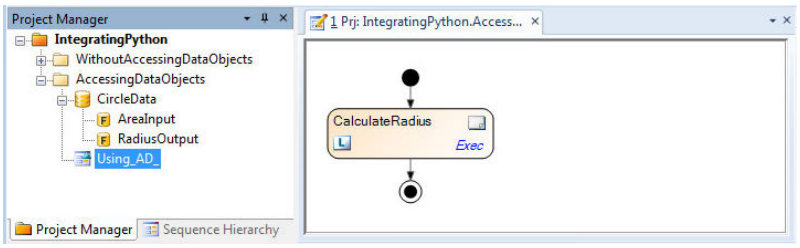
5 Add a new sequence to the AccessingDataObjects folder and rename it to Using_AD_.

6 Double-click the Using_AD_ sequence to open it in the Sequence Builder.

7 Click Home - Insert - Exec to add an Exec block to the sequence and rename the block to `CalculateRadius`.

8 From the block's context menu, select Set Python Namespace – Local.

The block icon changes to indicate that for its script execution the local namespace is used.



9 In the Sequence Builder, double-click the CalculateRadius block.

The **CalculateRadius** Python script is opened in AutomationDesk's Python Editor.

10 In the Python Editor, enter the following Python code:

```
_AD_.CircleData.RadiusOutput =
math.sqrt(_AD_.CircleData.AreaInput / math.pi)
print("Area of circle     : ", _AD_.CircleData.AreaInput)
print("=> Radius of circle: ", _AD_.CircleData.RadiusOutput)
```

This script uses the **_AD_** alias to access the AutomationDesk data objects. Within a structure of data containers, the data objects are addressed via their path.

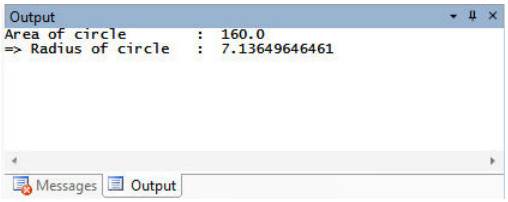
Tip

- When you enter **_AD_.** in the Python Editor you can select the data object name from the list of the currently available names.
- Alternatively, you can drag the data object's element from the Project Manager to the Python Editor to insert the related expression to the script.

11 Close the Python Editor to apply your input and save your project.

Executing the sequence

When you execute this sequence, the Exec block reads the values of the *AreaInput* project-specific data object. It calculates the related radius and writes it to the *RadiusOutput* project-specific data object. Then the *RadiusOutput* is printed to the Output Viewer.



Note

- Using the *_AD_* alias in Python is equivalent to referencing an data object in AutomtionDesk.
- In the example above, the data objects in the *CircleData* data container could be used by other sequences that you add to the *AccessingDataObjects* folder. They cannot be used in sequences in the *WithoutDataObjects* folder.
- The current values of the AutomationDesk data objects are saved together with their projects. When you reopen the projects, the last saved values are set.
- The highest level of the object hierarchy path is always the current project. You cannot share data objects among projects.

What's next

In the next step you will learn how to integrate an existing Python library in AutomationDesk.

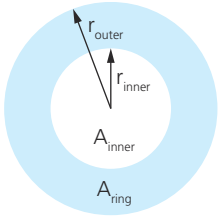
Step 3: How to use Python Modules in Custom Libraries

Introduction

To manage Python sources together with the related AutomationDesk elements, such as automation blocks, you can add Python modules and packages to a custom library. This lets you create, edit, and store the Python code by means of AutomationDesk in the custom library's folder in the file system.

Your task

The task in this step is to calculate the areas of a ring that is specified by its inner and outer radius.



The formulas are:

- For the inner area:

  $A_{inner} = r_{inner}^2 \cdot \pi$

- For the ring area:

  $A_{ring} = A_{outer} - A_{inner} = r_{outer}^2 \cdot \pi - r_{inner}^2 \cdot \pi$

The calculated result for the ring area could be required in other automation tasks, so it must be written to a project-specific data object.

Part 1

To create a Python module in a custom library

1. On the Library ribbon, click Custom Library - New to create a new custom library.

   The Create Custom Library dialog opens.

2. In the File name edit field, enter `IntegratingPythonLib` as the name of the new custom library and click Create.

   The IntegratingPythonLib library is created in your working folder and the custom library element is displayed in the Library Browser.

3. From the context menu of the IntegratingPythonLib library, select New Python module.

A new element named **module** is added to the custom library in the Library Browser.

4 Rename the new Python module to `CircleCalculations`.

5 Double-click the **CircleCalculations** module to open it in the Python Editor.

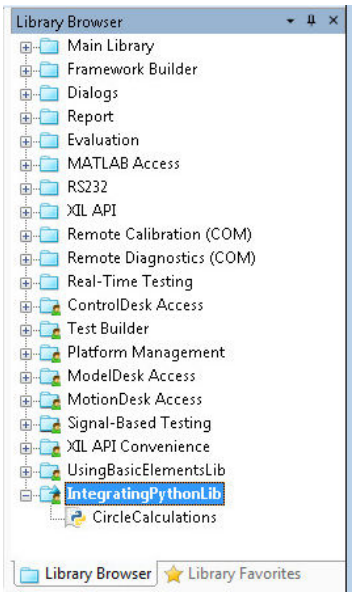
6 Enter the following Python code to implement a function that calculates the area of a circle by its radius:

```
import math
def CalculateCircleArea(r):
    f = math.pi*math.pow(r,2)
    return (f)
```

This script only defines the `CalculateCircleArea` function. The function is not executed.

7 Close the Python Editor.

8 On the **Library** ribbon, click **Custom Library - Save** to save the module together with the library.



Continue with the next part

You created a new custom library that contains a Python Module.

In the next part, you use the function that is provided by the Python module in an **Exec** block.

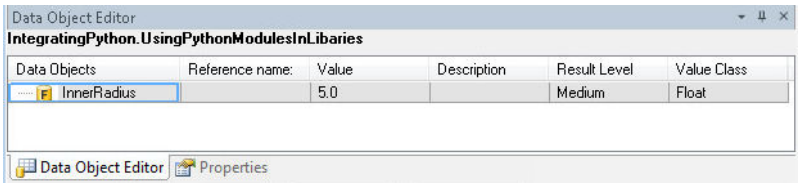
Part 2

To use functions from a Python module in an Exec block

1 In the Project Manager, add a new folder to the **IntegratingPython** project and rename it to `UsingPythonModulesInLibraries`.

2 Add a Float data object to the folder and rename it to `InnerRadius`.

3 Select the UsingPythonModulesInLibraries folder, and in the Data Object Editor, specify `5.0` as the value of the InnerRadius Float data object.



4 Add a new sequence to the UsingPythonModulesInLibraries folder and rename the sequence to `UsingPythonFunctions`.

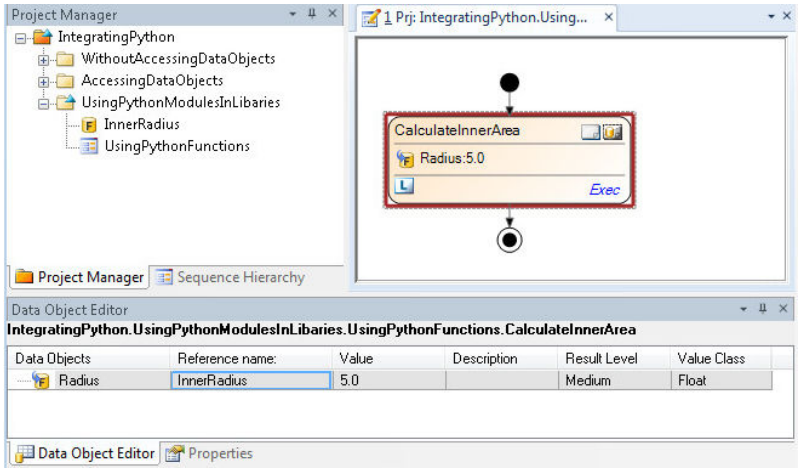
5 In the Sequence Builder, add a new Exec block to the UsingPythonFunctions sequence and rename it to `CalculateInnerArea`.

6 From the block's context menu, select Set Python Namespace – Local.

The block icon changes to indicate that for its script execution the local namespace is used.

7 Select the CalculateInnerArea Exec block and click Home - Insert - Float to add a Float data object to the block's parameters.

8 In the Data Object Editor, rename the Float data object to `Radius` and set the project-specific InnerRadius Float data object as its reference.



9 In the Sequence Builder, double-click the CalculateInnerArea block.

The CalculateInnerArea Python script is opened in the Python Editor.

10 In the Python Editor, enter the following Python code:

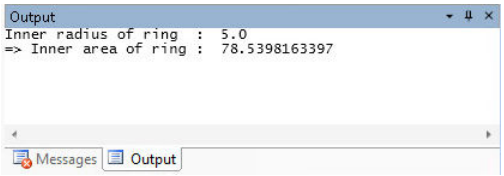
```
import CircleCalculations
InnerArea = CircleCalculations.CalculateCircleArea(_AD_.Radius)
print("Inner radius of ring  : ", _AD_.Radius)
print("=> Inner area of ring : ", InnerArea)
InnerArea = None
```

This script uses the CalculateCircleArea function of the custom library's module to calculate the inner area of the ring.

11 Close the Python Editor and save your project.

Executing the sequence

When you execute this sequence, the Exec block reads the value of the project-specific *InnerRadius* data object. It calculates the related inner area of the ring and prints the result to the Output Viewer.



Tip

- The path to Python modules that are integrated in custom libraries is automatically registered in the Python search path that is used by the `import` statements. You do not need to extend the Python search path.
- You can add additional data objects to an Exec block to use them as block parameters. This is shown by the *Radius* Float data object in the example above.

Continue with the next part

You used the function from the custom library's Python module in the Python code of an Exec block.

In the next part, you create a template for an automation block that uses the Python module.

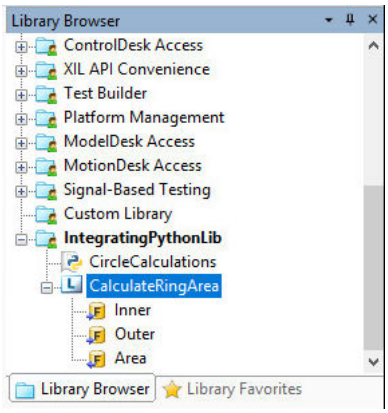
Part 3

To create a block template in the library that uses the Python module

1. In the Library Browser, select the IntegratingPythonLib library.
2. Click Home - Insert - Exec to add a new Exec block to the library and rename the new block to `CalculateRingArea`.
3. Click Home - Insert - Float three times to add three Float data objects as the block's parameters. Rename them to `Inner`, `Outer`, and `Area`.
4. From the context menu of the Area data object, select Set Data Object as Output.

   This specifies that the CalculateRingArea block provides the value of this data object as output data. This parameter of the block is read-only.
5. For each of the Inner and Outer data objects, select Set Data Object as Input from their context menu.

This specifies that the CalculateRingArea uses this parameters as input data.



6 Double-click the CalculateRingArea block to open it in the Python Editor.

7 Enter the following Python code to implement a block that calculates the area of a ring by its inner and its outer radius:

```
from CircleCalculations import CalculateCircleArea
_AD_.Area = CalculateCircleArea(_AD_.Outer) - CalculateCircleArea(_AD_.Inner)
print("Inner radius of ring  : ", _AD_.Inner)
print("Outer radius of ring  : ", _AD_.Outer)
print("=> Area of ring       : ", _AD_.Area)
```

To calculate the areas of the inner and the outer circle, the script uses the `CalculateCircleArea` function that you implemented in the CalculateCircleArea module.

8 Close the Python Editor.

9 On the Library ribbon, click Custom Library - Save to save your changes and make the block template available to be used in sequences.

Continue with the next part

You created a template for an automation block that uses the Python module.

In the next part, you use this block in a sequence.

Part 4

To use blocks from the custom library in a sequence

1 In the Project Manager, add two Float data objects to the UsingPythonModulesInLibraries folder and rename them to OuterRadius and RingArea.

2 Add a new sequence to the UsingPythonModulesInLibraries folder and rename it to `UsingLibraryBlocks`.

3 Double-click the UsingLibraryBlocks sequence to open it in the Sequence Builder.

4 From the Library Browser, drag an IntegratingPythonLib - CalculateRingArea block template to the sequence.

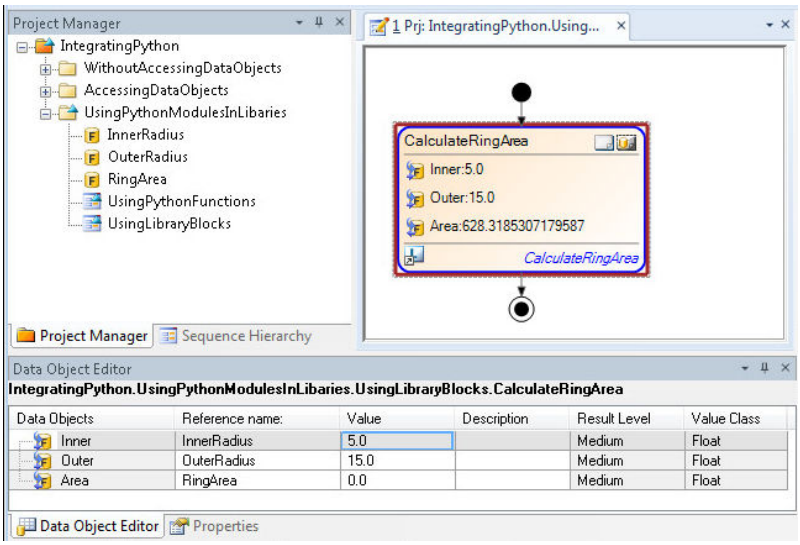
A new instance of the block template is added to the sequence.

5 In the Data Object Editor, set the project-specific OuterRadius data object as the reference for the block's Outer data object.

6 Specify `15.0` as the value of the OuterRadius data object.

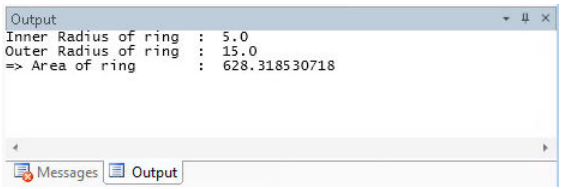
7 Specify the project-specific InnerRadius data object as the reference for the block's Inner data object.

8 Specify the project-specific RingArea data object as the reference for the block's Area data object.



Executing the sequence

When you execute this sequence, the *CalculateRingArea* block reads the values of the project-specific data object *InnerRadius* and *OuterRadius*. It calculates the ring area, writes it to the project-specific *RingArea* data object, and prints it to the Output Viewer.



What's next

For a summary of the lesson's exercises, refer to Results of Lesson 6 on page 128.

Results of Lesson 6

Results

AutomationDesk provides the following ways to integrate Python code:

- You can specify Python code via Exec blocks.
- You can specify PY files that contain Python code via ExecFile blocks.
- You can add Python modules and packages to your custom libraries.

The values of project-specific and automation blocks' data objects can be written and read in Python using the `_AD_` alias. You can add data objects to a custom automation block to adapt the number of input and output variables to the new block's functionality.

Tip

The custom libraries that are shipped with AutomationDesk, such as *XIL API Convenience*, are implemented in the same way as shown in *Step 3* of this lesson:

- Each custom library provides sources in an integrated Python module or Python package. You can view their sources in the Python Editor in read-only mode.
- The templates for data objects and automation blocks of those custom libraries can be instantiated in sequences.

  You can use the related functions in the Python code of your Exec blocks. For more information, refer to the *Access via Exec block* paragraph of a block's reference.

Programming hints

When you execute a sequence, the blocks will be executed in one or more threads according to the implemented serial or parallel control flow. Objects that have been created in a thread must be deleted after execution. Normally, this is done automatically using the automation blocks from the AutomationDesk libraries. However, if you implement your task using Exec blocks, you have to manage the object handling in your own Python code. You can do this by using a TryFinally automation block as a frame for your custom Exec blocks, or by using the `try` statement in your Python code. For further information, refer to Basics of Execution Handling (AutomationDesk Basic Practices 🕮).

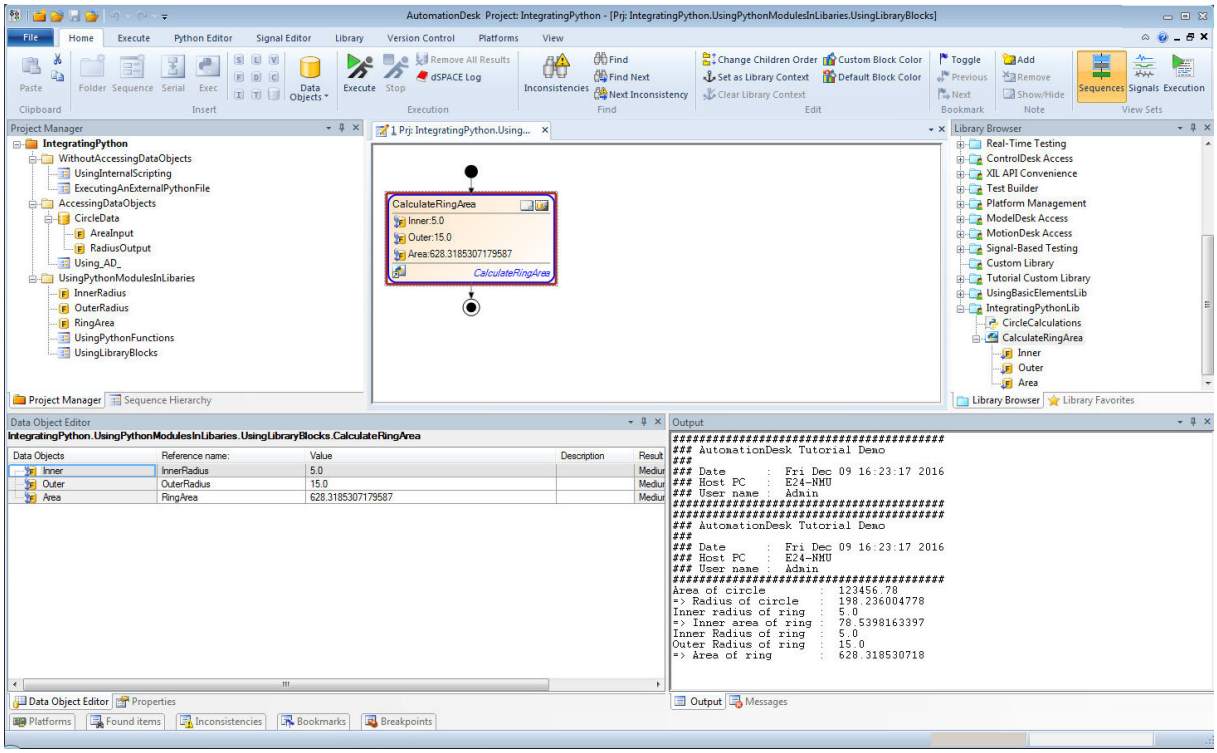
Representation in the User Interface

When you have performed all of the previous steps, a new project called *IntegratingPython* is opened in the Project Manager. It contains three folders with sequences that demonstrate the different ways to integrate Python scripts.

