ModelDesk

Testing

For ModelDesk 5.5

Release 2021-A – May 2021

dSPACE

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

Contents

About This Document 7

Basics and Instructions 9

Basics of ModelDesk Testing ... 10
- Features of ModelDesk Testing ... 10
- Requirements of the Simulation Model ... 11
- Basics of Logical and Concrete Test Cases ... 12
- Basics of Validation and Reporting ... 13
- Workflow of Testing ... 16

Preparing the Tests ... 17
- How to Configure the File Names for Captures and Reports ... 17
- How to Create a Test and Logical Test Cases ... 19
- How to Create Concrete Test Cases ... 20
- How to Create a Validation Function for Evaluation ... 22
- How to Use a Python Script for a Custom Validation ... 22
- Implementing a Python Script for Validation ... 24
- How to Create a Validation Function for a Report ... 27

Working with Alias Variables ... 29
- Basics of the Alias Support ... 29
- How to Create Alias Variables ... 30

Executing and Validating the Test Cases ... 33
- How to Check the Consistency of the Test ... 33
- How to Execute Tests ... 34

Reference Information 37

Testing Commands ... 38
- Activate ... 39
- Activate & Open Plotting ... 40
- Activate & Open Parameter Set ... 40
- Append ... 41
- Check Consistency ... 41
- Configure Variables ... 42
- Delete ... 43
- Duplicate ... 43
- Execute ... 44

Insert ... 45
Load Data - Load <Document> ... 45
New ... 46
Open ... 46
Open from Pool ... 47
Open Road ... 48
Open Scenario ... 48
Remove Markings ... 49
Save ... 49
Save All ... 50
Save As ... 50
Settings ... 51
Show Context (Concrete Test Case) ... 52

Testing Panes and Dialogs ... 53
Concrete Test Cases Pane ... 53
Logical Test Cases Pane ... 54
Test Settings Dialog ... 55
Variables Configuration Dialog ... 56
Validation Pane ... 56

Testing Properties ... 59
Concrete Test Case Properties ... 59
Logical Test Case Properties ... 60
Test Properties ... 61
Validation Function Properties ... 62

Signal Class ... 63
Signal Class Description ... 63
all (Signal) ... 65
any (Signal) ... 65
Binary Operators ... 66
Slicing (Signal) ... 68

Alias Support Commands and Panes ... 70
Add Alias ... 70
Alias Overview ... 71
Remove Alias ... 72
Show Context ... 73

Automation 75

General Information on Automation of Testing ... 76
Basics on the Automation of Testing ... 76

Example of Automating Testing ... 76
Overview of the Object Model of Testing ... 77

Classes for Testing ... 79

ConcreteTestCases ... 79
Class Description (ConcreteTestCases) ... 80
Add ... 80
Item ... 81

ConcreteTestCase ... 82
Class Description (ConcreteTestCase) ... 82
Execute ... 83

LogicalTestCases ... 84
Class Description (LogicalTestCases) ... 84
Add ... 85
Item ... 86

LogicalTestCase ... 86
Class Description (LogicalTestCase) ... 87
Execute ... 87

Test ... 88
Class Description (Test) ... 88
Execute ... 89

Constants for Testing ... 91
Constants for Testing ... 91

General Information on Automation of Alias Support ... 92
Overview of the Object Model for the Alias Support ... 92

Classes for Alias Support ... 94

Aliases ... 94
Class Description (Aliases) ... 95

List ... 95
Class Description (List) ... 95

Lists ... 96
Class Description (Lists) ... 97
Item ... 97
ItemByContext ... 98

Reference ... 99
Class Description (Reference) ... 99
GetValue ... 100

References..... 101
Class Description (References)..... 101
Item..... 102

Variable..... 102
Class Description (Variable)..... 103
SetValue..... 103

Variables..... 104
Class Description (Variables)..... 104
Item..... 105

Index 107

About This Document

Contents

This document introduces you to the testing feature of ModelDesk.

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.
	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

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.

Basics and Instructions

Where to go from here

Information in this section

Basics of ModelDesk Testing 10
Provides you basic information on the testing feature in ModelDesk.

Preparing the Tests 17
You must prepare and configure the tests before you can execute them.

Working with Alias Variables 29
An alias variable allows you to modify properties of the environment for tests and one or more variables at once via automation scripts.

Executing and Validating the Test Cases 33
When the test cases are prepared, you can execute them.

Basics of ModelDesk Testing

Introduction

The following topics provides you basic information on the testing feature in ModelDesk.

Where to go from here

Information in this section

Features of ModelDesk Testing..... 10
ModelDesk testing allows you to specify and execute test cases within ModelDesk.

Requirements of the Simulation Model..... 11
The simulation model used for testing must fulfill some requirements.

Basics of Logical and Concrete Test Cases..... 12
A test can contain several logical test cases. A logical test case can contain several concrete test cases.

Basics of Validation and Reporting..... 13
You can specify validation functions to validate the test results and to provide more information in the test report.

Workflow of Testing..... 16
This overview shows the workflow for testing with the ModelDesk testing component.

Features of ModelDesk Testing

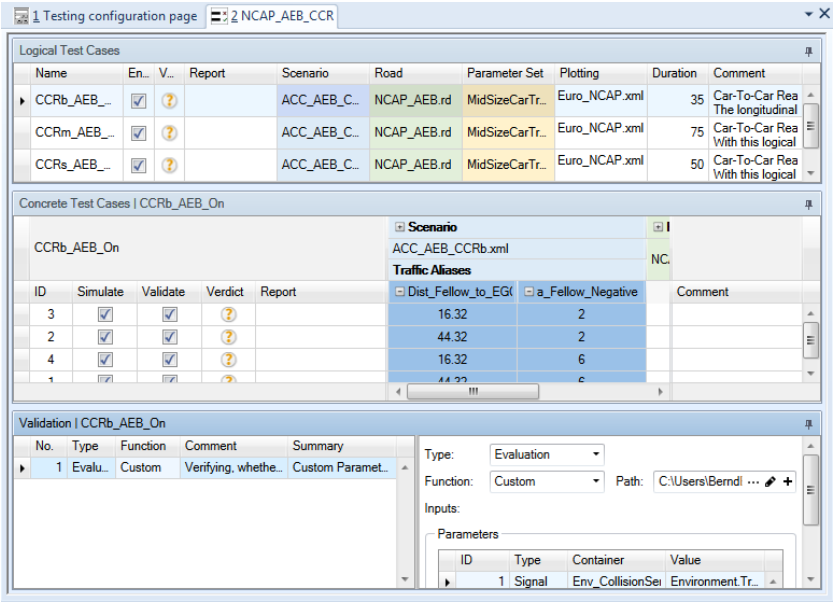
Introduction

ModelDesk testing allows you to specify and execute tests in ModelDesk.