In your working folder, you created the *PrintHeader.py* file that contains a script which prints typical header information to the Output Viewer.

A new custom library called *IntegratingPythonLib* is open in the Library Bowser. It provides the following:

- A Python module with a function to calculate the area of a circle
- An automation block to calculate the area of a ring



Prepared Demo

If there were problems in editing your own AutomationDesk project, or if you want to compare your results with the result we wanted to achieve, you can look at the prepared demo project *Tutorialdemo06.adpx* and the related custom library *IntegratingPythonLib*. You can find it at `<DocumentsFolder>\Tutorial Demos`.

Further information

- For detailed information about the commands used, refer to Commands And Dialogs (AutomationDesk Basic Practices 🕮).
- For detailed information about the automation blocks used, refer to Automation Blocks (AutomationDesk Basic Practices 🕮).
- For information about the basic principles of AutomationDesk and a detailed description of using Python scripts in AutomationDesk, refer to Basics of Execution Handling (AutomationDesk Basic Practices 🕮) and Using Python in AutomationDesk (AutomationDesk Basic Practices 🕮).

What's next

The next lesson (refer to Lesson 7: Control-Flow-Based Testing on a Simulation Platform on page 131) shows you how to work with a simulation platform.

Lesson 7: Control-Flow-Based Testing on a Simulation Platform

Introduction

You learn how to implement a test routine via its control flow, which uses automation blocks to access your platform.

Introduction to Lesson 7

Safety Precautions

During this lesson, your platform is accessed. If you use other platforms than VEOS, you have to consider the following note that is relevant for dSPACE real-time hardware.

Note

To avoid risk of injury and/or property damage, you must read and understand the safety precautions. Refer to General Warning on page 133.

Task to be performed

This lesson describes how to perform the example task for implementing a test by specifying the related control flow. For more information, refer to Example Task for Implementing Tests on page 38.

Before you begin

The following preconditions must be met:

- If you want to access dSPACE real-time hardware, it must be installed and connected to your host PC. For more information, refer to the hardware-specific Installation and Configuration Guide. For example, for the DS1006 Processor Board, refer to DS1006 Hardware Installation and Configuration Guide.

If you want to access VEOS, the VEOS kernel must be installed and running on your host PC or a connected PC. For more information, refer to the VEOS Manual 🕮.

- You must know how the connection to your real-time hardware or VEOS is parameterized. For example, for a TCP/IP connection, you must know the IP address and the port number.
- The AutomationDesk demo projects must be available in your *Documents folder*. They are usually copied to the *Documents folder* the first time you start AutomationDesk. The copied demo projects include the simulation application used in this lesson.
- You must know how to create and edit AutomationDesk projects and how to edit, parameterize and execute automation sequences and blocks (refer to the previous lessons in this tutorial).

What you will learn

In this lesson, you will learn how to access your simulation platform to read and write variables.

- You will register your platform.
- You will load the demo simulation application to a platform and start it.
- You will write and read variable values of this running simulation application.
- You will capture variable values for a specified duration.
- You will add plots of the capture result to the report on the sequence execution.

All the steps are described in detail for a dSPACE DS1006 Processor Board, but generally you can follow this lesson for any dSPACE real-time hardware or VEOS.

Duration

Working through this lesson will take you about 45 minutes.

Summary

To check the results, refer to Results of Lesson 7 on page 154. This topic also provides information on executing the demo project without a connected platform.

Starting point

This lesson is independent of the previous work.

At this point in the tutorial, you access a platform for the first time. Thus, consider the following general warning before you start with the first step.

You can start with the first step, see below.

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.

Step 1: How to Register Your Platform

Introduction

To register a platform means to make it known to the host PC that runs AutomationDesk and to configure its connection. The registration and all the

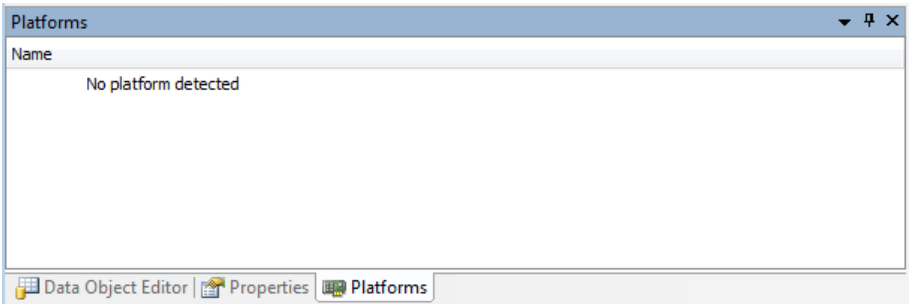
platform management are shared by the dSPACE software products that are installed on the same host PC, for example, ControlDesk.

To handle platform management tasks manually, you can use AutomationDesk's Project Manager or the commands on the **Platforms** ribbon. For the following steps, the Platform Manager is used.

Method

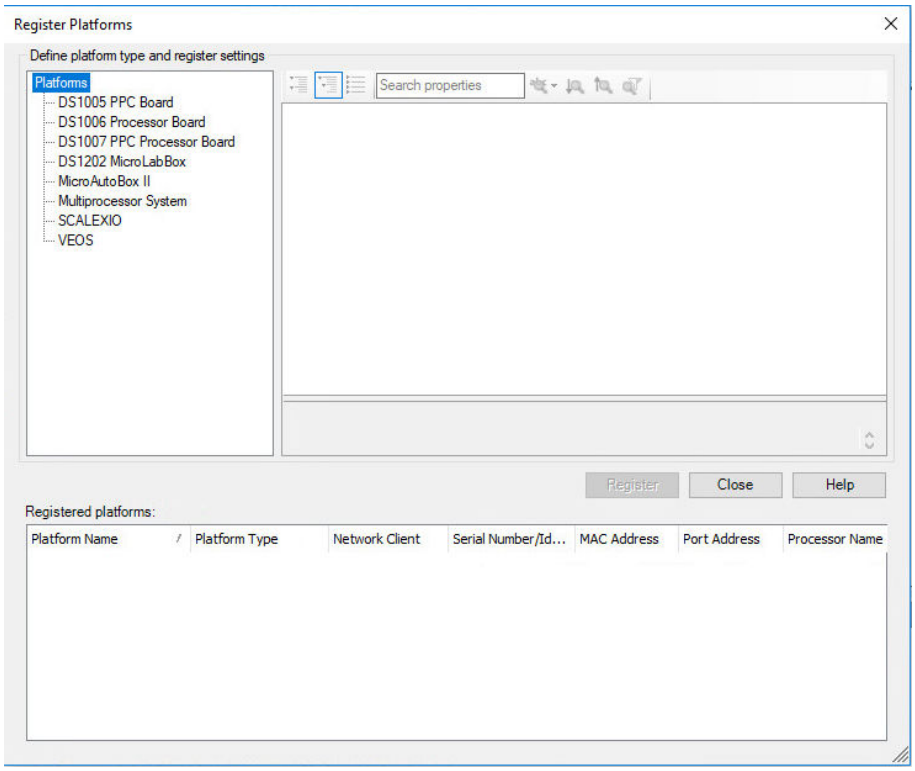
To register your platform

1 If the **Platform Manager** is hidden, you have to add it to the current screen arrangement. On the **View** ribbon, click **Controlbar - Switch Controlbars - Platform** to display the **Platform Manager**.

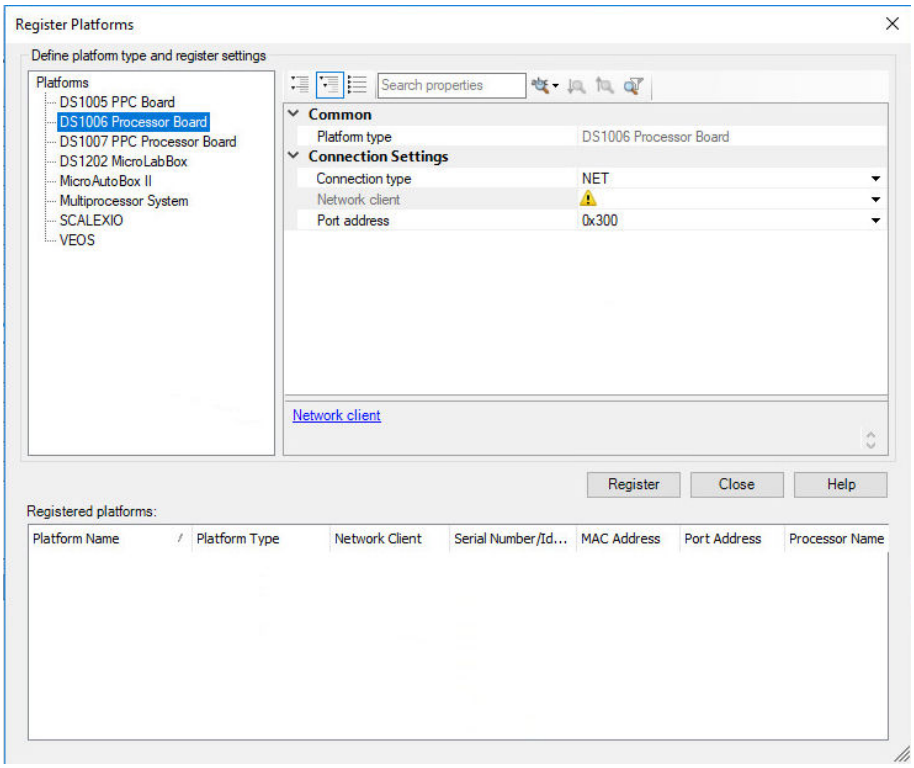


If the **Platform Manager** displays your platform, your platform is already registered. You can skip the rest of this step and proceed with Step 2: How to Load a Simulation Application to Your Platform and Start It on page 139.

2 In the **Platform Manager**, right-click a blank field to open the context menu and select **Register Platforms** to open the **Register Platforms** dialog.



3 From the Platforms list, select DS1006 Processor Board to specify the type of the platform to be registered.

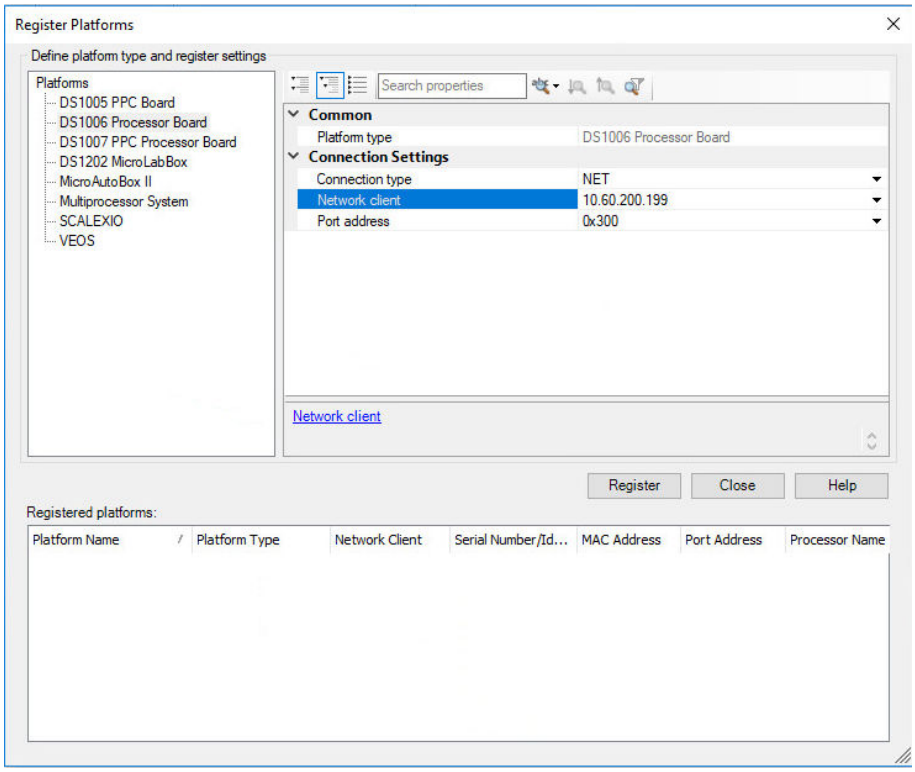


The platform type-specific property list is shown on the top right pane of the dialog.

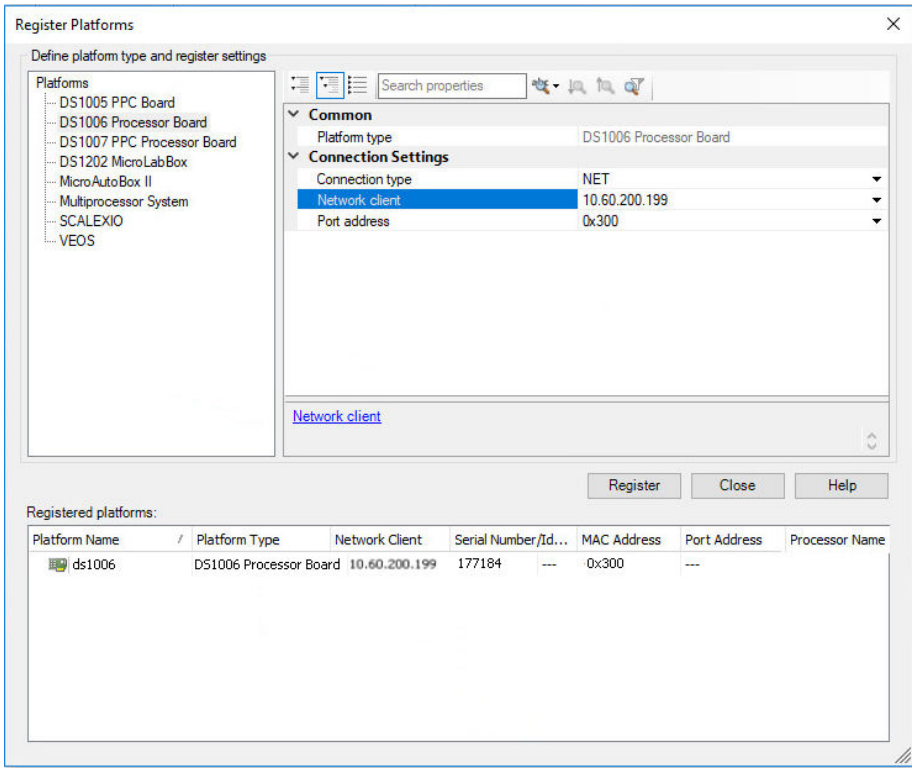
4 In the Connection Settings category of the property list, set the Connection type property to NET.

5 In the Network client edit field, enter the IP address of your DS1006 Processor Board.

6 In the Port address edit field, enter the base address of your DS1006 Processor Board as specified with the rotary switches on the board.



7 Click Register to start the registration.

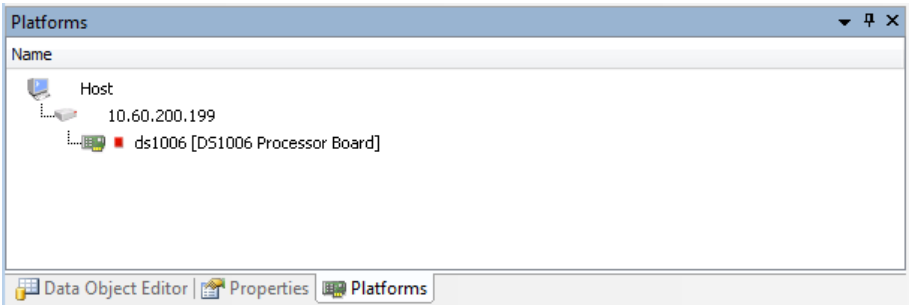


After registration, the platform is added to the Registered Platforms list.

8 Click Close to close the dialog.

Result

The registered platform is displayed in the Platform Manager. The red square (▪) indicates that there is no simulation application running on the platform.



What's next

Now you can download a simulation application to the registered platform. The next step describes how to do this.

Step 2: How to Load a Simulation Application to Your Platform and Start It

Introduction

Once the platform is registered, you can load a simulation application to it.

You can specify the simulation application to be loaded by its variable description file (SDF). This file is created when you build the simulation application. It contains information on the executable file, for example, an *.x86 file if you use a DS1006 Processor Board, and the variables that you can access during the execution of the simulation application.

AutomationDesk provides the following libraries to start a simulation application and access its variables:

- XIL API

  This is a built-in library providing data objects and blocks according to the MAPort definition in the ASAM AE XIL API standard.
- XIL API Convenience

  This is a custom library providing convenience blocks that are based on the XIL API library. These blocks facilitate platform access and are recommended for common use cases.

The XIL API library provides the MAPortConfiguration object type to store the configuration of the simulation application to access. It contains the name of the registered platform and the path and file name of the variable description file (SDF) that specifies the simulation application.

A demo simulation application is available with AutomationDesk. It represents a turn signal control in a car. Because the build process depends on the platform type used, you can find platform-specific simulation applications at `<DocumentsFolder>\TurnSignalTests\SampleExperiments\<Platform Type>`. Each of these simulation applications is configured to start the execution of the simulation application automatically after loading.

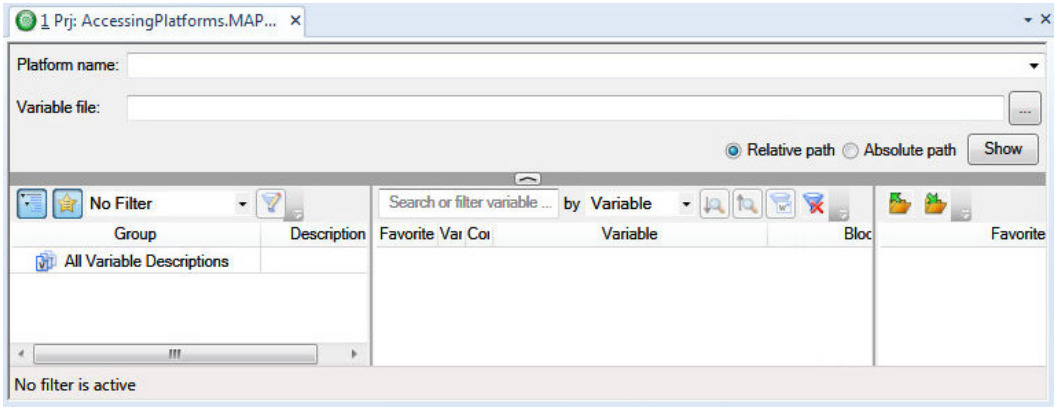
The included CDP files are ControlDesk project files that you can use with ControlDesk to perform experiments.

Part 1

To configure the platform access

1. In the Project Manager, create a project named `AccessingPlatforms`.
2. Select the AccessingPlatforms project and click Home – Insert – Data Objects - MAPortConfiguration to create a data object to store the model access port configuration.

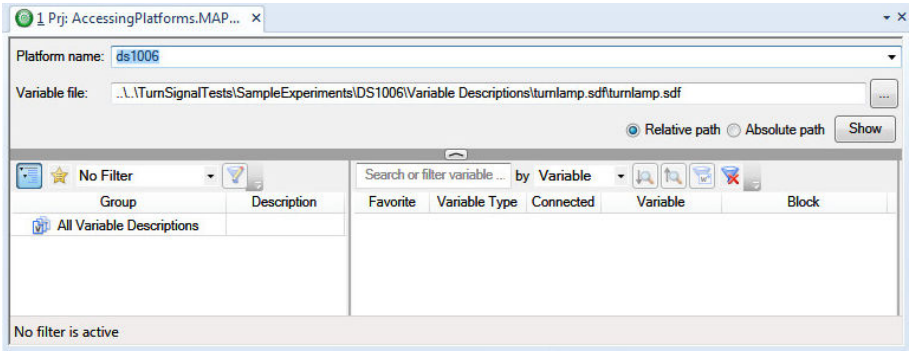
3 Double-click the MAPortConfiguration data object to open the Variables pane.



If the Favorite List is displayed on the right in the Variables pane, click the Show/Hide Favorites button ( ) to hide the Favorites List.

4 From the Platform name list, select ds1006 to assign the name of the registered platform to the platform access configuration.

5 Click the Browse button of the Variable file edit field to specify the variable description file (SDF). This must be the turnlamp.sdf file that belongs to the loaded simulation application. You will find it in `<DocumentsFolder>\TurnSignalTests\SampleExperiments\DS1006\VariableDescriptions\turnlamp.sdf`.



The MAPort is now configured. One of the next methods describes how to use the Variables pane to add variables to your AutomationDesk project.

Continue with next part

In the next part, you create the basic sequence to access your platform by automation blocks.

To make sure that the resources that you will add later are always released, the sequence is structured as a TryFinally block.

Part 2

To load a simulation application to your platform and to execute it

1 Add a new folder to your tutorial project and rename it to `ControlFlowBasedTesting`.

2 In the ControlFlowBasedTesting folder, create a new sequence and rename it to `AccessTurnsignalSimulation`.

3 Double-click the new sequence to open it in the Sequence Builder.

4 From the Main Library, drag a Control Flows - TryFinally block to the Sequence Builder.

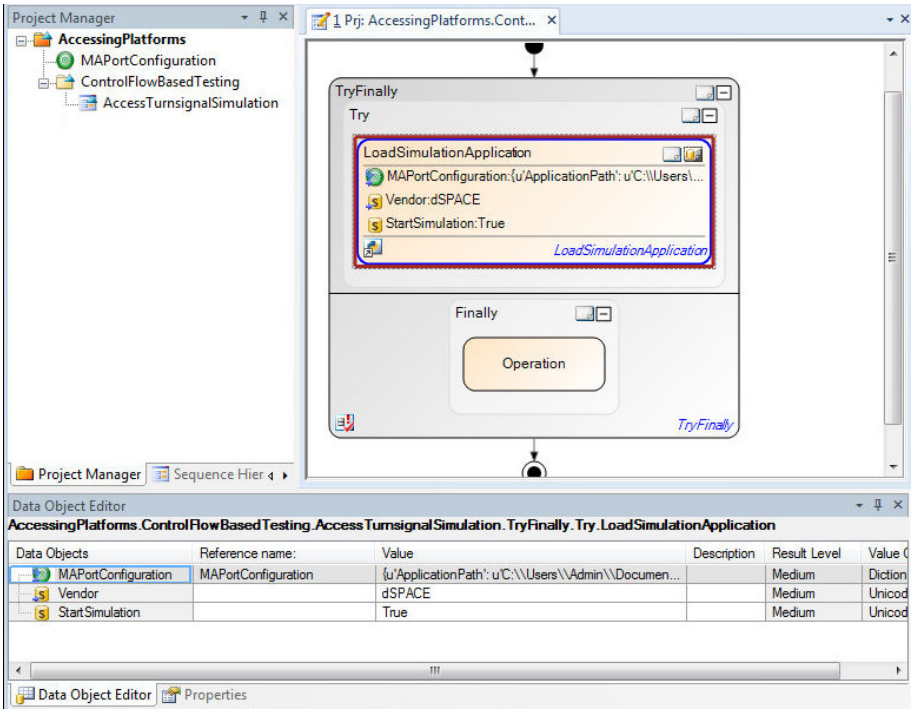
A termination procedure will be implemented later in the Finally path. It will be executed if an exception occurs in the Try path.

5 From the XIL API Convenience library, drag an Model Access Port - SimulationApplication - LoadSimulationApplication block to the Try path of the TryFinally block.

The data objects of the LoadSimulationApplication block are left at their default values:

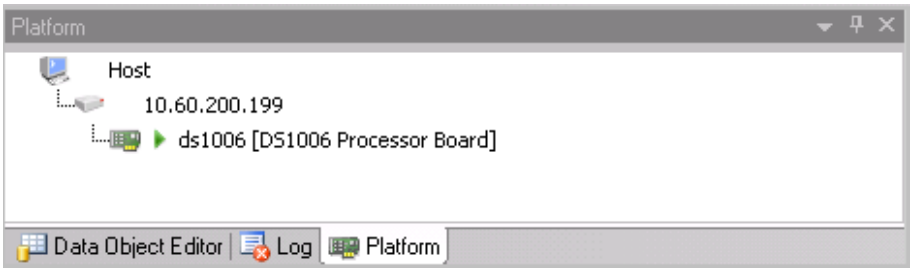
- By auto-referencing, the project-specific MAPortConfiguration data object is set as the reference for the block's data object of the same name to specify the platform and the simulation application.
- The Vendor String data object is set to `dSPACE` to specify the XIL API implementation.
- The StartSimulation String data object is set to `True` to specify that the simulation application starts automatically after it is loaded to the platform.

The Finally path of the TryFinally block is left unchanged.



Result

When you execute the sequence, AutomationDesk's demo simulation application is loaded to the platform and then started. In the Platform Manager, the green triangle (▸) indicates that the simulation application is running.



What's next

Now you can access variables of the running simulation application. The next step describes how to do this.

Step 3: How to Access Variable Values of the Running Simulation Application

Introduction

Once the simulation application is running, you can access the values of its variables via a model access port (MAPort) that is provided by the XIL API library.

A model access port is handled by using an MAPort data object that is created in your project. It is configured by the MAPortConfiguration data object and initialized and released by automation blocks of the XIL API Convenience library.

Part 1

To create a model access port for variable access

1 In the Project Manager, select the ControlFlowBasedTesting folder and click Home – Insert – Data Objects – MAPort to create a data object to handle the MAPort instance.

By creating the MAPort data object in the project before you add the related automation blocks, you enable AutomationDesk's auto-referencing feature.

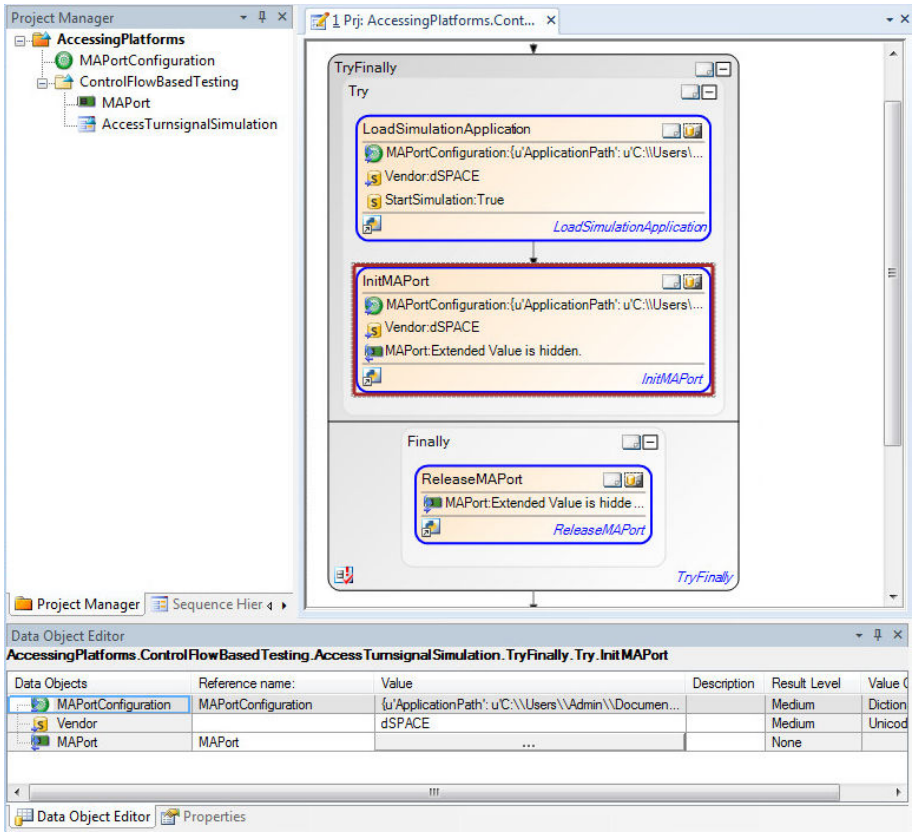
2 From the XIL API Convenience library, drag an Model Access Port - InitMAPort block to the Try path of the TryFinally block.

The following data objects of the InitMAPort block are left at their default values:

- By auto-referencing, the project-specific MAPortConfiguration data object is set as the reference for the block's data object of the same name to specify the platform and the simulation application.
- The Vendor String data object is set to dSPACE to specify the XIL API implementation.
- By auto-referencing, the project-specific MAPort data object is set as the reference for the block's data object of the same name. It is used later to access the variable values.

3 From the XIL API Convenience library, drag a Model Access Port – ReleaseMAPort block to the Finally path of the TryFinally block.

By auto-referencing, the project-specific MAPort data object is set as the reference for the block's data object of the same name.



When you execute this sequence, the model access port to access variables is created.

Continue with next part

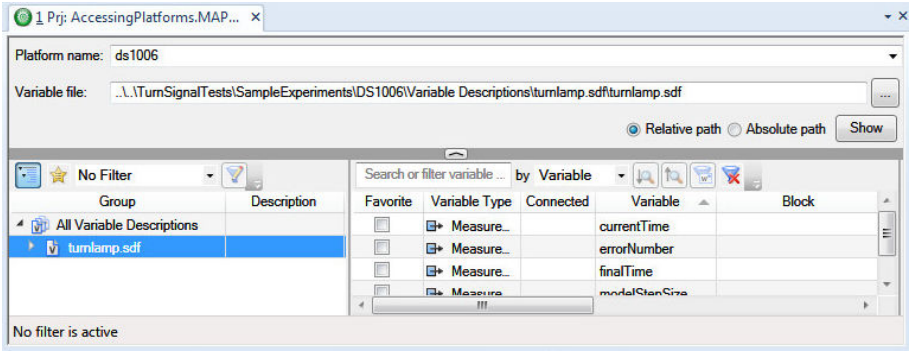
In the next part, you access a single variable value of the running simulation application and write a new value to the battery voltage of the turn signal model.

Part 2

To write a single variable value to the platform

1 Double-click the MAPortConfiguration object to open the Variables pane again.

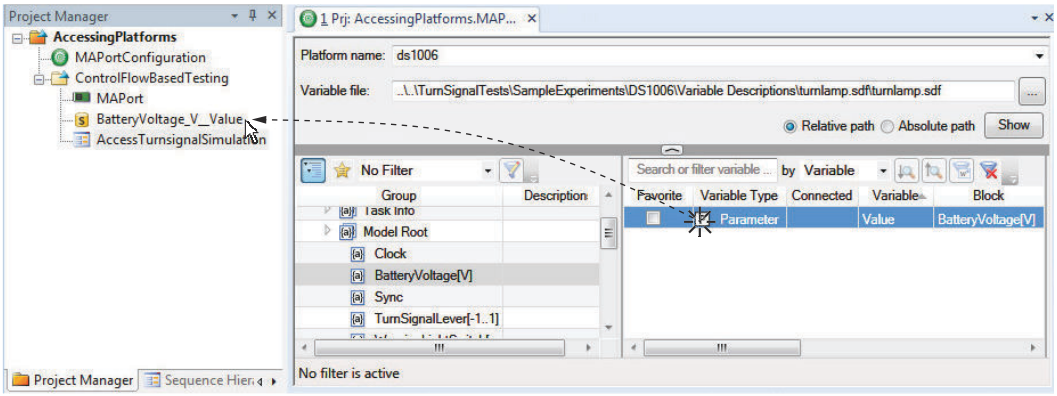
2 In the Variables pane, click Show to display the variable tree.



If the favorites list is displayed on the right in the Variables pane, click the Show/Hide Favorites button ( ) to hide the favorites list.

3 In the model tree, navigate to
All Variable Descriptions/turnlamp.sdf/Model Root/BatteryVoltage[V].

4 Drag the Value variable from the variable list and drop it on the ControlFlowBasedTesting folder.



A String data object named BatteryVoltage_V__Value is added to the folder. It contains the model path of the variable.

5 Add a Float data object to the ControlFlowBasedTesting folder and rename it to `InitialBatteryVoltage`.

6 Double-click the InitialBatteryVoltage data object to open the Value Editor and enter `12.3` to specify the value you want to write to the battery voltage variable in the turn signal simulation application.

Tip

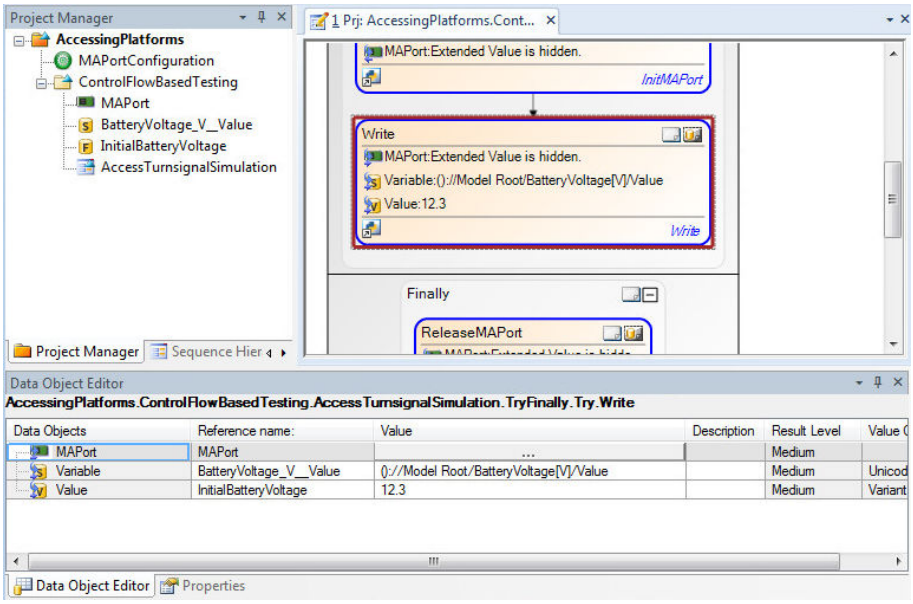
The Data Object Editor provides a Description column to specify additional information to a data object. For example, if you want to specify V as the unit for the InitialBatteryVoltage data object, you can do it here.

7 From the XIL API Convenience library, drag a Model Access Port - Write block to the end of the Try path.

By auto-referencing, the project-specific MAPort data object is set as the reference for the block's data object of the same name.

8 In the Data Object Editor, specify the project-specific BatteryVoltage_V__Value data object as the reference for the block's Variable data object. This data object specifies the model path of the battery voltage.

9 Specify the project-specific InitialBatteryVoltage data object as the reference for the block's Value data object. This data object stores the value of the battery voltage to be written to the running simulation application.



When you execute this sequence, the battery voltage will be set to 12.3 V.

Continue with next part

In the next part, you create a variable mapping in a Mapping data object, where variable aliases are mapped to variable model paths. Thus, you prepare to specify the access to simulator variables via their aliases.

Part 3

To specify a variable mapping

1 In the Project Manager, select the AccessingPlatforms project and click Home – Insert – Data Objects – Mapping to create a Mapping data object.

By creating the Mapping data object in the project before you add the related automation blocks, you enable AutomationDesk's auto-referencing feature.