Features

The ModelDesk testing component provides the following features:

- Used for testing based on ASM.
- Supporting all simulation platforms (Simulink, VEOS, and real-time platforms).
- Graphical user interface to specify the test cases.
- Starting the simulation of all the specified test cases at once.
- Validating and reporting the results of the test cases.



Requirements of the Simulation Model

Introduction

The simulation model used for testing must fulfill some requirements.

Requirements

Required ASM blocks To be able to control the simulation, the simulation model must contain the following ASM blocks:

- Maneuver_Start
- Maneuver_Stop
- Maneuver_State

Required signals To get information on the maneuver state, the following signals must be included in ASMSignalBus and the plotting configuration:

- ManeuverState (signal of Maneuver_Scheduler block)
- ManeuverTime (signal of Maneuver_Scheduler block)

Both signals must be connected to an ASMSignalInterface block in the simulation model. The label of this block and the path and name of the signals must be specified in the Environment Configuration dialog on the Testing page.

Note

A real-time application can consists of several application processes, for example, when it is built for a multi-processing-unit system. In this case, the signals used for triggering the plotting and the ManeuverState/ManeuverTime signals must come from the same application process. Otherwise, triggering is not possible.

Related topics

Basics

ASMSignalBus (ASM User Guide 🕮)
Collecting Signals for Plotting in the ASM Model (ModelDesk Plotting 🕮)

References

ASMSignalInterface (ASM User Guide 🕮)
Environment Configuration Dialog (ModelDesk Scenario Creation 🕮)
Maneuver Scheduler (ASM Environment Reference 🕮)
Maneuver Start (ASM User Guide 🕮)
Maneuver State (ASM User Guide 🕮)
Maneuver Stop (ASM User Guide 🕮)

Basics of Logical and Concrete Test Cases

Introduction

A test can contain several logical test cases. A logical test case can contain several concrete test cases.

Logical test cases

Logical test cases specify the environment of the tests. The environments can consist of the following elements:

- Scenarios that specify the maneuver of the vehicle and the movement of the fellows if used.
- Roads that specify the underground and sceneries used in the tests.
- Parameter sets that specify the parameters of the vehicle used in the tests
- Plotting configurations that specify the signals that are measured during the tests and that can be used to evaluate the test results and specify the triggers to start and stop measurements

Plotting configurations are required for a test. Scenarios, roads, and parameter sets are optional.

The elements are specified in ModelDesk and saved in documents available in the Pool of the ModelDesk project.

The following illustration shows an example of logical test cases.

Logical Test Cases

Name	En...	V...	Report	Scenario	Road	Parameter Set	Plotting	Duration	Comment
CCRb_AEB_...	☑	?		ACC_AEB_C...	NCAP_AEB.rd	MidSizeCarTr...	Euro_NCAP.xml	35	Car-To-Car Rea The longitudinal
CCRm_AEB_...	☑	?		ACC_AEB_C...	NCAP_AEB.rd	MidSizeCarTr...	Euro_NCAP.xml	75	Car-To-Car Rea With this logical
CCRs_AEB_...	☑	?		ACC_AEB_C...	NCAP_AEB.rd	MidSizeCarTr...	Euro_NCAP.xml	50	Car-To-Car Rea With this logical

Concrete test cases

A concrete test case is based on a logical test case. It is used to specify concrete values of the variables that are modified during the tests.

To modify variables of the scenarios or roads, you must use alias variables that reference these variables. For details on creating alias variables, refer to Working with Alias Variables on page 29.

To modify parameters of the vehicle no special action is necessary. You can access all scalar parameters that are available in the parameter set.

The following illustration shows an example of concrete test cases.

Concrete Test Cases | CCRb_AEB_On

CCRb_AEB_On					Scenario		
					ACC_AEB_CCRb.xml		NC.
					Traffic Aliases		
ID	Simulate	Validate	Verdict	Report	Dist_Fellow_to_EG(	a_Fellow_Negative	Comment
3	☑	☑	?		16.32	2	
2	☑	☑	?		44.32	2	
4	☑	☑	?		16.32	6	
1	☑	☑	?		44.32	6	

Related topics

HowTos

How to Create a Test and Logical Test Cases..... 19
How to Create Concrete Test Cases..... 20

Basics of Validation and Reporting

Introduction

You can specify validation functions to validate the test results and to provide more information in the test report.

Evaluation

During execution of the concrete test cases, evaluation functions are called that are assigned to the same logical test case. These evaluation functions can be used to observe the signals and check whether the test case was completed without errors. Each evaluation function is executed for every concrete test case.

ModelDesk provides the following predefined operations that you can use to specify the evaluation function. You can also write a Python script for the evaluation:

Custom To check the validation using a Python script. Refer to How to Use a Python Script for a Custom Validation on page 22.

Is Above Bound To check whether a signal value exceeds a reference value.

Is Below Bound To check whether a signal value is below a reference value.

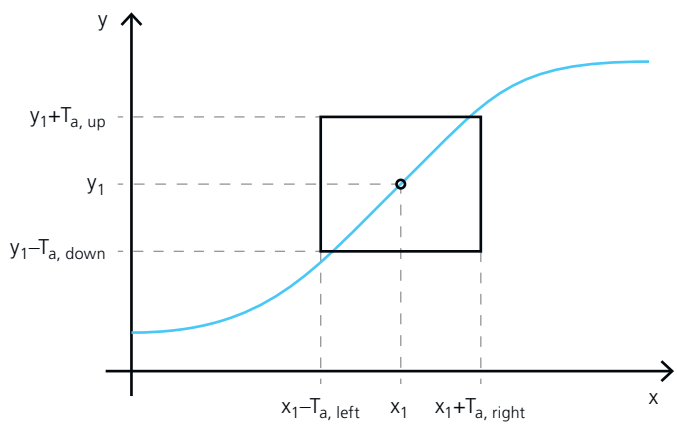
Is Inside Bounds To check whether a signal is within a value range.