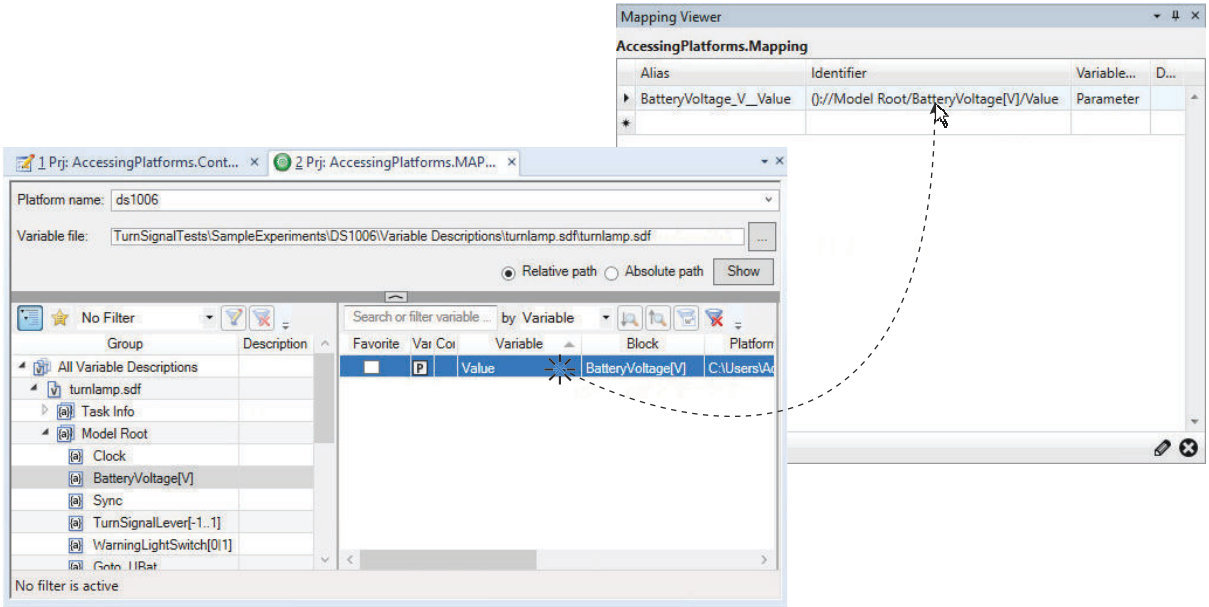
2 Double-click the Mapping data object to open it in the Mapping Viewer.

An empty variable mapping is displayed.

3 Drag the Mapping Viewer by its title bar and place it in the working area so that both the Variables pane and the Mapping Viewer are displayed without auto-hiding.

4 In the model tree of the Variables pane, navigate to All Variable Descriptions/turnlamp.sdf/Model Root/BatteryVoltage[V].

5 Drag the Value variable from the variable list of the Variables pane to the mapping table of the Mapping Viewer.

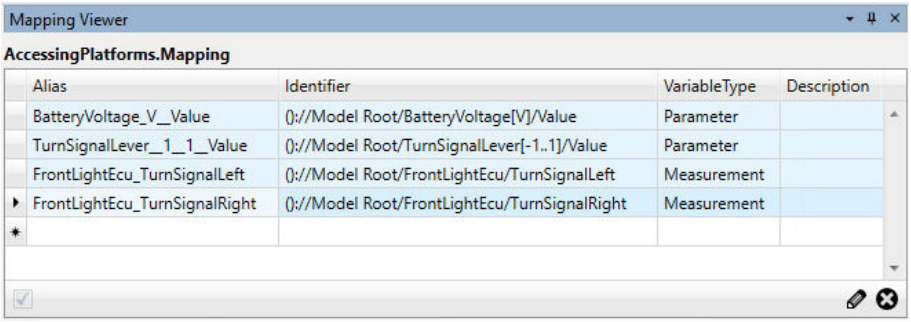


A mapping of the variable path to an automatically generated alias is added to the mapping table.

6 In the variable tree, navigate to All Variable Description/turnlamp.sdf/Model Root/TurnSignalLever[-1..1] and drag the Value variable to the mapping table.

7 Navigate to All Variable Description/turnlamp.sdf/Model Root/FrontLightEcu and drag the TurnSignalLeft and the TurnSignalRight variable to the mapping table.

The mappings of the variable aliases to the related model paths are displayed in the Mapping Viewer.



8 Close the Mapping Viewer.

You specified a variable mapping.

Continue with next part

In the next part, you read the variable values you specified in the variable mapping.

The read values are stored in a Dictionary data object where the key of each value is the variable's alias in the variable mapping.

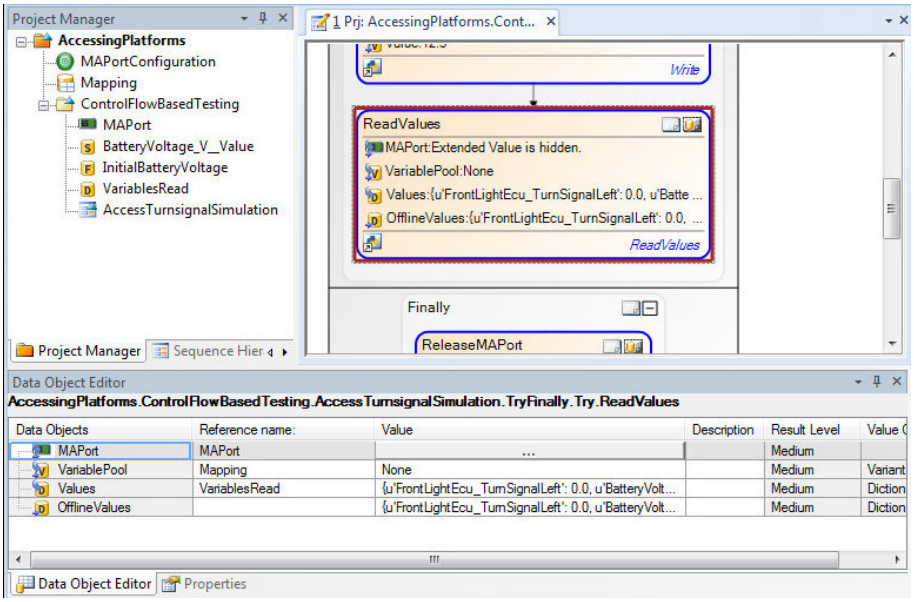
Part 4

To read the current values of several variables at once

1. Add a Dictionary data object to the ControlFlowBasedTesting folder and rename it to `VariablesRead`. This is to store the variables and their values after they have been read from the platform.
2. From the XIL API Convenience library, drag a Model Access Port - ReadValues block to the end of the Try path.

   By auto-referencing, the project-specific MAPort data object is set as the reference for the block's data object of the same name.
3. Specify the project-specific Mapping data object as the reference for the block's VariablePool Variant data object. This data object contains the paths of the variables to be read.
4. Specify the project-specific VariablesRead data object as the reference for the block's Values data object.

   This dictionary defines which variables are to be read and, later, stores the values of these variables. Because the dictionary is empty before the first execution of the sequence, all variables contained in the variable mapping will be read.



Result

When you execute the sequence, the ReadValues block reads the values for each variable contained in the Mapping data object and saves them to the VariablesRead Dictionary data object.

What's next

Now you can capture the values of several variables for a specified duration. The next step describes how to do this.

Step 4: How to Capture Variable Values for a Specified Duration

Introduction

In contrast to *reading* a variable value, *capturing* a variable means reading the variable value not only once but repeatedly for a specified duration.

The repeated reading of variables is controlled via the tasks implemented in the simulation application. To configure a Capture data object, you must therefore specify the task to be used. You find a table with the task names for the different platforms at the end of this lesson. Further capture settings specify the variables to be captured, the downsampling and the default duration if no other stop conditions are configured.

Part 1

To initialize a capture process

1. In the Project Manager, add a String data object to the AccessingPlatforms project and rename it to `TaskName`.
2. Double-click TaskName to open the Value Editor and enter `HostService` to specify the name of the task within the simulation application to be used for data capturing.

   Note

   The task name to be specified depends on the registered platform. For information on the task names used by other platforms, refer to Results of Lesson 7 on page 154.

3. Add a List data object to the ControlFlowBasedTesting folder and rename it to `SignalNames`. This is to hold the signal names to be captured and plotted.
4. In the Mapping Viewer, multiselect all contained variables and drop them on the SignalNames List data object to specify the list of signals to be captured.

   You can verify the contents of the SignalNames data object by moving the cursor to its icon or name in the Project Manager. It contains the following value:
   ```
   [u'BatteryVoltage_V__Value',
   u'TurnSignalLever_1_1__Value',
   u'FrontLightEcu_TurnSignalLeft',
   u'FrontLightEcu_TurnSignalRight']
   ```
5. Select the ControlFlowBasedTesting folder and click Home – Insert – Data Objects - Capture to create a data object to store the configuration of the capture process.

6 From the XIL API Convenience library, drag a Model Access Port - Capture - InitializeCapture block to the end of the Try path.

By auto-referencing, the project-specific MAPort and Capture data objects are set as the references for the block's data objects that have the same names.

7 Specify the project-specific TaskName data object as the reference for the block's data object of the same name.

8 Specify the project-specific Mapping data object as the reference for the block's VariablePool data object.

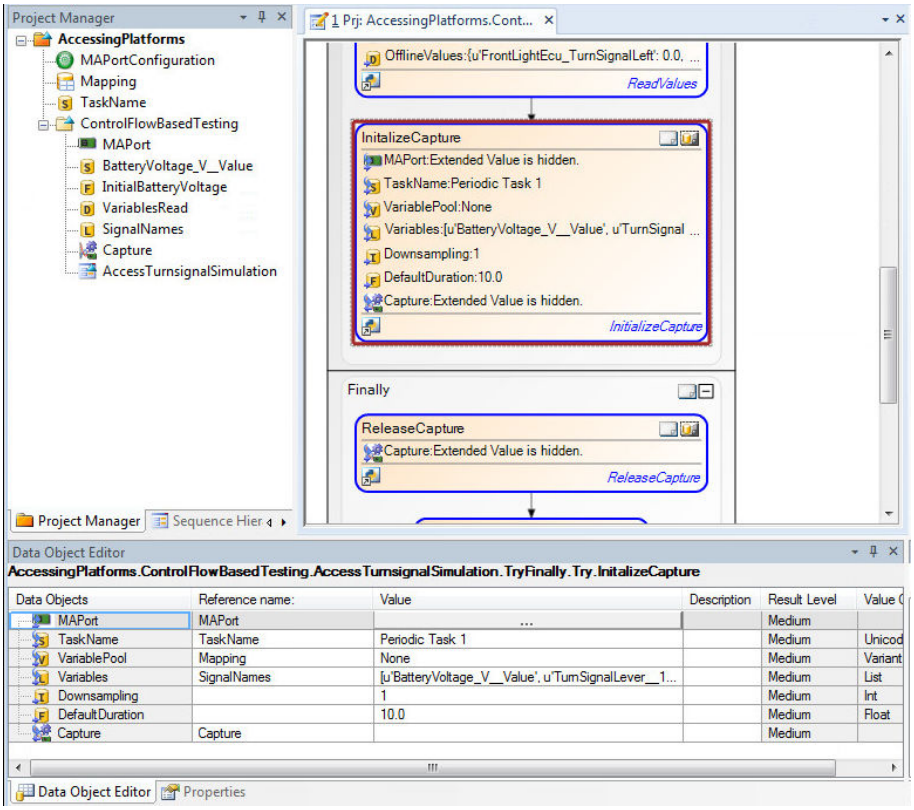
9 Specify the project-specific SignalNames data object as the reference for the block's Variables data object. This data object stores the signal names.

The following data objects of the InitializeCapture block are left at their default values:

- Downsampling is set to `1` to capture every value the task generates.
- DefaultDuration is set to `10.0` to limit the default duration of the capture to ten seconds.

10 From the XIL API Convenience library, drag a Model Access Port - Capture - ReleaseCapture block to the beginning of the Finally path.

By auto-referencing, the project-specific Capture data object is set as the reference for the block's data object of the same name.



You have initialized the capture.

Continue with next part

In the next part, you specify the scenario for the capturing process.

To see the reactions to changed variable values in the capture result, the position of the turn-signal lever is changed while the capture is running. The durations between the value changes are specified with Sleep blocks.

No stop conditions have been implemented in this scenario. The capture process is therefore terminated after the default duration of ten seconds.

Part 2

To specify the scenario for the capturing process

1 From the XIL API Convenience library, drag a Model Access Port - Capture - StartCapture block to the end of the Try path.

By auto-referencing, the project-specific Capture data object is set as the reference for the block's data object of the same name.

The block's optional CaptureResultWriter data object is left unchanged to specify that the capture is stored at the host PC's memory.

2 From the Main Library, drag a Basic Elements - Sleep block to the Try path.

Via this and the following automation blocks, you implement the changes of the turn-signal level over time. By using Sleep blocks, you specify, how long the lever position is held.

3. From the XIL API Convenience library, drag a Model Access Port - Write block to the Try path.

   By using Write blocks, you parameterize the position of the turn-signal lever. The value `0` represents a turn-signal lever that is switched off, the value `1` represents the lever switched to the right.

   By auto-referencing, the project-specific MAPort data object is set as the reference for the block's data object of the same name.

4. In the Object Editor, click the Browse button in the Reference name column of the Variable data object.

   The Data Object Selector opens.

5. In the Data Object Selector, select AccessingPlatforms - Mapping - LabelMapping - TurnSignalLever_1_1_Value - Identifier.

   This specifies the variable that represents the state of the turn-signal lever.

6. Multi-select the Write and the Sleep block and press `Ctrl+C` to copy the blocks to the Clipboard.

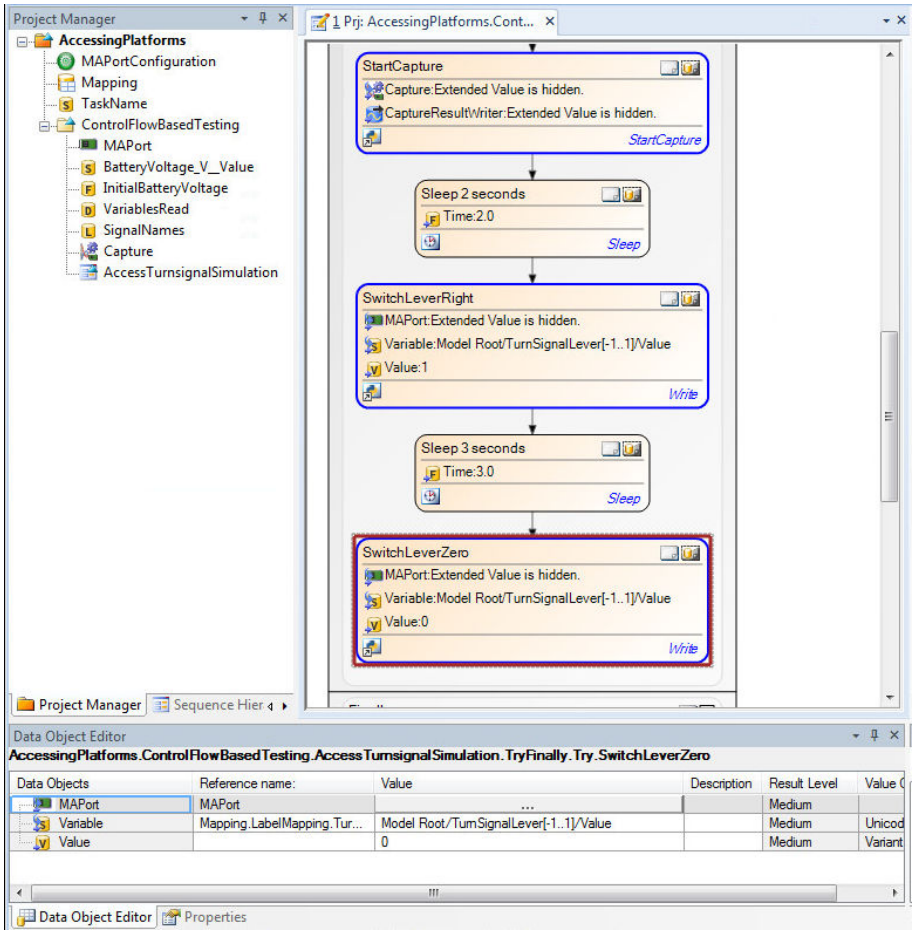
7. Select the Try path and press `Ctrl+V` to append the copied blocks from the Clipboard.

8. Rename the first Sleep block to `Sleep2Seconds` and set its Time data object to `2.0`.

9. Rename the first Write block to `SwitchLeverRight` and set its Value data object to `1`.

10. Rename the second Sleep block to `Sleep3Seconds` and set its Time data object to `3.0` .

11 Rename the second Write block to `SwitchLeverZero` and set its Value data object to `0`.



You specified the scenario for the capturing process.

Continue with next part

In the next part, you access the result of the capturing process and add plots of it to the sequence's report.

To access the data which is captured during a capturing process, the XIL API library provides the CaptureResult data object. The CaptureResult is plotted to the report and does not need to be stored in the AutomationDesk project. Clearing its values after plotting minimizes the size of the AutomationDesk project.

Part 3

To add plots of the capture result to the report

1 In the Project Manager, select the ControlFlowBasedTesting folder and click Home – Insert – Data Objects - CaptureResult to create a data object to store the configuration of the capture process.

2 From the XIL API Convenience library, drag a Model Access Port - Capture - GetCaptureResult block to the end of the Try path.

By auto-referencing, the project-specific **Capture** and **CaptureResult** data objects are set as the references for the block's data objects that have the same names.

The block's **StopCapturing** Integer data object is left at its default value **1**.

3. In the Data Object Editor, parameterize the block's **WhenFinished** Integer data object with **1**.

   Leave the block's **CaptureResultMetaData**, **OfflineCaptureResult** and **OfflineCaptureResultMetaData** default data objects unchanged.

4. From the **XIL API Convenience** library, drag a **Model Access Port - Capture - AddCustomPlotsToReport** block to the end of the **Try** path.

   By auto-referencing, the project-specific **CaptureResult** data object is set as the reference for the block's data object with the same name.

5. In the Data Object Editor, specify the project-specific **SignalNames** data object as the reference for the block's data object of the same name.

6. Specify the project-specific **Mapping** data object as the reference for the block's **VariablePool** data object. This data object specifies which signals are to be plotted.