Is Outside Bounds To check whether a signal is outside of a value range.

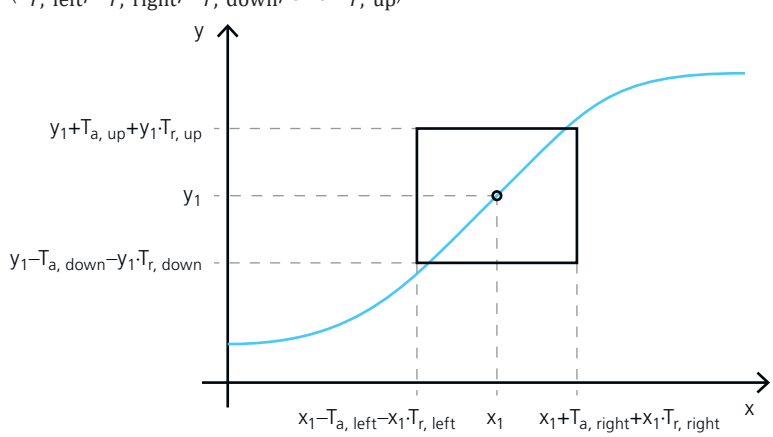
Is Equal To check whether the difference of a signal to a reference value is within a tolerance range.

Is Not Equal To check whether the difference of a signal to a reference value is outside a tolerance range.

Is Inside Region To check whether a signal value is within a range around a reference value specified. The region is specified by absolute tolerance values ($T_{a,\ \mathrm{left}}$, $T_{a,\ \mathrm{right}}$, $T_{a,\ \mathrm{down}}$, and $T_{a,\ \mathrm{up}}$).

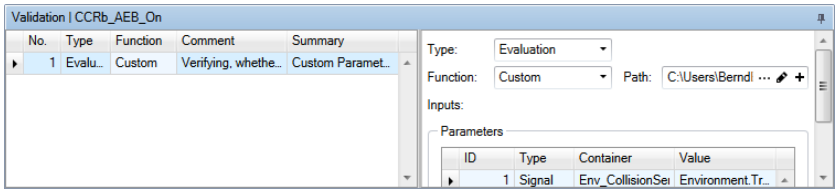


Is Inside Dynamic Region To check whether a signal value is within a range around a reference value specified. The region is specified by absolute tolerance values ($T_{a,\ \mathrm{left}}$, $T_{a,\ \mathrm{right}}$, $T_{a,\ \mathrm{down}}$, and $T_{a,\ \mathrm{up}}$) and relative tolerance values ($T_{r,\ \mathrm{left}}$, $T_{r,\ \mathrm{right}}$, $T_{r,\ \mathrm{down}}$, and $T_{r,\ \mathrm{up}}$).



The signals that are observed, must be available in the selected plotting configuration or a MAT file that contains signals captured in a previous simulation.

The following illustration shows an example of an evaluation function.



Report

After a validation, reports are written. You can use report functions to add additional information to the reports.

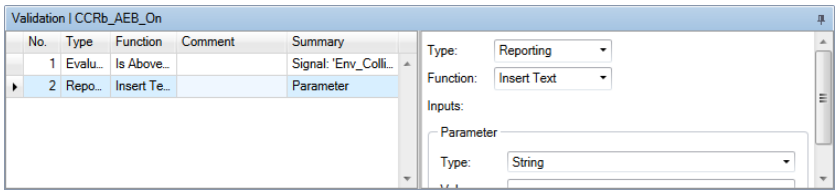
ModelDesk provides the following kinds of functions that insert information into the report.

Insert Text To insert a text into the report.

Insert Image To insert an image into the report.

Plot 2D To insert a two-dimensional plot of a signal into the report.

The following illustration shows an example of a report function.



Limitation A real-time application can consist of several application processes, for example, when it is built for a multi-processing-unit system. If ASMSignalCollector blocks with the same label are used in different application processes of such a system, their signals cannot be used for a validation function. In this case, the signals used for triggering the plotting and the ManeuverState/ManeuverTime signals must come from the same application process. Otherwise, triggering is not possible.

Related topics

HowTos

How to Create a Validation Function for a Report.....27
How to Create a Validation Function for Evaluation.....22

Workflow of Testing

Introduction

This overview shows the workflow for testing with the ModelDesk testing component.

Workflow

To perform the tests, you have to do the following steps:

1. Prepare the test environment. The settings of the environment must be ready to use and stored in the Pool of the ModelDesk project.
   - Scenarios and roads: When you want to modify variables of the scenarios or road during the test, you must create alias variables for them. Refer to Working with Alias Variables on page 29.
   - Parameter set of the vehicle: You can modify all the scalar parameters of the parameter set.
   - Plotting configurations: The plotting configuration must contain the variables that you want to use for evaluating the test or the report. Refer to Introduction to the Plot Manager (ModelDesk Plotting 🕮).
2. Create the test and specify logical test cases. Refer to How to Create a Test and Logical Test Cases on page 19.
3. Specify the concrete test cases. Refer to How to Create Concrete Test Cases on page 20.
4. Specify the validation functions. Refer to How to Create a Validation Function for Evaluation on page 22 and How to Create a Validation Function for a Report on page 27.
5. Check the consistency of the test. Refer to How to Check the Consistency of the Test on page 33.
6. Execute the test. Refer to How to Execute Tests on page 34.

Related topics

Basics

Basics of Logical and Concrete Test Cases..... 12
Basics of Validation and Reporting..... 13
Features of ModelDesk Testing..... 10

Preparing the Tests

Introduction

You must prepare and configure the tests before you can execute them.

Where to go from here

Information in this section

How to Configure the File Names for Captures and Reports..... 17
You can configure a template for the file names of captures and reports. Then the capture and report files are named according to the configured template.

How to Create a Test and Logical Test Cases..... 19
A test contains one or more logical test cases.

How to Create Concrete Test Cases..... 20
You use concrete test cases to vary the values of parameters for tests.

How to Create a Validation Function for Evaluation..... 22
To evaluate the test results, you can specify validation functions that compare the signal values with reference values.