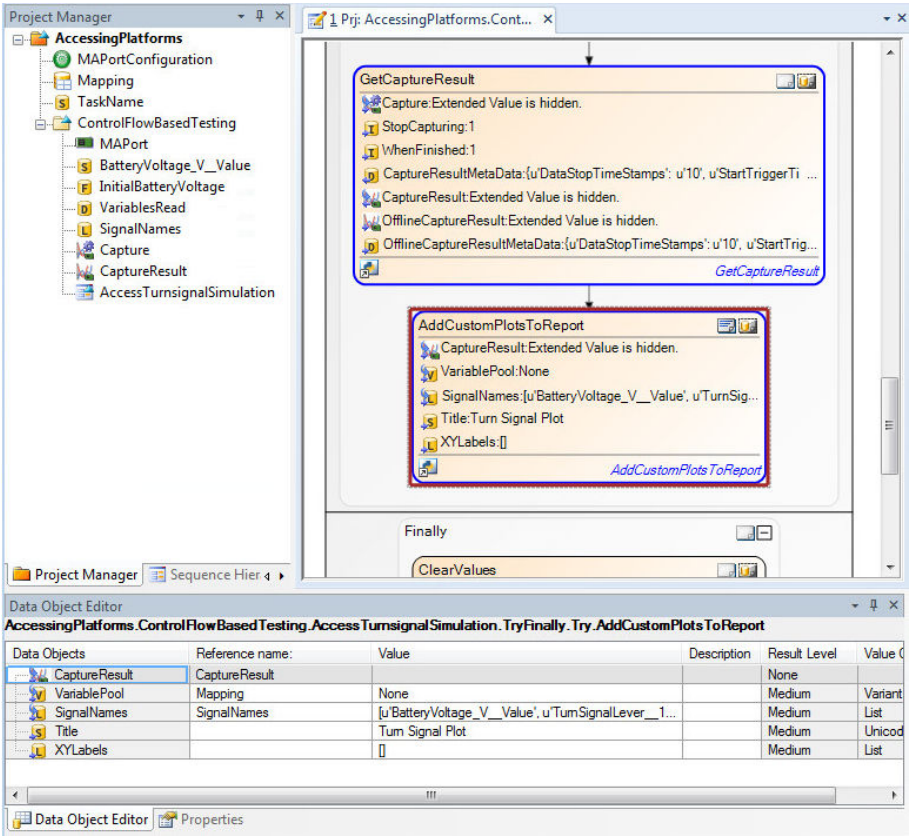
7. Double-click the blocks **Title** String data object to open the Value Editor. Enter `Turn Signal Plot` to specify the title of the plot in the report.

   Leave the block's **XYLabels** data object unchanged.

8. From the **Main Library**, drag a **Basic Elements - ClearValues** block to the beginning of the **Finally** path.

9. In the Sequence Builder, select the **ClearValues** block and click **Home – Insert – Data Objects - CaptureResult** to specify the **CaptureResult** data object to be cleared.

10 Specify the project-specific CaptureResult data object as the reference for the block's data object of the same name.



Result

You added plots of the capture result to the report.

What's next

You created a sequence that captures variable values for the duration of a specified scenario and plots the result in the sequence report.

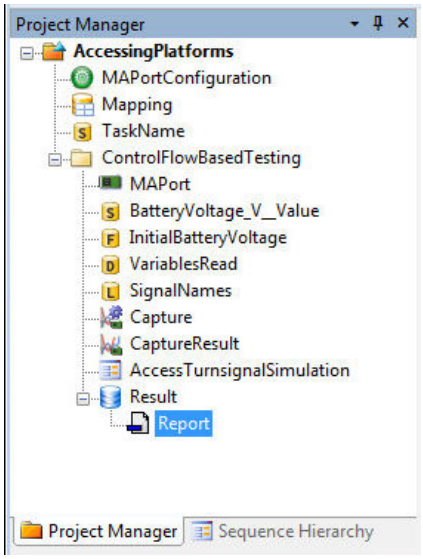
For a summary of the lesson's exercises, refer to Results of Lesson 7.

Results of Lesson 7

Results

If you have performed all the steps in this lesson, the turn-signal simulation application is running on the platform you registered.

You created the AccessingPlatform AutomationDesk project that contains data objects to configure the access to the simulation application running on the platform.



You have learned how to write one single variable and how to read several variables at once. For this, the model paths of the variables to be accessed are configured in strings.

You have also learned how to capture variable values for a specified duration. For this, the model paths to be accessed are collected in a variable mapping that is implemented in a Mapping data object.

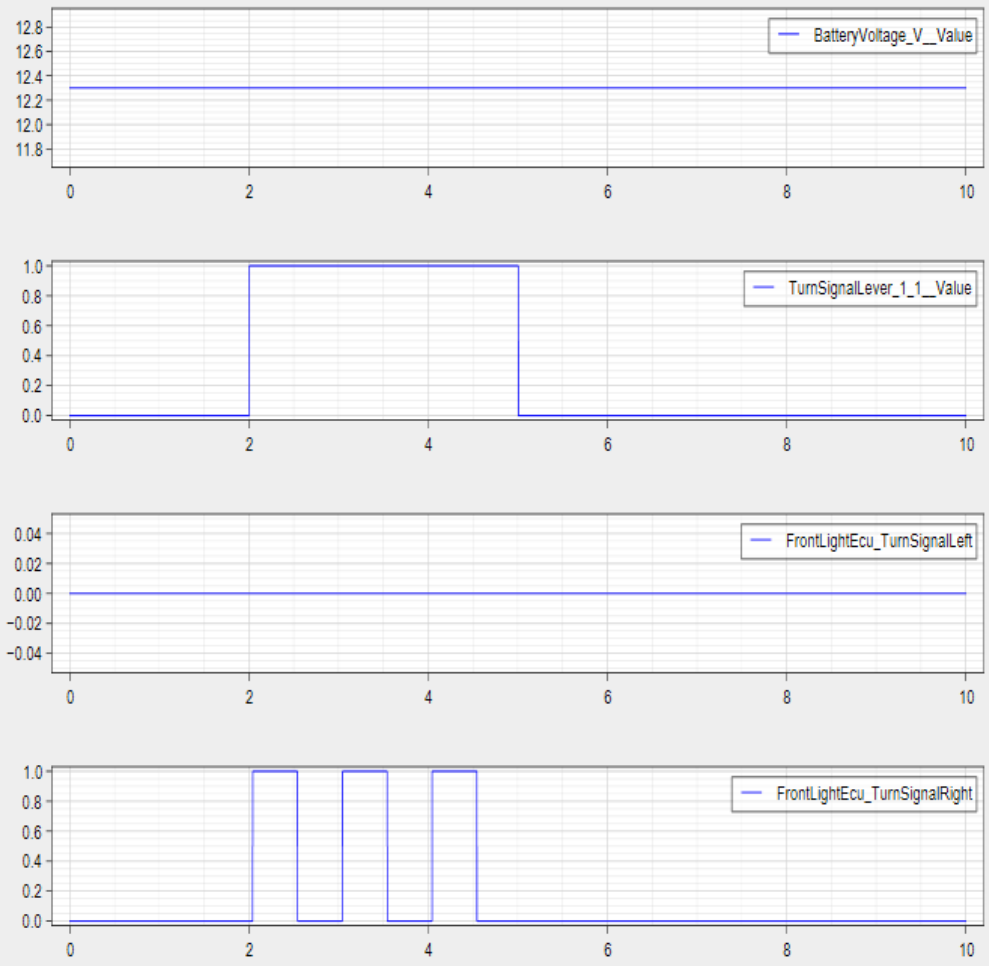
In the AccessTurnSignalSimulation sequence, you implemented a scenario where the turn-signal lever is switched to the right and back and the state of the front lights is captured.

Executing this sequence leads to the following report.

AccessTurnsignalSimulation (Sequence)

Start time:	2015-04-01 13:41:15	**Name:**	AccessTurnsignalSimulation
Stop time:	2015-04-01 13:41:32	**Library link:**	
Execution duration:	16.905 sec.	**Author:**	dSPACE GmbH, Paderborn, Germany
Creation date:	2014-01-13 10:48:56	**Modification date:**	2015-02-24 11:15:50
Hierarchy:	AccessingPlatforms. ControlFlowBasedTesting. AccessTurnsignalSimulation		
Description:			
Result state:	Executed		

Turn Signal Plot



Variable	Value
FrontLightEcu_TurnSignalLeft	Model Root/FrontLightEcu/TurnSignalLeft
BatteryVoltage_V__Value	Model Root/BatteryVoltage[V]/Value
FrontLightEcu_TurnSignalRight	Model Root/FrontLightEcu/TurnSignalRight
TurnSignalLever_1_1__Value	Model Root/TurnSignalLever[-1..1]/Value

You can see that the turn-signal lever is switched after 2 seconds for a duration of 3 seconds and the front right turn-signal reacts correctly to the state of the turn-signal lever.

Prepared Demo

If you had problems while working through this lesson, or if you want to compare your results with the result we wanted to achieve, you can look at the prepared demo project *Tutorialdemo07.adpx* in `<DocumentsFolder>\Tutorial Demos`.

While this lesson describes the handling with a DS1006 Processor Board, the prepared demo is configured for VEOS as platform.

Using other platforms

You can also work through this lesson if you do not have the same real-time hardware as used here. If you use other dSPACE real-time hardware or VEOS, note that:

- The dialogs to register a platform vary slightly according to the platform type and the way it is connected to the host PC. For more information, refer to **Registering and Managing dSPACE Platforms** (AutomationDesk Accessing Simulation Platforms 🕮).

  For platform-specific details on registering a platform, refer to **Register Platforms** (AutomationDesk Accessing Simulation Platforms 🕮).

  For platform-specific details on loading and starting a simulation application on DS1005, DS1006, DS1104, MicroAutoBox II and Multiprocessor Systems, refer to **Load** (refer to Real-Time Application - Load (AutomationDesk Accessing Simulation Platforms 🕮)). If you want to load and start the simulation application on VEOS, SCALEXIO, DS1202 MicroLabBox, MicroAutoBox III, or DS1007 refer to **Load and Start** (refer to Real-Time Application / Offline Simulation Application - Load and Start (AutomationDesk Accessing Simulation Platforms 🕮)).
- The task names of the turn-signal simulation application depend on the platform type, as shown in the following table.

Platform Type	Task Name
DS1005 PPC Board	HostService
DS1006 Processor Board	
DS1007 PPC Processor Board	
DS1202 MicroLabBox	
MicroAutoBox	
MicroAutoBox III	
SCALEXIO System	
VEOS	Periodic Task 1

Platform Type	Task Name
DS1005 Multiprocessor System	masterAppl/HostService
DS1006 Multiprocessor System	
DS1007 Multicore System	
DS1202 Multicore System	
SCALEXIO Multicore System	
SCALEXIO Multiprocessor System	
VEOS Multicore	masterAppl/Periodic Task 1

Running demo without a platform

If no platform is available to you, you can execute this demo in offline operation mode, i.e., the demo's sequence works with included sample data. For more information on operation modes, refer to Executing Sequences Using Different Operation Modes (AutomationDesk Basic Practices 🕮).

Further information

- For detailed information about the commands used, refer to Commands And Dialogs (AutomationDesk Accessing Simulation Platforms 🕮).
- For detailed information about the automation blocks used, refer to Automation Blocks (AutomationDesk Accessing Simulation Platforms 🕮).
- For more information on using the XIL API Convenience library, refer to Accessing Simulation Platforms via the XIL API Convenience Library (AutomationDesk Accessing Simulation Platforms 🕮).

What's next

The next lesson (refer to Lesson 8: Signal-Based Testing on a Simulation Platform on page 159) shows you how to implement access to a platform via signal-based testing.

Lesson 8: Signal-Based Testing on a Simulation Platform

Introduction

You learn how to create a similar test as described in lesson 7 but now you implement the test by using the signal-based testing strategy.

Introduction to Lesson 8

Safety precautions

During this lesson your platform is accessed. If you use other platforms than VEOS, you have to consider the following note that is relevant for dSPACE real-time hardware.

Note

To avoid risk of injury and/or property damage, you must read and understand the safety precautions. Refer to General Warning on page 133.

Task to be performed

This lesson describes how to perform the example task for implementing a test by using a standardized test scenario that you can configure by specifying the involved signals. For more information, refer to Example Task for Implementing Tests on page 38.

Before you begin

The following preconditions must be met:

- If you want to access dSPACE real-time hardware, it must be installed and connected to your host PC. For more information, refer to the hardware-specific Installation and Configuration Guide, for example. For the DS1006 Processor Board, refer to DS1006 Hardware Installation and Configuration Guide.

If you want to access VEOS, VEOS must be installed and running on your host or on a connected PC. For more information, refer to VEOS Manual 🕮.

- The AutomationDesk demo projects must be available in your *Documents folder*. They are usually copied to the *Documents folder* the first time you start AutomationDesk. The copied demo projects include the simulation application used in this lesson.
- The platform that you want to access must be registered. For more information, refer to the previous lesson (refer to Step 1: How to Register Your Platform on page 133).
- The simulation application to be accessed must be downloaded and started on the platform. For more information, refer to the previous lesson (refer to Step 2: How to Load a Simulation Application to Your Platform and Start It on page 139).
- You must know how to edit AutomationDesk projects and how to edit, parameterize and execute automation sequences and blocks (refer to lessons 1 to 3 in this tutorial).

What you will learn

In this lesson, you will learn how to access your simulation platform to read and write variables.

- You will add a signal-based test case sequence to your project.
- You will configure the simulation application to be accessed on the platform.
- You will configure which simulator variables are accessed during the test.
- You will configure the signals to stimulate simulator variables.
- You will configure the reference signals and methods to compare them to captured signals.
- You will monitor test execution and inspect the test result.

Duration

Working through this lesson will take you about 60 minutes.

Summary

To check the results, refer to Results of Lesson 8 on page 176.

Starting point

This lesson starts with a new folder in your AutomationDesk project.

If you have not created your own AutomationDesk project as shown in the previous lesson, you can use the prepared demo project *Tutorialdemo07.adpx* in `<DocumentsFolder>\Tutorial Demos`.

Now you can start with the first step, see below.

Step 1: How to Configure Variable Access

Introduction

For signal-based testing, you use already coded, standardized test scenarios that you parameterize via data objects. In the first step, you create and configure these data objects.

Configuring platform access The variable access during a signal-based test is configured in the same way as for the control-flow-based testing that is describes in the previous lesson.

- You must configure which platform and which simulation application is to be accessed on the platform in a MAPortConfiguration data object.
- You must configure a variable mapping in a Mapping data object that contains the mappings of the aliases of each variable to the variable's model path.
- In a String data object, you must configure the name of the task in the simulation application that is used to perform the data capturing on the platform.

Configuring the synchronization variable Only one additional variable is required for configuring a signal-based test. It must be writable and is used during the test to synchronize the start of stimulation and capturing of the other variables.

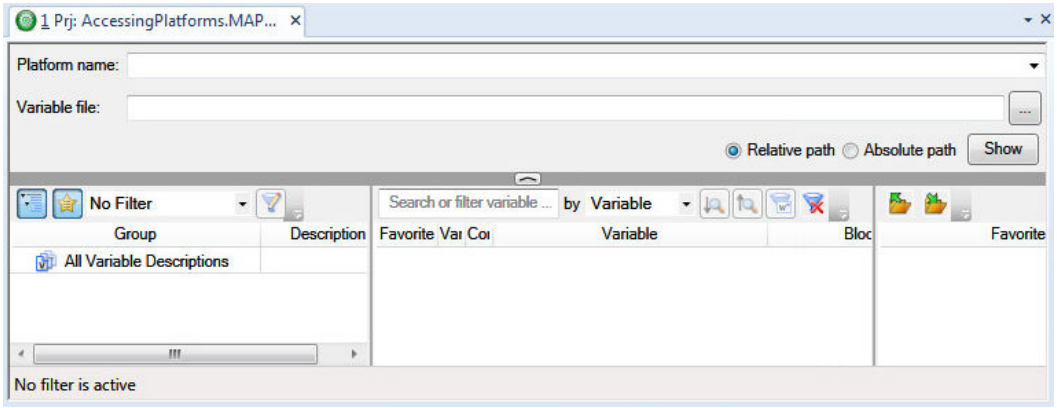
First, configure the platform and the simulation application to be accessed. You can skip this part, if you created and configured the project-specific MAPortConfiguration data object in the previous lesson.

Part 1

To configure the platform and the simulation application to be accessed

1. On the Home ribbon, click View Sets - Sequences to activate the panes suitable for editing sequences.
2. In the Project Manager, select your project element and click Home – Insert – Data Objects – MAPortConfiguration to create a data object to store the model access port configuration.

3 Double-click the MAPortConfiguration data object to open the Variables pane.

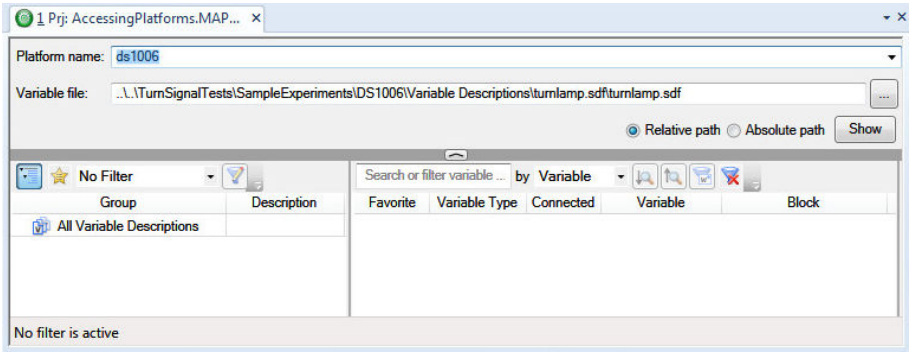


If the favorites list is displayed on the right in the Variables pane, click the Show/Hide Favorites button ( ) to hide the favorites list.

4 From the Platform name list, select ds1006 to assign the name of the registered platform to the platform access configuration.

The platform name depends on the used platform type. For more information, refer to Using other platforms on page 157.

5 Click the Browse button of the Variable file edit field to specify the variable description file (SDF). This must be the turnlamp.sdf file that belongs to the loaded simulation application. You will find it in `<DocumentsFolder>\TurnSignalTests\SampleExperiments\DS1006\VariableDescriptions\turnlamp.sdf`.



Continue with next part

You configured which platform and which simulation application is to be accessed on the platform.