How to Use a Python Script for a Custom Validation..... 22
You can use your own Python script for validating the test results.

Implementing a Python Script for Validation..... 24
If the standard functions in ModelDesk testing are not sufficient, you can write your own Python script that checks the test results.

How to Create a Validation Function for a Report..... 27
To get more detailed reports of the test results, you can add text, images, and two-dimensional plots of signals to the report.

How to Configure the File Names for Captures and Reports

Objective

You can configure a template for the file names of captures and reports. Then the capture and report files are named according to the configured template.

Basics

You can create a template for file names of the captures and reports. The template can consist of several elements that are defined by placeholders. The placeholders are inserted by drag & drop.

Captures The template always starts with the index of the concrete test case. You can add the time/date and a free text to the template. The elements can be specified in any order. They are separated by underscores. If the template has the

Free text element, you must specify a text for it. If you add time/date to the file name, new captures do not overwrite previous captures.

Reports The template can consist of four different elements: Time/date, logical text case name, test name, and free text. The elements can be specified in any order. They are separated by underscores. If the template has the *Free text* element, you must specify a text for it. If you add time/date to the file name, new reports do not overwrite previous reports.

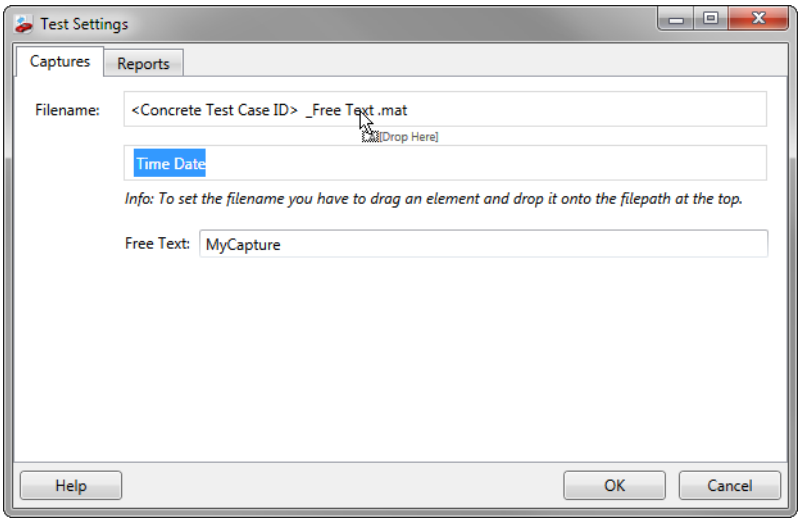
Method

To configure the file names for captures and reports

1 On the Testing ribbon, click Test – Settings.

The Test Settings dialog opens.

2 To use an element for the file name, drag it from the row below the file name to the required position in the row of the file name. Refer to the following illustration.



Tip

To delete an element of the template, drag it from the file name row to the row below.
To modify the order of the elements, delete the elements, and drag them to the new position.

3 If you use the *Free Text* element, specify a text.

Result

The templates for the file names are configured. When the test case is executed, the files of captures and reports are named according to the configured template.

Related topics

References

Settings.....51
Test Settings Dialog.....55

How to Create a Test and Logical Test Cases

Objective

A test contains one or more logical test cases.

Preconditions

The environment of the test must be specified: The scenario, road, parameter set, and plotting configuration to be used in the test case must be stored in the pool.

Method

To create a test with logical test cases

1. On the Testing ribbon, click Test – New.
   ModelDesk creates a new test with one logical test case.
2. Click in the empty area of the Logical Test Cases pane.
   The Properties pane displays the properties of the test.
3. Specify the properties of the test on the Properties pane.
4. Click the row of the logical test case.
   The Properties pane displays the properties of the logical test case.
5. Specify the properties of the logical test case in the row on the Logical Test Cases pane or on the Properties pane.
6. To add more logical test cases, enter properties in the empty row on the Logical Tests Cases pane or select Insert in the context menu.
7. Specify the properties of the new logical test cases in the row on the Logical Test Cases pane or on the Properties pane.
8. On the Testing ribbon, click Test – Save As to specify a name of the test and save it.

Result

A test with logical test cases is created.

Next Step

You can create concrete test cases that base on the logical test case. Refer to How to Create Concrete Test Cases on page 20.

Related topics

Basics

Basics of Logical and Concrete Test Cases..... 12

References

Logical Test Case Properties..... 60
New..... 46
Save As..... 50
Test Properties..... 61

How to Create Concrete Test Cases

Objective

You use concrete test cases to vary the values of parameters for tests.

Preconditions

- The logical test case must be created. Refer to How to Create a Test and Logical Test Cases on page 19.
- If you want to modify variables of the environment (scenario or road), alias variables assigned to these variables must be created. Refer to Working with Alias Variables on page 29.

Method

To create a concrete test case

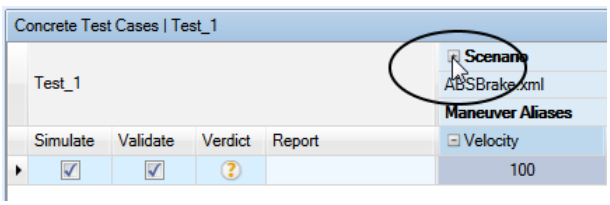
1 On the Logical Test Cases pane, click the logical test case for which you want to create the concrete test cases.

The Concrete Test Cases pane displays already one concrete test case.

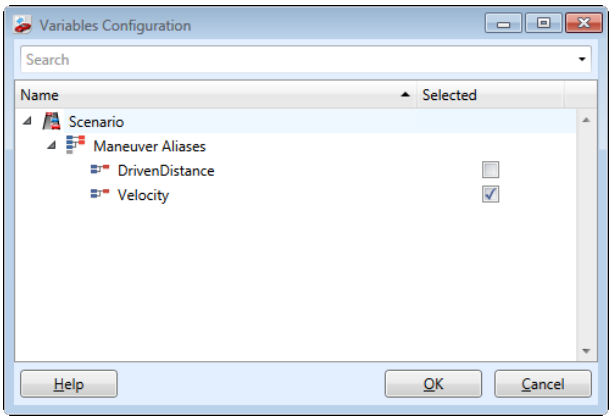
2 Click the concrete test case.

The Properties pane displays the its properties. The variable to be modified is added on the Concrete Test Cases pane.

3 Click the plus sign of the experiment contents to which the variable belongs.



The **Variables Configuration** dialog opens. It displays all the alias variables or model parameters that can be modified during testing.



4 In the **Variables Configuration** dialog select the parameter and click **OK**.

5 Specify a value of the parameter.

6 To modify another parameter in the concrete test case, repeat the previous steps.

7 To add more concrete test cases, open the context menu on the **Concrete Test Case** pane and select **Insert**, **Append**, or **Duplicate**.

ModelDesk adds a further concrete test case. If you use **Duplicate** the concrete test case has the same settings as the selected one.

8 To specify the concrete test case, repeat the steps described above.

9 On the **Testing** ribbon, click **Test** – **Save** or **Save As** to save the test.

Result

A concrete test case is created.

Next Step

You can create validation functions to evaluate the test case. Refer to How to Create a Validation Function for Evaluation on page 22 and How to Create a Validation Function for a Report on page 27.

Related topics

Basics

Basics of Logical and Concrete Test Cases.....12

References

Append.....41
Concrete Test Case Properties.....59
Variables Configuration Dialog.....56

How to Create a Validation Function for Evaluation

Objective

To evaluate the test results, you can specify validation functions that compare the signal values with reference values.

Basics

For basic information, refer to Basics of Validation and Reporting on page 13.

Preconditions

The logical test case must be created. Refer to How to Create a Test and Logical Test Cases on page 19.

Method

To create a validation function for evaluation

1. On the Logical Test Cases pane, select the logical test case for which you want to specify the validation function.
2. Open the Validation pane.
3. Open the context menu and select Insert.

   ModelDesk creates a new validation function.
4. In the Type column, select `Evaluation`.
5. In the Function column, select a function.
6. On the right side of the Validation pane, specify the parameters of the function.

Result

A validation function for the evaluation is specified.

Related topics

Basics

Basics of Validation and Reporting 13

References

Insert 45
Validation Function Properties 62
Validation Pane 56

How to Use a Python Script for a Custom Validation

Objective

You can use your own Python script for validating the test results.

Preconditions

The logical test case must be created. Refer to How to Create a Test and Logical Test Cases on page 19.

Method

To create a Python function for a custom validation

1. On the **Logical Test Cases** pane, select the logical test case for which you want to specify the custom validation function.
2. Open the **Validation** pane.
3. Open the context menu and select **Insert**.

   ModelDesk creates a new validation function.
4. In the **Type** column, select `Evaluation`.
5. In the **Function** column, select `Custom`.
6. On the right side of the **Validation** pane, specify the parameters that you want to use in the Python script. You can use a signal of a plotting configuration, a captured signal of a MAT file, an alias variable of a scenario or road, or a constant value.
7. Click the plus sign to create the Python script.

   ModelDesk opens a **Specify File Name** dialog
8. Specify a file name and click **Save**.

   ModelDesk creates the Python script using a predefined template.
9. Click ✎ to open the script in an Python editor.
10. Implement your validation function. For information, refer to Implementing a Python Script for Validation on page 24.

Result

A custom validation function for the evaluation is specified.

Related topics

Basics

Basics of Validation and Reporting 13

References

Insert 45
Signal Class 63
Validation Pane 56

Implementing a Python Script for Validation

Introduction

If the standard functions in ModelDesk testing are not sufficient, you can write your own Python script that checks the test results.

Basics

ModelDesk provides a template for the Python script. You are recommended to use the template as basis for your Python script that you use for validation. The template imports some modules that can be used for accessing the variables, evaluation, and returning the validation results.

So that ModelDesk can call the custom function, the function definition must be as follows:

```
def CustomFunction(parameters: List[Parameter], result_picture_path_without_extension: str) -> EvaluationResult:
```

The name `CustomFunction` is mandatory.

Two parameters are mandatory. For more information, refer to Passing variables to the custom function on page 24 and Adding a custom image to the report on page 26.

The return value must be an `EvaluationResult` object. For more information, refer to Returning the result on page 26.

Passing variables to the custom function

The `parameters` argument contains a list to parameters. You can pass different types of variables to the Python script via `parameters`:

- Constant type: To pass a constant value.
- Variable type: To pass the value of an alias variable.
- Signal type: To pass a signal that is measured during the test in the plotting configuration.
- Captured signal type: To pass a signal that was measured in a previous simulation and stored in a MAT file.

You must handle the different parameter types in different ways. `ID` is the index in the parameter list used in the following code examples.

Passing a constant type

Setting in ModelDesk On the Validation pane, select the Constant type and specify the value in the Value column.

Accessing the parameter in Python script In the Python script, you can check the parameter type using:

```
parameters[ID].parameter_type is ParameterType.Constant
```

You can access the value as follows:

```
Name = parameters[ID].name
Constant = parameters[ID].value
```

`Name` is a string containing the name of the constant.

`Constant` is a float type.

Passing a variable type

Setting in ModelDesk On the Validation pane, select the Variable type and then the alias variable.

Accessing the parameter in Python script In the Python script, you can check the parameter type using:

```
parameters[ID].parameter_type is ParameterType.Variable
```

You can access the alias variable as follows:

```
Name = parameters[ID].name
List = parameters[ID].list
Variable = parameters[ID].value
```

`Name` is a string containing the name of the alias variable.

`List` is a string containing the name of the list.

`Variable` is a float type.

Passing a signal type

Setting in ModelDesk On the Validation pane, select the Signal type, the layout of the plotting configuration, and then the measured signal.

Limitation for multi-processing application A real-time application can consist of several application processes, for example, when it is built for a multi-processing-unit system. If ASMSignalInterface blocks with the same label are used in different application processes of such a system, their signals cannot be used for a validation function because ModelDesk cannot distinguish between these signals.

Accessing the parameter in Python script In the Python script, you can check the parameter type using:

```
parameters[ID].parameter_type is ParameterType.CurrentSignal
```

You can access the signal as follows:

```
Name = parameters[ID].name
CurrentSignal = parameters[ID].value
```

`Name` is a string containing the name of the signal (full path).

`CurrentSignal` is an object of the Signal class.

Passing a captured signal type

Setting in ModelDesk On the Validation pane, select the Captured signal type, the MAT file and then the captured signal.