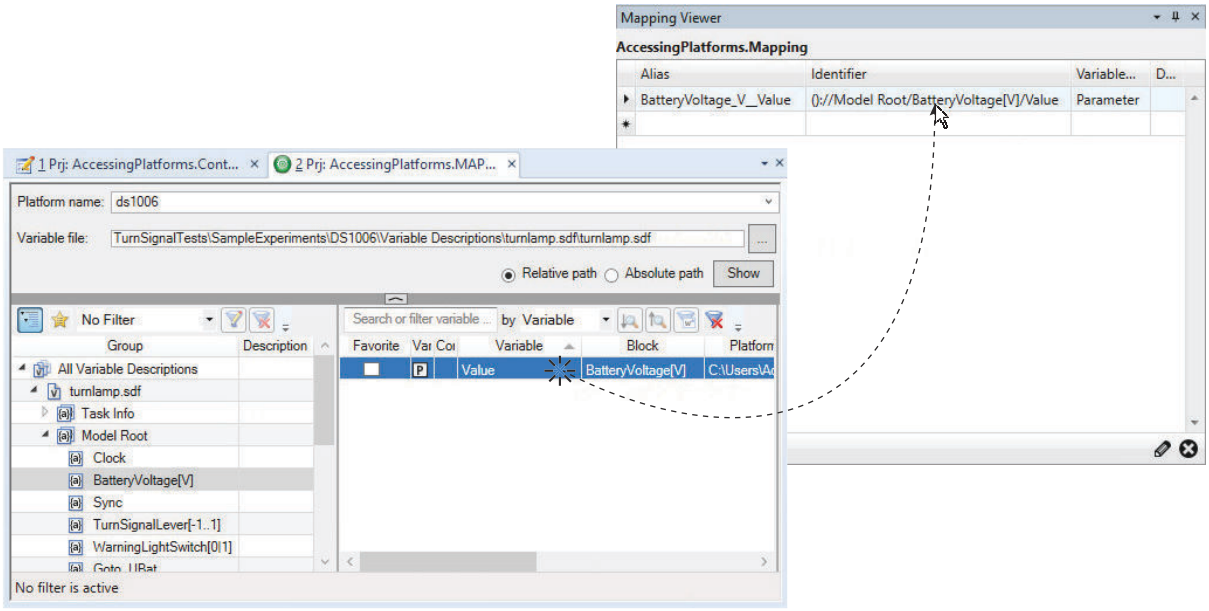
Now you can map the variable aliases to the variables model paths in a variable mapping that you implement by using a Mapping data object.

You can skip this part, if you created and configured the project-specific Mapping data object in the previous lesson.

Part 2

To configure a variable mapping

1. In the Project Manager, select your project element and click Home – Insert – Data Objects – Mapping to create a Mapping data object.
2. Double-click the Mapping data object to open it in the Mapping Viewer.
   An empty mapping table is displayed.
3. Drag the Mapping Viewer by its title bar and place it in the working area so that both the Variables pane and the Mapping Viewer are displayed without auto-hiding.
4. In the model tree of the Variables pane, navigate to All Variable Descriptions/turnlamp.sdf/Model Root/BatteryVoltage[V].
5. Drag the Value variable from the variable list of the Variables pane to the mapping table of the Mapping Viewer.



A mapping of the variable path to an automatically generated alias is added to the mapping table.

6. In the variable tree, navigate to All Variable Description/turnlamp.sdf/Model Root/TurnSignalLever[-1..1] and drag the Value variable to the mapping table.
7. Navigate to All Variable Description/turnlamp.sdf/Model Root/FrontLightEcu and drag the TurnSignalLeft and the TurnSignalRight variable to the mapping table.

The mappings of the variable aliases to the related model paths are displayed in the Mapping Viewer.

Continue with next part

You configured a variable mapping that contains the mapping of the variable's alias to its model path for each accessed simulator variable in a Mapping data object.

Now you can configure the name of the task in the simulation application that is used to perform the capturing on the simulator platform.

You can skip this method, if you created and configured the project-specific TaskName String data object in the previous lesson.

Part 3

To configure the task that is used for capturing

1. In the Project Manager, add a String data object to your project and rename it to `TaskName`.
2. Double-click TaskName to open the Value Editor and enter `HostService` to specify the name of the task within the simulation application to be used for data capturing.

   The task name depends on the used platform type. For more information, refer to Using other platforms on page 157.

Continue with next part

You configured the platform access as common when using automation blocks of the XIL API Convenience library.

With the following method, you configure the variable that is used for synchronizing variable stimulation and capturing during the test.

Part 4

To configure the synchronization variable

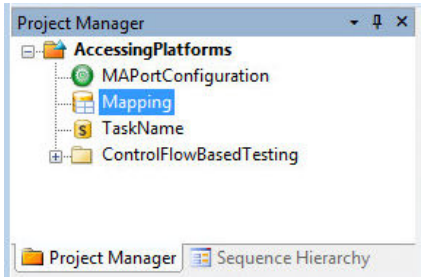
1. In the model tree of the Variables pane, navigate to All Variable Descriptions/turnlamp.sdf/Model Root/Sync.
2. Drag the Value variable from the variable list to the mapping table in the Mapping Viewer.

   The Sync_Value String data object that contains the variable's model path is added to the variable mapping.

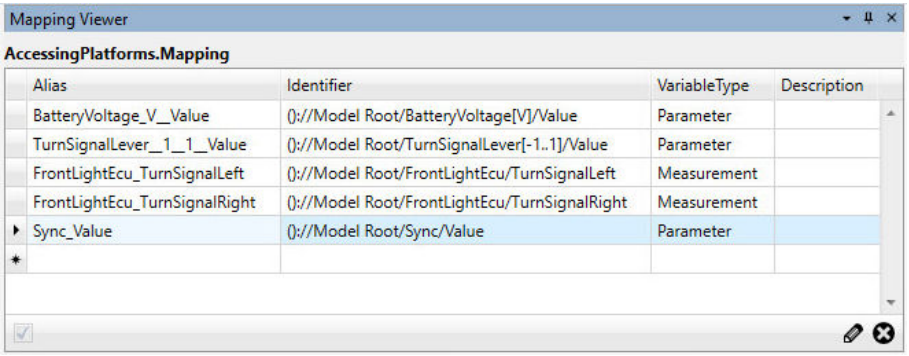
   This simulator variable is used to synchronize the start of stimulating and capturing the other variables during the test.

Result

You added the data objects that configure the signal-based test to your project.



In the variable mapping in the Mapping data object, you configured the variable access during the test, including the variable that is used for the synchronization of variable stimulation and capturing.



Alias	Identifier	VariableType	Description
BatteryVoltage_V__Value	():://Model Root/BatteryVoltage[V]/Value	Parameter	
TurnSignalLever__1__1__Value	():://Model Root/TurnSignalLever[-1..1]/Value	Parameter	
FrontLightEcu_TurnSignalLeft	():://Model Root/FrontLightEcu/TurnSignalLeft	Measurement	
FrontLightEcu_TurnSignalRight	():://Model Root/FrontLightEcu/TurnSignalRight	Measurement	
Sync_Value	():://Model Root/Sync/Value	Parameter	

What's next

You can now add and parameterize the sequence that implements the standardized test scenario. The next step describes how to do this.

Step 2: How to Add the Test Sequence

Introduction

The Signal-Based Testing library provides the TestCase sequence template in which the standardized test scenario is already coded.

Before you can parameterize and execute such a test case, you must instantiate the sequence template by adding it to your project. Then you can parameterize the sequence with references to the data objects that you created in the previous step.

Method

To add the test sequence

1 In the Project Manager, add a new folder to your tutorial project and rename it to `SignalBasedTesting`.

2 From the **Signal-Based Testing** library, drag a **TestCase** sequence template to the **SignalBasedTesting** folder and rename it to `TurnSignalSBT`.

   A test case sequence is added to your project.

3 In the Data Object Editor, specify the project-specific data objects **MAPortConfiguration** and **TaskName** as the references for the sequence-specific data objects that have the same names.

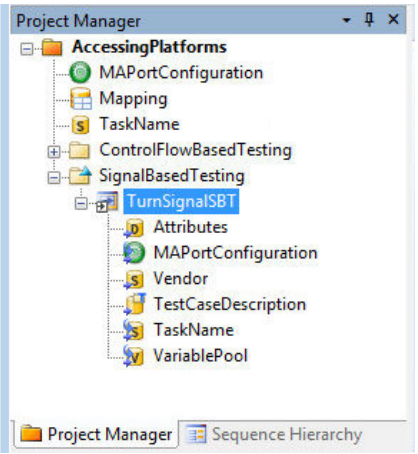
   This parametrizes the platform access of the sequence.

4 Specify the project-specific **Mapping** data object as the reference for the sequence-specific **VariablePool** data object.

   This parameterizes the model paths of the variables to be accessed.

Result

You added the test sequence to your project that implements a standardized test scenario and parameterized it.



What's next

You can now configure the test case by specifying the involved signals in the test case description. The next step (refer to Step 3: How to Configure the Involved Signals on page 166) describes how to do this.

Step 3: How to Configure the Involved Signals

Introduction

In this step, you create the file with the description of the signals that are involved in your signal-based test.

The simulator variables to be stimulated or captured during the test case are specified via the descriptions of the involved signals. These signal descriptions are gathered to a test case description that is stored in an STZ file. 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.

You can also add the descriptions of reference signals that are used for result evaluation to a test case description.

To edit a test case description, AutomationDesk provides the Signal Editor.

Signal Editor You can open one test case description in one Signal Editor page. The signals that are contained in the test case description are displayed in a signal table. In the last column of the signal table, the shape of the signal is displayed.

Signals and Segments Signals can consist of a sequence of segments. A segment is characterized by its type, i.e., by its basic shape. For example, there are Const segments that represent a straight line or Pulse segments that represent a square wave.

One way to add signals to a test case description is to drag variables from the variable mapping to the Signal Editor. The name of the signal created in this way is the same as the variable alias. Thus, the signal is mapped to the simulator variable for signal-based testing.

Another way to add signals to a test case description is to drag a signal symbol from the Signal Selector to the Signal Editor. By dragging and dropping segment symbols, you can add segments to a segment signal.

Properties Signals and segments are configured via their properties. In the Properties pane, you can edit the properties of the signal or segment that is currently selected in the Signal Editor. For example, you can change the duration of a selected segment.

You can define the shape of a segment signal by configuring the sequence of its segments and by specifying the values of the segments' properties.

Action The Action property of a signal specifies how a signal is treated during a signal-based test.

The following table shows the provided Action values.

Action	Description
Stimulate	This signal is generated by a signal generator to stimulate the mapped model variable during the run time of the simulation application.
Capture	The values of the variable that are mapped to this signal are captured during the run time of the simulation application. The data is stored in an MDF file (ASAM Common MDF Version 4.1, file name extension: MF4) that is added to the signal as a DataFile segment.
Sync	This signal is used to synchronize the stimulation and capturing during the run time of the simulation application. It defines the start of the Stimulate signals and the start and end of the specified captures.
None	This signal does not interact with the simulation application. It is used for reference signals of evaluation methods, for example.

Evaluation method You can specify the captured signals to be compared to expected reference signals at the end of the signal-based test. To do this, you can select one of the following methods:

Evaluation Method	Description
IsEqual	Checks whether the values of the evaluated signal are equal to the specified reference signal.

Evaluation Method	Description
IsNotEqual	Checks whether the values of the evaluated signal are not equal to the specified reference signal.
IsAboveBound	Checks whether the values of the evaluated signal are above the specified bound signal.
IsBelowBound	Checks whether the values of the evaluated signal are below the specified bound signal.
IsInsideBounds	Checks whether the values of the evaluated signal are inside the range that is specified by the upper bound and the lower bound signals.
IsOutsideBounds	Checks whether the values of the evaluated signal are outside the range that is specified by the upper bound and the lower bound signal.
IsInsideRegion	Checks whether the values of the evaluated signal are inside a region that is specified by the reference signal and absolute tolerances.
IsInsideDynamicRegion	Checks whether the values of the evaluated signal are inside a region that is specified by the reference signal and absolute and relative tolerances.
None	No evaluation is performed.

Depending on the selected evaluation method, you must specify reference signals, tolerances, and bounds.

The shapes of the reference signals are edited in the same way as the signals that are used for stimulation, except for the Action property, which is set to **None**.

Part 1

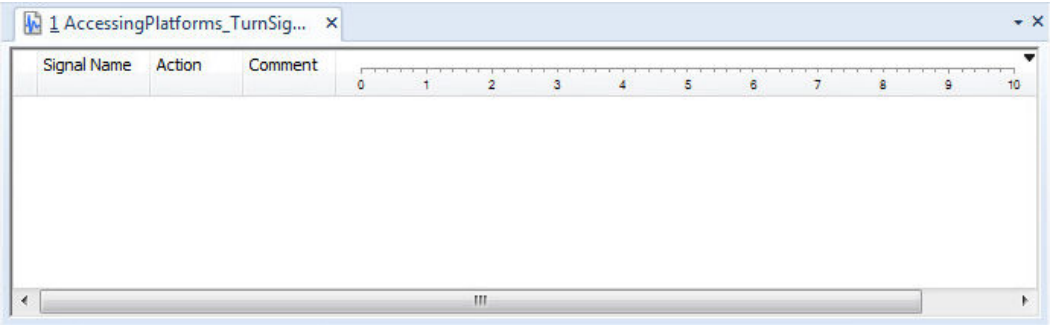
To configure the test case description file

1 In the Project Manager, double-click the TestCaseDescription data object of the TurnSignalSBT sequence. This data object holds the name and the path of the file that contains the signal descriptions for configuring the test.

The Create or Open SignalGenerator file dialog opens. In the File name edit field, `AccessingPlatforms_TurnSignalSBT.stz` is displayed as the default file name for the test case description.

2 Click OK to open the specified STZ file that stores the descriptions of the signals that are involved in the test.

In the working area, the test case description opens in a Signal Editor page. It shows an empty signal table.



Continue with next part

You configured the test case description file that stores the involved signal descriptions.

In the following part, you add signal descriptions to the empty test case description.

Part 2

To add signals that are mapped to simulator variables

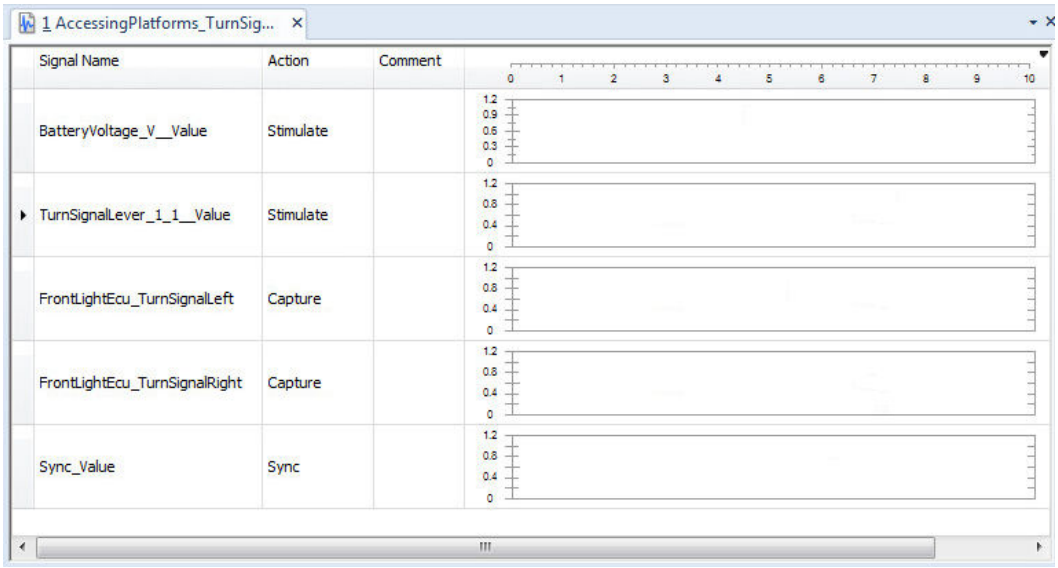
1 In the Mapping Viewer, select all the variables contained in the variable mapping and drag them to the Signal Editor.

For each variable, a signal description is added to the signal table in the Signal Editor. Via their signal names, the signals are mapped to simulator variables' aliases.

By default, the Action property of the signals that relate to variables of the Parameter type is set to Stimulate. The Action property of the signals that relate to variables of the Measurement type is set to Capture.

2 For the Sync_Value signal, select Sync as the Action property.

This variable is used to synchronize the starting and stopping of variable stimulation and data capturing.



Continue with next part

You have added empty signals to the test case description that are mapped to simulator variables.

Except for the capturing signals, you have to configure the signals for stimulating, the synchronization signal, and the reference signals.

In the next part, you specify the shape of the stimulating signals by adding segments to the signals and specifying the segments' properties.

Part 3

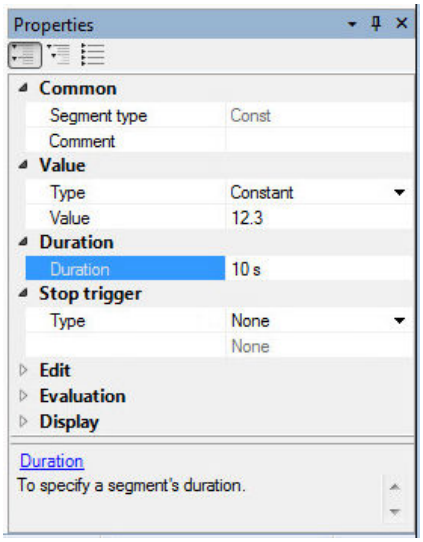
To configure the stimulating signals

1 On the Signal Editor ribbon, click View Sets - Signals to activate the panes suitable for editing signals.

2 Configure the shape of the signal that specifies a constant battery voltage of `12.3` volts for the entire duration of the test, i.e., for `10` seconds.

From the Signal Selector, drag a Const segment to the BatteryVoltage_V__Value signal. The added segment is parameterized with default properties and displayed in the Signal Editor.

3 Select the new Const segment, and in the Properties pane, set the Duration - Duration property to `10` seconds.



4 Set the Value -Type property to `Constant` and Value - Value to `12.3` volt.

5 Now configure the shape of the signal that specifies the turn-signal lever state. The value `0` represents a turn-signal lever that is switched off, the value `1` represents the lever switched to the right.