Accessing the parameter in Python script In the Python script, you can check the parameter type using:

```
parameters[ID].parameter_type is ParameterType.CapturedSignal
```

You can access the signal as follows:

```
Name = parameters[ID].name
CapturePath = parameters[ID].capture_path
CapturedSignal = parameters[ID].value
```

`Name` is a string containing the name of the signal (full path).

`CapturePath` is a string containing the absolute path of the MAT file.

`CapturedSignal` is an object of the Signal class.

Adding a custom image to the report

You can add an image into the report. To do this, the following steps are required.

1. The `CustomFunction` has the `result_picture_path_without_extension` argument. ModelDesk passes the absolute path and file name without extension via this argument to the `CustomFunction`.

   For example, `%AbsolutePath%\001_001_Custom`.
2. In the `CustomFunction`, create the image and save it under the absolute path and file name passed from ModelDesk. Add the extension of the image format to the file name

   For example, `%AbsolutePath%\001_001_Custom.svg`
3. Return the absolute path and file name with extension using the `EvaluationResult` object. For example:

```
path = result_picture_path_without_extension + '.svg'
result = EvaluationResult(Verdict.Passed,path,'Text')
```

   For more information, refer to Returning the result on page 26.

Working with the Signal class

You can use the `Signal` class to evaluate the test. The class can handle signals of the simulation model and captured signals of a MAT file. Refer to Signal Class Description on page 63.

The class gives you the following possibilities:

all method To test whether all function values (y-axis) of a signal are truthful. Refer to all (Signal) on page 65.

any method To test whether at least one function value (y-axis) of a signal is truthful. Refer to any (Signal) on page 65.

Slicing To get the x-axis or y-axis of a signal. Refer to Slicing (Signal) on page 68.

Binary operators To create an evaluation signal by applying a binary operation to the two input evaluation signals in an element-wise manner. Refer to Binary Operators on page 66.

Returning the result

To return the result of the evaluation, you must use the `EvaluationResult` and `Verdict` classes. To use the classes, they must be imported:

```
from TestingLibrary.Validation.validation_result import EvaluationResult
from TestingLibrary.Validation.validation_result import Verdict
```

EvaluationResult An `EvaluationResult` object specifies the return value. It can contain the following three parameters:

1. Mandatory: Verdict of the evaluation (`Verdict.Passed`, `Verdict.Undefined`, `Verdict.Failed`).

2. Optional: String that contains the absolute path to an image including the file name and extension to be displayed in the report. Refer to Adding a custom image to the report on page 26.
3. Optional: String that you can use to describe the validation result.

Example The following examples shows how you use the `EvaluationResult` to return the result.

```
# To return a passed test:
result = EvaluationResult(Verdict.Passed,'','Text for a passed validation.')
# To return a failed test:
result = EvaluationResult(Verdict.Failed,'','Text for a failed validation.')
# To return an undefined test:
result = EvaluationResult(Verdict.Undefined,'','Text for an undefined validation.')
return result
```

Debugging

You can use an external debugger for debugging the `CustomFunction`.

Related topics

Basics

Implementing a Python Script for Validation 24

HowTos

How to Use a Python Script for a Custom Validation 22

References

Signal Class 63

How to Create a Validation Function for a Report

Objective

To get more detailed reports of the test results, you can add text, images, and two-dimensional plots of signals to the report.

Basics

For basic information, refer to Basics of Validation and Reporting on page 13.

File name

You can configure how to build the file names of the reports. Refer to How to Configure the File Names for Captures and Reports on page 17.

Preconditions

The logical test case must be created. Refer to How to Create a Test and Logical Test Cases on page 19.

Method

To create a validation function for a report

1. On the Logical Test Cases pane, select the logical test case for which you want to specify the validation function.
2. Open the Validation pane.
3. Open the context menu and select Insert.

   ModelDesk creates a new validation function.
4. In the Type column, select `Reporting`.
5. In the Function column, select a function.
6. On the right side of the Validation pane, specify the parameters of the function.

Result

A validation function for the report is specified.

Related topics

Basics

Basics of Validation and Reporting..... 13

References

Insert..... 45
Validation Function Properties..... 62
Validation Pane..... 56

Working with Alias Variables

Introduction

An alias variable allows you to modify properties of the environment for tests and one or more variables at once via automation scripts.

Where to go from here

Information in this section

Basics of the Alias Support.....29
Using alias support, you can modify the values of properties in automation scripts.

How to Create Alias Variables.....30
Alias variables are created in the user interface of ModelDesk.

Basics of the Alias Support

Introduction

Using alias support, you can modify the values of properties in automation scripts.

Modifying properties

Using the automation interface of the **Road Generator** and **Scenario Editor** gives you the option to modify the properties of roads and scenarios.

Another possibility is to create alias variables that are assigned to one or more properties, named property references. If you modify the values of alias variables, the values of their assigned property references are also modified. This method allows you to modify several properties at the same time.

Supported properties

You can assign alias names only for scalar values.

Alias variables can be created only for properties of the same road or scenario.

Workflow

Using the alias support consists of two steps: Creating alias variables and using them in automation scripts:

1. The alias variables are created in the user interface of ModelDesk. In the **Road Generator** and **Scenario Editor**, you select the properties that must be referenced by alias variables. In the **Alias Overview**, you manage the alias variables and property references, for example, you can move property references from one alias variable to another. Refer to How to Create Alias Variables on page 30.

2. When alias variables exist, you can use them in tests and in automation scripts to modify their values. When the values of alias variables are modified, the values of the property references are modified. Refer to How to Create Concrete Test Cases on page 20 and Setting Values of Properties Using Alias Variables (ModelDesk Automation 🕮).

Related topics

Basics

Automating Scenarios in Python (ModelDesk Scenario Creation 🕮)
Modifying a Road in Python (ModelDesk Road Creation 🕮)

How to Create Alias Variables

Objective

Alias variables are created in the user interface of ModelDesk.

Workflow

The workflow is divided into two parts:

- To create alias variables, you start in the Road Generator or Scenario Editor. You have to go all the properties that you want to assign to alias variables and select the Add Alias command. Then the properties are immediately available as property references in the Alias Overview controlbar. Refer to Part 1 on page 30.
- When the property references are in the Alias Overview controlbar, you can move property references to other alias variables, so that alias variables are assigned to several property references. Refer to Part 2 on page 31.

Preconditions

The road or scenario must be activated.

Part 1

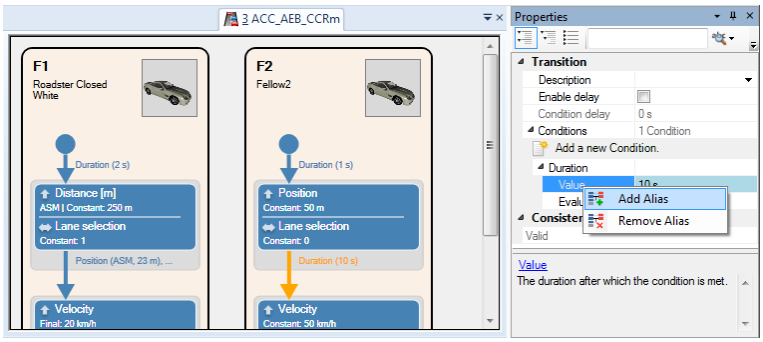
To create alias variables

1. Open the road or scenario which has the property you want to create an alias variable for.
2. In the working view, go to the segment that includes the property to be referenced.

   In the Road Generator and Scenario Editor, the Properties pane shows the property.

3 Open the context menu of the property and select Add Alias.

The following illustration shows an example in the Scenario Editor.



An alias variable is created and assigned to the selected property. The Alias Overview controlbar lists the new alias variable.

4 Optional: In the Alias Overview controlbar, enter a comment for the property reference. Name, type, and description of the property reference come from the property of the road or scenario and cannot be modified. If you want to distinguish different properties, you can use the comment field.

5 Repeat the previous steps for other properties.

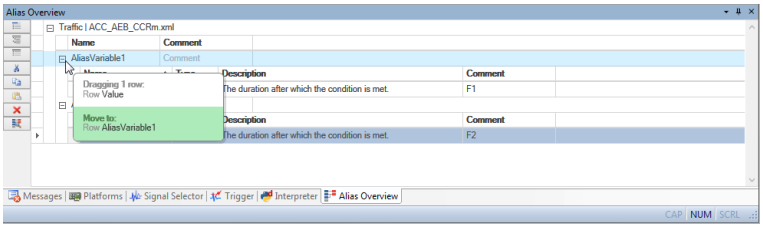
Intermediate result

The alias variables are listed in the Alias Overview. Each alias variable contains one property reference.

Part 2

To assign property references to other alias variables

1 In the Alias Overview, drag a property reference to another alias variable, refer to the following illustration.



The property reference is moved to the alias variable. An alias variable is automatically removed if no property reference is assigned to it.

Tip

- To get the context of a property reference, click .
  The property is highlighted in the editor.
- You can also use the and buttons to cut & paste a property reference to another alias variable.

2 Repeat the previous step for other property references.

3 Specify a name, maximum value, minimum value, and a comment for the alias variable. When you use the alias variable in the automation scripts, a descriptive name, and comment is useful. The maximum and minimum value can be read and written in test scripts. They are not evaluated in the user interface.

Result

You have created alias variables. You can use them in the tests of ModelDesk testing and automation scripts. Refer to Setting Values of Properties Using Alias Variables (ModelDesk Automation 🕮).

Related topics

Basics

Basics of the Alias Support.....29

References

Add Alias.....70
Alias Overview.....71

Executing and Validating the Test Cases

Introduction

When the test cases are prepared, you can execute them.

Where to go from here

Information in this section

How to Check the Consistency of the Test.....33
Before you execute the test, you can check the consistency.

How to Execute Tests.....34
You can execute the complete test, all the concrete test cases of a logical test case, or a specific concrete test case.

How to Check the Consistency of the Test

Objective

Before you execute the test, you can check the consistency.

Basics

Because it is possible that the ModelDesk experiment is modified after you configured the test, you can perform a consistency check. In the consistency check, ModelDesk checks whether all referenced experiment content is still available. This is done for each part of the test, i.e., logical test cases, concrete test cases, and validation functions.

Method

To check the consistency of the test case

1. Open the test to be checked.
2. On the Testing ribbon, click Test – Check Consistency.
   ModelDesk starts the consistency check.
3. To remove the markings of a consistency check, go to the Testing ribbon and click Test – Remove Markings.

Result

The consistency of the test is checked.

Next step

You can start the test execution. Refer to How to Execute Tests on page 34.

Related topics

References

Check Consistency.....41
Execute.....44
Remove Markings.....49

How to Execute Tests

Objective

When a test case is configured, you can execute it.

Preconditions

A test is configured.

Executing the tests

When the test is executed, ModelDesk does the following procedure for each concrete test case that is enabled for simulation:

1. ModelDesk activates the experiment documents that are selected for the logical test case.
2. ModelDesk sets the variables according to the settings of the concrete test case.
3. ModelDesk downloads the content of the experiment and if necessary the application to the platform.
4. ModelDesk starts the maneuver on the platform.
5. ModelDesk starts the measurement according to the plotting configuration to capture the signals. The measurement is started when one of the following conditions is fulfilled:
   - The value of **ManeuverState** (signal the of **Maneuver_Scheduler** block) exceeds the value 2.5.
   - The condition of the start trigger in the plotting configuration is fulfilled.
6. ModelDesk stops the maneuver and the capturing of the signals. The measurement is stopped when one of the following conditions is fulfilled:
   - The value of **ManeuverState** (signal the of **Maneuver_Scheduler** block) exceeds the value 4.5.
   - The condition of the stop trigger in the plotting configuration is fulfilled.
   - The maximum duration of the logical test case is reached.
7. ModelDesk saves the captures.

For the complete test, ModelDesk repeats the procedure for all the enabled logical test cases and concrete test cases selected for simulation.

After the simulation, ModelDesk validates the captures of each concrete test case that is enabled for validation.

1. ModelDesk reads the captures.
2. ModelDesk executes the defined validation functions for each concrete test case.
3. ModelDesk creates a report for the logical test case that contains the test result of all included concrete test cases.

Stopping test execution

You can stop the test execution only during the simulation.

Possible methods

You can execute the complete test, a logical test case, or a concrete test case.