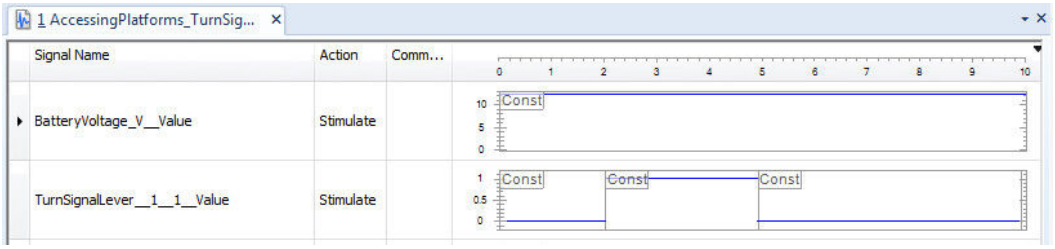
In the test scenario, a turn signal that is switched off is switched to the right after two seconds for a duration of three seconds and then switched to off again.

To implement this sequence of turn-signal lever states, drag three times a Const segment from the Signal Selector to the TurnSignalLever_1_1__Value signal.

6 Parameterize the three Const segments as previously described with the following properties:

Segment	Duration - Duration	Value - Type	Value - Value
1	2	Constant	0
2	3	Constant	1
3	5	Constant	0

The shapes of the stimulating signals that you configured are displayed in the Signal Editor.



Continue with next part

You have specified the shape of the stimulating signals.

In the next part, you specify the shape of the signal that is used for synchronizing the stimulating and capturing.

Each test case description must contain exactly one synchronization signal that consists of Const segments not equal to zero and that has no zero crossing.

Part 4

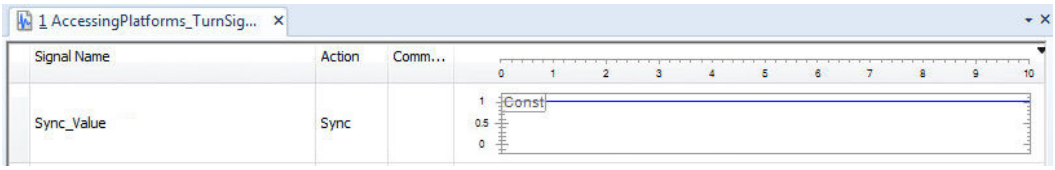
To configure the synchronization signal

1 From the Signal Selector, drag a Const segment to the Sync_Value signal.
The added segment is parameterized with default properties and displayed in the Signal Editor.

2 Select the new Const segment, and in the Properties pane, set the Duration - Duration property to `10` seconds.

The resulting signal shape is displayed in the Signal Editor.

The shapes of the stimulating signals that you configured are displayed in the Signal Editor.



Continue with next part

You have specified the shape of the synchronization signal.

In the next part, you add reference signals to the test case description and specify their shapes. These signals are used later to evaluate the captured signals.

Part 5

To add and configure the reference signals

1 Create a reference signal for the state of the left turn signal.

From the Signal Selector's Signals group, drag a Segment Signal symbol to the Signal Editor.

An empty segment signal is added to the signal list.

2 In the Signal Name column, rename the new signal to `Reference_TurnSignalLeft`.

3 The shape of the signals that represents the state of the left turn signal is expected to be constantly zero.

From the Signal Selector's Segments group, drag a Const symbol to the new signal's last column in the signal list.

A default Const segment is added and shown in the signal's shape display.

4 Select the new segment, and in the Properties pane, set Value - Type to `Constant`, Value - Value to `0`, and Duration - Duration to `10` seconds.

The displayed Const segment changes accordingly.

5 Then create a reference signal for the state of the right turn signal. The state oft the right turn signal is expected to be a square pulse as long as the turn-signal lever is switched to the right. You can create the base of the reference signal via copy & paste.

In the Signal Editor, select the TurnSignalLever_1_1__Value signal by clicking the signal's first cell in the signal list.

The signal's description is highlighted.

6 Via copy & paste, duplicate the selected signal description.

An identical signal with three segments is added to the test case description. The signal name of the new signals is postfixed with _1.

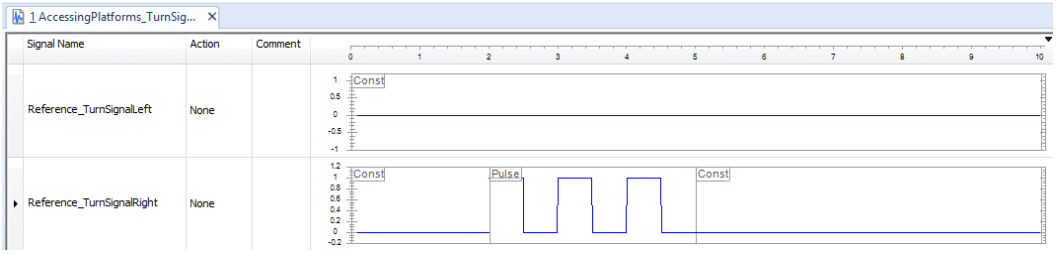
7 In the Signal Name column, rename the new signal to `Reference_TurnSignalRight`.

8 Select the second segment of the Reference_TurnSignalRight signal and from its context menu select Convert Segment - Pulse.

The second Const segment is converted to a Pulse segment.

9 In the Action column, set the value for the Reference_TurnSignalLeft and Reference_TurnSignalRight signals to `None`.

The shapes of the reference signals that you configured are displayed in the Signal Editor.



Continue with next part

You configured the reference signals that are used for evaluation.

In the next part, you specify how the captured signals are compared to the reference signals.

Part 6

To configure the evaluation method

1. In the Signal Editor, select the FrontLightEcu_TurnSignalLeft signal by clicking the signal's first cell in the signal list.

   The signal's description is highlighted and the signal's properties are shown in the Properties pane.

2. In the Properties pane, set the Evaluation - Evaluation method property to `IsEqual`.

   This configures the evaluation to check whether the deviation of the captured data in the direction of the x-axis is within a specified tolerance.

   The method-specific properties Reference, Signal and Tolerance appear in the Properties pane.

3. Set the Evaluation - Reference property to Reference_TurnSignalLeft.

   This configures the signal that is used as the reference for the evaluation method.

4. Set the Evaluation - Tolerance property to `0.1`.

   In the display of the FrontLightEcu_TurnSignalLeft signal, two parallels that represent the borders of the tolerated values are shown.

5. In the Signal Editor, select the FrontLightEcu_TurnSignalRight signal to configure the evaluation method for this signal.

   The signal's description is highlighted and the signal's properties are shown in the Properties pane.

6. In the Properties Pane, set the Evaluation - Evaluation method property to `IsInsideRegion`.

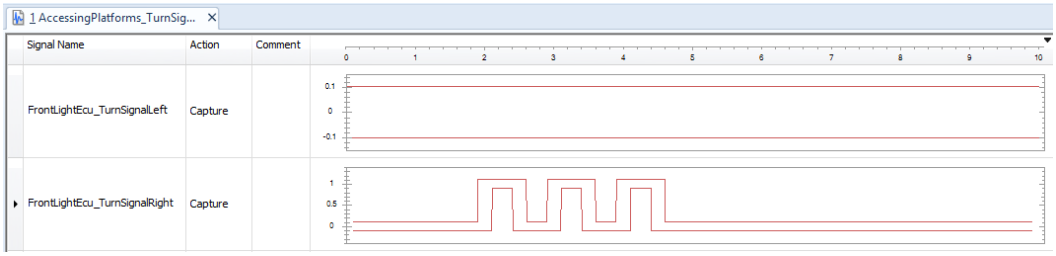
   This configures the evaluation to check whether the deviation of the captured data in the direction of the x-axis and the y-axis is within a specified region.

   The method-specific properties appear on the Properties pane.

7. Set the Evaluation - Reference property to Reference_TurnSignalRight.

   This configures which signal is used as the reference for the evaluation method.

   In the display of the FrontLightEcu_TurnSignalRight signal, the borders of the region that is used for evaluation is shown.



8. On the Signal Editor ribbon, click Signal Description Set - Save Signal Set.

   You configured the evaluation method and the complete test case description is saved to the specified STZ file.

Result

You specified all signals that are involved in the signal-based test.

What's next

You can now execute the completely configured signal-based test. The next step describes how to do this.

Step 4: How to Execute the Test

Introduction

You can execute the instantiated test case sequence on a registered platform in the same way as you execute sequences that you implemented as control flows.

You can track the execution of the test in the Message Viewer.

At the end of the test, a report of the test is generated.

Method

To execute the test

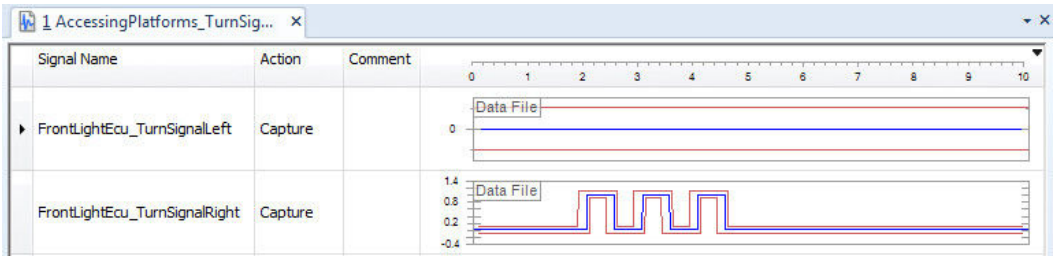
1. On the **Signal Editor** ribbon, click **View Sets - Execution** to activate the panes suitable for executing tests.
2. In the Project Manager, select the **SignalBasedTesting** folder.
3. On the **Execute** ribbon, click **Execution - Execute** to start the test.

During the test, the reached test steps and phases are written to the Output Viewer, together with their evaluation verdicts.

```
Output

[1> TurnsignalSimulationSBT
    ======================


[1.1> Initialization
      --------------
<1.1] Initialization --> Executed

[1.2> ActionsAndEvaluation
      --------------------
[1.2.1> StimulateAndCapture ()
    Load TestCaseDescription to Target...
    Start SignalGenerator and Capture...
    Add Captured Signals to TestCaseDescription...
<1.2.1] StimulateAndCapture --> Executed
[1.2.2> Evaluate ()
[1.2.2.1> Evaluation of "FrontLightEcu_TurnSignalLeft". ()
<1.2.2.1] Evaluation of "FrontLightEcu_TurnSignalLeft". --> Passed
[1.2.2.2> Evaluation of "FrontLightEcu_TurnSignalRight". ()
<1.2.2.2] Evaluation of "FrontLightEcu_TurnSignalRight". --> Passed
<1.2.2] Evaluate --> Passed
<1.2] ActionsAndEvaluation --> Passed

[1.3> Cleanup
      -------
<1.3] Cleanup --> Executed
<1] TurnsignalSimulationSBT --> Passed
```

Result

After the execution of the test finished, the shapes of the captured signals were displayed together with the borders of the evaluation regions in the Signal Editor.



In the Project Manager, a result node was added that provides the report of the execution of the signal-based test. The result element is prefixed with the symbol, which indicates the **Passed** result state.

You can open the report by double-clicking its element in the Project Manager.

What's next

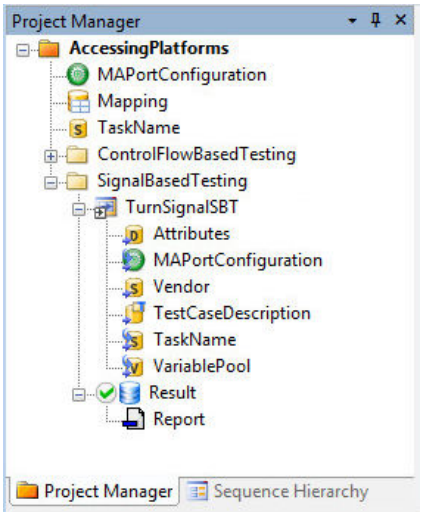
For a summary of the lesson's exercises, refer to Results of Lesson 8 on page 176.

Results of Lesson 8

Results

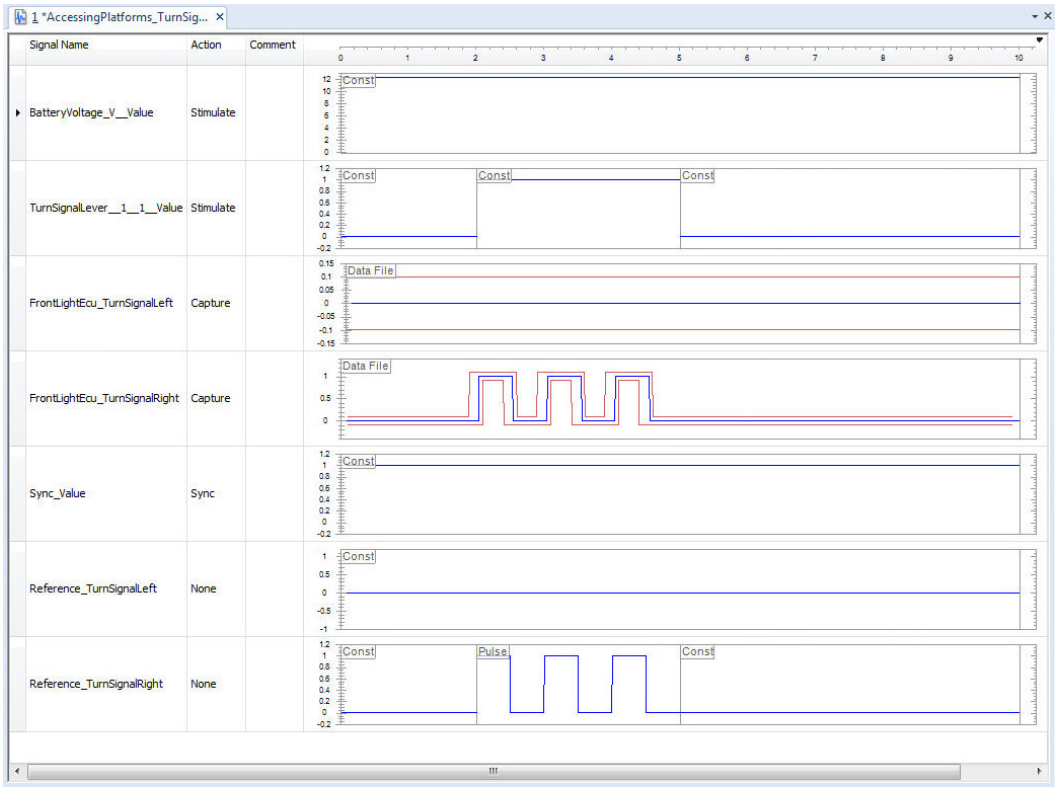
If you performed all the steps in this lesson, you configured a signal-based test and executed it.

You added the SignalBasedTesting folder to your project that contains the TurnSignalSBT sequence which implements a standard test routine. The test routine is configured via signals that are contained in an STZ file. Its file name and path are parameterized in the sequence's TestCaseDescription data object. To configure the platform access, the data objects MAPortConfiguration, Mapping and TaskName that you created in the previous lesson are used.

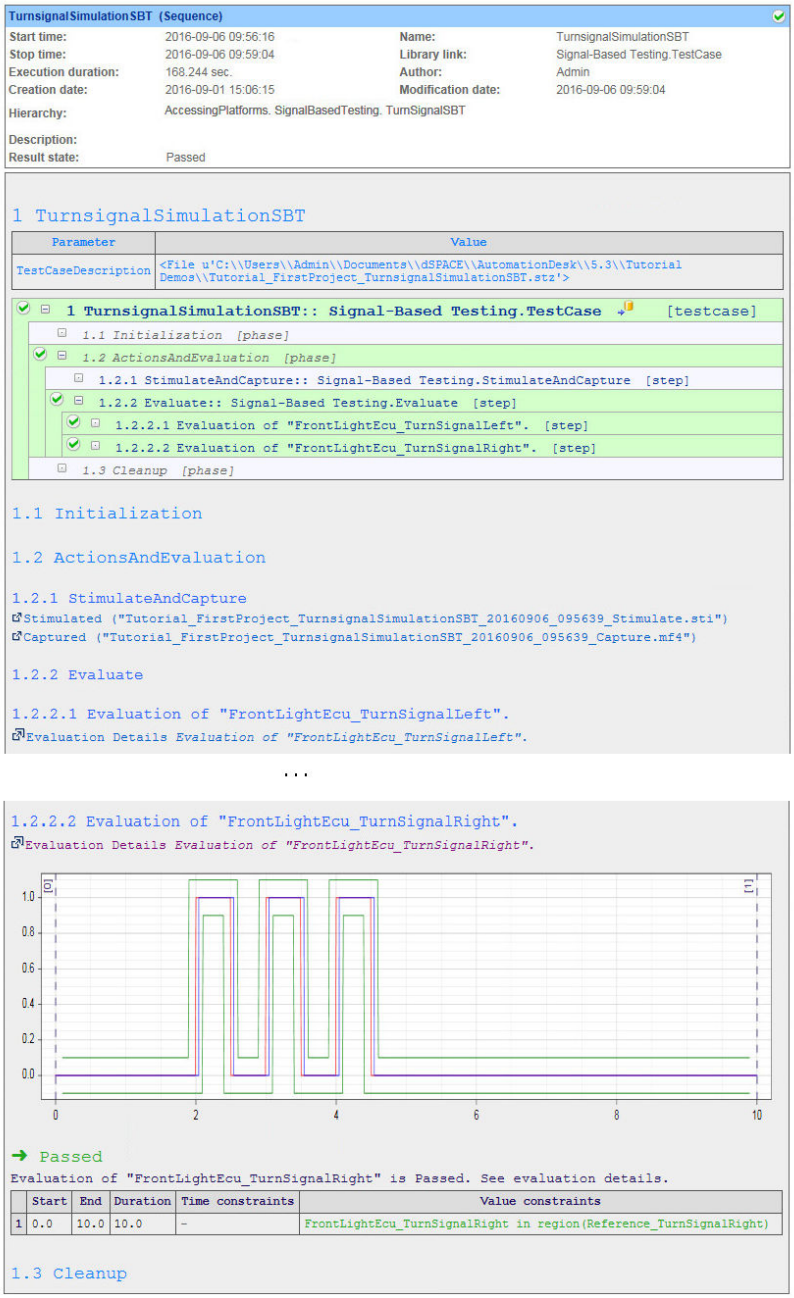


You learned how to use the Signal Editor to configure the signals of a test case description. You created the related STZ file that contains the description of the signals used during the test for stimulating and capturing simulator variables. You also configured reference signals and evaluation methods to compare the captured data with expected data and to issue a verdict on whether or not the test was passed.

Your test case description in `AccessingPlatforms_TurnSignalSBT.stz` configures a constant battery voltage and a turn-signal lever that is switched to the right and back (identically to the test case that is implemented as a control-flow-based test in the previous lesson). The state of the front lights is captured and compared to the expected behavior that is specified via their reference signals.



Executing the **SignalBasedTesting** project folder leads to the following report:



You can see that the result state for all test steps and phases is Passed (✓). The right front turn signal reacts within the tolerances.

Prepared Demo

If you had problems while working through this lesson, or if you want to compare your results with the result we wanted to achieve, you can look at the prepared demo project *Tutorialdemo08.adpx* in `<DocumentsFolder>\Tutorial Demos`.

Using other platforms

You can also work through this lesson if you do not have the same real-time hardware as used here. For more information, refer to Using other platforms on page 157.

Further information

- For more information on using the **Signal-Based Testing** library, refer to Implementing Signal-Based Tests (AutomationDesk Implementing Signal-Based Tests 🕮).
- For more information about the automation blocks used, refer to the Signal-Based Testing Library (AutomationDesk Implementing Signal-Based Tests 🕮).

What's next

The summary (refer to Summary on page 181) gives you a short overview of the lesson results in this tutorial.

Summary

Your Working Results

What's next

You have finished your work successfully. The AutomationDesk project you created covers basic instructions and some advanced features. You are now able to derive the concepts of AutomationDesk from the various lessons and apply them to your daily work.

Further Examples

Demo projects

If you need further examples of using automation blocks, you can refer to the AutomationDesk demo folder `<DocumentsFolder>`.

Note

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

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

@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 Bowser, you always use this folder in combination with the related library file (ADLX).

@ADPX folder A file system folder with the name `<ProjectName>@adpx` that stores information on the elements in the related project, such as sequence files, attached files, results, and reports.

In the Project Manager, you always use this folder in combination with the related project file (ADPX).

@BLKX folder A file system folder with the name `<ElementName>@blkx` that stores information on the subelements in an exported element, such as sequence files, attached files, Python modules, or packages.

You always use this folder in combination with the related parent element file (BLKX).

A

ADL file An AutomationDesk legacy library file that contains the specification of a custom library which was saved to the file system.

These files were created using AutomationDesk 6.1 or earlier. You can open and migrate them to library XML files (ADLX).

ADL.ZIP file An AutomationDesk legacy archive file that contains the specification of a custom library which was managed under version control.

These files were created using AutomationDesk 6.1 or earlier and must be migrated to let you continue to work under version control.

ADLX file An AutomationDesk library XML file that contains the specification of a saved or exported custom library. The ADLX file is located in the same folder as the @ADLX folder.

You can open, edit, save, import, and export ADLX files via the Library Bowser.

ADO file An AutomationDesk display options file that contains information on how a project or library is displayed when it is opened in the AutomationDesk user interface. This includes bookmarks, breakpoints, and the collapse state of folders and blocks.

These files are created when a project or library is saved or closed.

ADP file An AutomationDesk legacy project file that contains a project's specification, its results, and its reports.

These files were created using AutomationDesk 6.1 or earlier. You can open and migrate them to project XML files (ADPX).

ADP.ZIP file An AutomationDesk legacy archive file that contains the specification of a project which was managed under version control.

These files were created using AutomationDesk 6.1 or earlier and must be migrated to let you continue to work under version control.

ADPX file An AutomationDesk project XML file that contains the specification of a saved or exported AutomationDesk project. The ADPX file is located in the same folder as the @ADPX folder.

You can open, edit, save, import, and export ADPX files via the Project Manager.

ALX file An AutomationDesk library legacy XML file that contains the specification of an exported custom library.

These files were created using AutomationDesk 6.0 or earlier. You can import ALX files in the Library Bowser.

APX file An AutomationDesk project legacy XML file that contains the specification of an exported AutomationDesk project.

These files were created using AutomationDesk 6.0 or earlier. You can import APX files in the Project Manager.

ASAM AE XIL API An API standard for the communication between test automation tools, such as AutomationDesk, and test benches, such as dSPACE real-time hardware. The notation XIL indicates that the standard can be used for various *in-the-loop* systems, e.g., SIL, MIL, PIL, and HIL. The XIL API standard is defined by the Association for Standardisation of Automation and Measuring Systems (ASAM).

ASAM General Expression Syntax (ASAM GES) The syntax definition that is used in AutomationDesk to specify trigger conditions. It is part of the XIL API standard that is defined by the Association for Standardisation of Automation and Measuring Systems (ASAM).

Automation block A part of a sequence that implements an automation task, similar to a subroutine.

Templates for automation blocks are provided by AutomationDesk libraries. Via the Sequence Builder, you can arrange automation blocks to implement the control flow of your automation task.

AutomationDesk Options A dialog that lets you modify the appearance and behavior of some AutomationDesk panes and the layout of the generated reports.

Automotive Simulation Model (ASM) The dSPACE product that provides open MATLAB®/Simulink® models that are relevant for the simulation of automotive engines (gasoline and diesel) and vehicle dynamics.

B

BLKX file An AutomationDesk element XML file that contains the specification of a saved or exported AutomationDesk element.

You can import and export BLKX files in the Project Manager, the Sequence Hierarchy Browser, the Sequence Builder, and the Library Bowser.

Block-specific data object A data object that resides in the interface of an automation block. It can be used to parameterize the block or to return a resulting data object after block execution.

Most blocks provided by AutomationDesk provide a static interface. However, some blocks let you add data objects to their interfaces dynamically, for example, Exec blocks.

Bookmark A label that you can attach to an automation block to use it later for quick navigation within the user interface.

Breakpoint A flag that you can set for a sequence or an automation block that pauses the execution in debug mode when the element with a set breakpoint is reached. You can manually control whether to resume the execution or to terminate it.

Built-in library The type of library that is included in AutomationDesk as a software component.

In contrast to custom libraries, you cannot create your own built-in libraries and you cannot view the library's source code.

C

Capture A data object type of the ASAM AE XIL API that is used to parameterize the capturing of measurement data.

In addition to the model access port (MAPort) to be used and the variables to be captured, you can specify, a condition to start or to stop data capturing, for example.

CaptureResult A data object type of the ASAM AE XIL API that is used to handle the captured data. It contains the time stamps and the related measured values of the captured variables.

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

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

or

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

Component Object Model (COM) An interface in Microsoft Windows that allows software products of different providers to communicate and to control each other.

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

Control-flow-based testing A test strategy that is based on implementing an automation task by specifying its control flow in sequences.

Custom library The type of library that you can create and include in AutomationDesk. The elements that you can add to a custom library are templates for data objects, automation blocks, and sequences. You can use the library elements as templates by adding library links to projects or sequences.

Some predefined custom libraries are part of the AutomationDesk product. They are read-only by default.

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 Bowser.

An element is displayed as an icon that reflects the element's type followed by the element's name.

Error configuration file A file in XML format that contains the specification of the simulated electrical errors as a series of states which are each specified via an error set.

Error set A list of electrical errors that occur to the signals at the same time and that specifies the simulated state of the wiring. An empty error set specifies a state with no errors.

Exec block An automation block that is specified by the Python script to be executed.

You can edit the script via AutomationDesk's Python Editor.

F

FDX file An AutomationDesk project folder legacy XML file that contains the specification of an exported AutomationDesk project folder.

These files were created using AutomationDesk 6.0 or earlier. You can import FDX files in the Project Manager.

H

Hyperlink A click-able reference. When you click the link, the target is opened in an appropriate component.

I

Input dialog A dialog window that demands a manual input.

Instance description The property of an instantiated element that contains a text which describes the element's purpose.

L

LabeledValue A type of data object for which you can define a dictionary of valid label-value pairs. LabeledValues can be set either by specifying a label or by specifying a value.

LFX file An AutomationDesk library folder legacy XML file that contains the specification of an exported library folder.

These files were created using AutomationDesk 6.0 or earlier. You can import LFX files in the Library Bowser.

Library A container for templates that you can use to instantiate data objects, sequences, or automation blocks in your projects.

Libraries are handled via the Library Bowser. Each library is organized as a tree and can be structured using library folders.

There are built-in libraries and custom libraries.

Library Bowser A pane in AutomationDesk that provides access to the elements of the open libraries.

Library folder An element that structures the contents of a library as a tree.

Library link A type of element that you can create in a project or sequence. This type of element is linked to a template in a library. Depending on the link mode, the library link represents an instance of the linked library element or a reference to this library element.

Library links let you reuse a library element at multiple positions in one or multiple projects.

Link mode The way in which an instantiated object in your project can be connected to its related template in the library.

The link mode determines the synchronization behavior after you modified an object's template.

The following link modes are available:

- *Dynamically linked* - A modification of the template takes immediate effect.
- *Statically linked* - A modification of the template takes effect after you manually synchronized it.

If you break the link between an instantiated object and its template, the object becomes independent from the template and cannot be linked again.

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

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

M

Mapping (For XIL API Testbench only) An object type of the ASAM AE XIL API for a data object that contains a mapping of variable aliases to their model paths in the related simulation application.

Only one **Mapping** data object is supported per project and it always resides at the top level of the object hierarchy.

Mapping Editor (For XIL API Framework only) A component that lets you configure an XIL API Framework. This includes, for example, the mapping of aliases to model paths, which you can use in your test cases and which are required to access variables via a model access port.

The configuration is saved to a framework configuration file (XML) as well as related port configuration and mapping files.

Mapping Viewer A pane that displays the contents of the used variable mapping.

If you are working with an XIL API Framework, the mapping relates to the framework configuration, which you can edit via the Mapping Editor.

If you are working with the XIL API Testbench, the mapping relates to the project's Mapping data object, which you can edit in the Mapping Viewer.

MAT file A file that contains measurement data in a format that allows data exchange with MATLAB.

MDF file A file that contains measurement data in a format that complies with the ASAM Common MDF standard. For version 4.1 of this standard, the file name extension is MF4.

Message dialog A dialog that demands manual confirmation for a message, error, warning, or information.

Message Viewer A pane that displays the history of all error and warning messages that occur while you are working with AutomationDesk.

MF4 file Refer to MDF file.

Model access port (MAPort) A data object type of the ASAM AE XIL API that is used to provide access to the variables of a running simulation application.

ModelDesk The dSPACE software product for parameterizing ASM models via graphical representations of the modeled components and controlling the related real-time simulation, offline simulation, or MATLAB®/Simulink® simulation.

MotionDesk The dSPACE software product that lets you visualize the movement of 3-D objects controlled by a running simulation application.

O

Object hierarchy The hierarchy tree that is built by all objects that are instantiated in a specific project.

Offline simulation application (OSA) A simulation application that can be executed without real-time hardware on a host PC with VEOS. The OSA file that implements the simulation application can be built from a Simulink model by the VEOS Player.

Operation mode A feature that is provided by some libraries and lets you decide whether to work online with the related device or to work with previously recorded data.

Operation signal The signal type of signals that are specified as an arithmetic operation (addition or multiplication) of two other signals.

Output Viewer A pane that displays all output messages generated by AutomationDesk.

P

PADL.ZIP file An AutomationDesk legacy element archive file that contains the specification of a custom library, a library folder, or a template which was managed under version control.

These files were created using AutomationDesk 6.1 or earlier and must be migrated to let you continue to work under version control.

PADP.ZIP file An AutomationDesk legacy element archive file that contains a project, a project folder, a sequence, or a result which was managed under version control.

These files were created using AutomationDesk 6.1 or earlier and must be migrated to let you continue to work under version control.

Pane The section of a window that provides related controls. AutomationDesk panes are arranged in view sets.

Parameter Any variable type that can be calibrated.

PCONFIG file An ASAM AE XIL API EES port configuration file that provides the hardware-dependent information for an electrical error simulation (EES) in XML format.

Platform A software component representing a simulator where a simulation application is computed in real-time (on dSPACE real-time hardware) or in non-real-time (on VEOS).

Platform Manager A software component that is commonly used by various dSPACE products to register and access platforms and to control the execution of simulation applications on the platforms.

Project A container for all instantiated resources that implement a specific automation task.

Projects are handled via the Project Manager. Each project is organized as a tree and can be structured using project folders.

Project folder An element that structures the contents of a project as a tree.

Project Manager A pane in AutomationDesk that provides access to the elements of the open projects.

Project-specific data object A data object that is created within a Project or a Project folder in the Project Manager. It can be used to parameterize elements a lower level in the object hierarchy.

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

Python Editor A component that lets you edit the Python scripts for Exec blocks, their templates, and Python modules and packages that are integrated in AutomationDesk libraries. Each of these elements can be opened in a separate Python Editor pane.

R

Real-time application An application that can be executed in real time on dSPACE real-time hardware. A real-time application can be built from a Simulink model containing RTI blocks, for example.

Real-Time Testing (RTT) The dSPACE software product that provides components for creating and executing Python scripts which run on the real-time hardware in parallel to the real-time application.

Record depth The attribute of an execution that specifies which project elements are to include in the execution's result depending on the element's result levels.

The following record depths are provided:

- No result
- High elements only
- High and medium elements

Report A document in PDF or in HTML format that is generated from an execution's result.

Result A set of data that results from the execution of a project, a project folder, or a sequence.

From a result, you can generate a report.

Result Browser A component that displays the result of the execution of a project, a project folder, or a sequence during the execution in form of a tree of the involved data objects and their values .

Each result that you open in the Result Browser is displayed in a separate pane.

Result level The attribute of an element that specifies whether to include the element in an execution's result, depending on the execution's record depth.

AutomationDesk provides the **None**, **Medium**, and **High** result levels.

Result parameter The attributes that specify whether an element is included in an execution's result. For this, AutomationDesk provides the result level and the record depth attributes.

Root element The top-level element of a tree data structure. A root element represents the entire element tree of a project in the Project Manager or a library in the Library Bowser, for example.

S

Segment signal The signal type of the signals that are specified as a sequence of signal segments.

Sequence The implementation of an automation task as a control flow specified with automation blocks.

Sequences are edited via the Sequence Builder.

Sequence Builder A component that lets you graphically edit the control flow of a sequence, sequence template or subsequence template. Each of these elements can be opened in a separate Sequence Builder pane.

Sequence Hierarchy Browser A pane in AutomationDesk that provides access to the elements of the sequence that is currently displayed in the Sequence Builder.

SequenceFrame A template that is provided by the **Framework Builder** built-in library and that lets you specify a predefined frame for implementing similar sequences.

SFX file A sequence frame legacy XML file that contains the specification of an exported SequenceFrame object.

These files were created using AutomationDesk 6.0 or earlier. You can use the Project Manager to import SFX files for handling instantiated sequence frames and the Library Bowser for handling sequence frame templates.

Signal The specification or measurement of the change of a value over time.

Signals can be specified by their shape in a signal description set as a segment signal or as an operation signal.

Signal description set A container for a set of signal specifications that implement a specific signal-based test.

Signal description sets are handled via the Signal Editor as a table of the contained signals.

Signal Editor A component that lets you graphically edit a signal description set as a table of its contained signals.

Multiple signal description sets can be opened in separate Signal Editor panes.

Signal file A file in CSV format that defines via failure classes which electrical errors can be simulated by the specific EES hardware.

Signal generator A software component, that can be configured and controlled via a data object in AutomationDesk. A signal generator can be downloaded to a platform and stimulate variables in a running simulation application in real-time.

Signal segment One member in the sequence of segments that builds a segment signal. A segment is specified by its type and by its other properties.

The segment type is specified at the segment's creation via the Signal Selector. Its other properties can be specified via the Signal Editor or the Properties panes.

Signal Selector The pane that provides elements to add segment signals, operation signals, and segments of various segment types to your signal description set by dragging them to the Signal Editor.

Signal-based testing A test strategy that is based on implementing an automation task by using templates of the **Signal-Based Testing** library and specifying all involved signals in a signal description set.

Simulation application The generic term for offline simulation application (OSA) and real-time application.

SQX file An AutomationDesk sequence legacy XML file that contains the specification of an exported AutomationDesk sequence.

These files were created using AutomationDesk 6.0 or earlier. You can use the Project Manager to import SQX files for handling instantiated sequences and the Library Bowser for handling sequence templates.

Stylesheet An XSL file that specifies the layout for the generation of a report from an execution's result.

STZ file A ZIP file that contains the description of a signal description set in STI format. The STI format is defined by the ASAM AE XIL API standard.

You can create and manage STZ files in AutomationDesk's Signal Editor.

Subsequence An automation block that can contain other automation blocks to implement a part of a sequence's control flow, for example, a loop or a subroutine.

T

Task A thread that is executed on dSPACE real-time hardware.

The execution of tasks is triggered by timer events, I/O events, or software events.

TBX file A block template legacy XML file that contains the specification of an exported library folder.

These files were created using AutomationDesk 6.0 or earlier. You can import TBX files in the Library Bowser.

Template The reusable pattern of a data object, an automation block, or a sequence.

To make a template executable, you must instantiate it as an object in your project.

Template description The property of a template that provides a text which describes the template's purpose.

TSX file A sequence frame legacy XML file that contains the specification of an exported TestSequence (Test Framework) object.

These files were created using AutomationDesk 6.0 or earlier. You can use the Project Manager to import TSX files for handling instantiated sequence frames and the Library Bowser for handling TestSequence templates.

U

User function The call of an external program that you can integrate in AutomationDesk's user interface.

V

Value Editor A component that opens a modal Input dialog to edit the selected data object's value.

The appearance of the dialog depends on the type of the selected data object.

Variable A parameter in the simulation application that can be read and written.

A parameter identified by its variable path.

Variable description file The SDF file, the RTA file, or the OSA file that contains the specifications for an executable simulation application.

Variable path The path to the variable in the hierarchy of the model from which the simulation application is built.

Variables pane A component that lets you edit the configuration of an model access port (MAPort). You can select the platform type to be accessed and specify the variable description file to be used. Then you can browse the tree of the provided model variables. Each MAPort configuration can be opened in a separate Variables pane.

Variant A type of data object that can reference other data objects of any type.

VEOS A dSPACE software product that can execute offline simulation applications on a HostPC independently of real time. No real-time hardware is required.

Verdict A type of data object that is used to qualify the current success status of a sequence, subsequence, or automation block.

View set A configuration of the screen arrangement. You can create various view sets and switch between them. By default, AutomationDesk provides the preconfigured view sets **Sequences**, **Signals**, and **Execution**.

VirtualCOM An interface object for handling AutomationDesk's COM objects. VirtualCOM ensures a proper cleanup of deleted objects in AutomationDesk's namespace.

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.