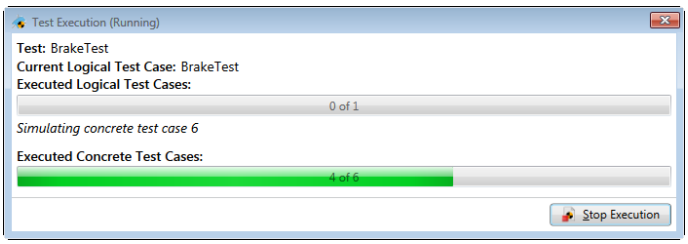
- To execute the complete test, refer to Method 1 on page 35.
- To execute all the concrete test case of a logical test case, refer to Method 2 on page 35.
- To execute a specific concrete test case, refer to Method 3 on page 35.

Method 1

To execute the test

1 On the Testing ribbon, click Test – Execute.

ModelDesk executes the enabled logical test cases. The following dialog opens where you can monitor the progress and abort the test execution.



Method 2

To execute a logical test case

1 On the Logical Test Cases pane, open the context menu of the logical test case and click Execute.

ModelDesk simulates and validates the concrete test cases of the logical test case if the appropriate option is enabled. A dialog opens where you can monitor the progress and abort the test execution.

Method 3

To execute a concrete test case

1 On the Concrete Test Cases pane, open the context menu of the concrete test case and click Execute and one of the following commands:

- Simulate: To start a simulation using the configuration of the concrete test case.
- Validate: To execute the validation functions of the concrete test case.

- **Simulate & Validate**: To start a simulation using the configuration of the concrete test case and execute the validation functions.

ModelDesk executes the concrete test case. A dialog opens where you can monitor the progress and abort the test execution.

Result

The test, logical test case, or concrete test case is executed and the test results are written to a report.

Related topics

Basics

Workflow of Testing 16

HowTos

How to Check the Consistency of the Test 33
How to Specify Start Trigger and/or Stop Trigger (ModelDesk Plotting 🕮)

References

Execute 44
Logical Test Case Properties 60
Maneuver Scheduler (ASM Environment Reference 🕮)

Reference Information

Where to go from here

Information in this section

Testing Commands.....38
Gives you information on the commands of ModelDesk Testing.

Testing Panes and Dialogs.....53
Gives you information on the panes and dialogs of ModelDesk Testing.

Testing Properties.....59
Gives you information on the properties of ModelDesk testing elements.

Signal Class.....63
The Signal class provides methods that you can use to evaluate signals in a custom evaluation.

Alias Support Commands and Panes.....70
Gives you information on all the commands and panes for the alias support.

Testing Commands

Introduction

The following topics gives you information on the commands of ModelDesk Testing.

Where to go from here

Information in this section

Activate....39
To activate the open test.

Activate & Open Plotting....40
To open and activate the plotting configuration.

Activate & Open Parameter Set....40
To open and activate the parameter set.

Append....41
To append a new element to the end of a list.

Check Consistency....41
To check whether the test is consistent.

Configure Variables....42
To select variables for variable variation.

Delete....43
To delete the selected logical test case or concrete test case.

Duplicate....43
To duplicate a concrete test case.

Execute....44
To execute the active test, the selected logical test case, or the selected concrete test case.

Insert....45
To insert a new element to a list.

Load Data - Load <Document>....45
To get the content of a scenario, road, parameter set, plotting, or all documents of a logical test case.

New....46
To create a new test.

Open....46
To open the active test document.

Open from Pool....47
To open a test from the Pool.

Open Road....48
To open the road.

Open Scenario....48
To open the scenario.

Remove Markings....49
To remove the marking of the consistency check.

Save....49
To save the test document.

Save All....50
To save all open test documents.

Save As....50
To save the test document using a new name.

Settings....51
To specify the general settings for testing.

Show Context (Concrete Test Case)....52
To show the context of an alias variable.

Activate

Access

You can access this command via:

Ribbon	Testing – Test
Context menu of	None
Shortcut key	None
Icon	

Purpose

To activate the open test.

Result

The test is active and can be executed.

Description

A ModelDesk project can have several tests. To execute a test you must activate the desired one.

Related topics

Basics

Basics of Logical and Concrete Test Cases..... 12

References

Execute..... 44

Activate & Open Plotting

Access

You can access this command via:

Ribbon	None
Context menu of	Cell of a plotting on **Logical Test Cases** pane
Shortcut key	None
Icon	None

Purpose

To open and activate the plotting configuration.

Result

The plotting configuration is opened and activated.

Related topics

References

Logical Test Cases Pane..... 54

Activate & Open Parameter Set

Access

You can access this command via:

Ribbon	None
Context menu of	Cell of a parameter set on **Logical Test Cases** pane
Shortcut key	None
Icon	None

Purpose

To open and activate the parameter set.

Result

The parameter set is opened and activated.

Related topics

References

Logical Test Cases Pane..... 54

Append

Access

You can access this command via:

Ribbon	None
Context menu of	Concrete Test Cases pane
Shortcut key	None
Icon	None

Purpose

To append a new element to the end of a list.

Result

A new element is added at the end of the list.

Related topics

References

Duplicate..... 43
Insert..... 45

Check Consistency

Access

You can access this command via:

Ribbon	Testing – Test
Context menu of	None

Shortcut key	None
Icon	

Purpose

To check whether the test is consistent.

Result

The test is checked and the inconsistencies are marked.

Related topics

HowTos

How to Check the Consistency of the Test..... 33

References

Remove Markings..... 49

Configure Variables

Access

You can access this command via:

Ribbon	None
Context menu of	Logical Test Cases pane
Shortcut key	None
Icon	None

Purpose

To select variables for variable variation.

Result

The command opens the Variables Configuration dialog for you.

Related topics

References

Variables Configuration Dialog..... 56

Delete

Access

You can access this command via:

Ribbon	None
Context menu of	▪ Logical Test Cases pane ▪ Concrete Test Cases pane ▪ Validation pane
Shortcut key	None
Icon	None

Purpose

To delete the selected logical test case, concrete test case, or validation function.

Result

The selected element is deleted.

Duplicate

Access

You can access this command via:

Ribbon	None
Context menu of	Concrete Test Cases pane
Shortcut key	None
Icon	None

Purpose

To duplicate a concrete test case.

Result

The current concrete test case is duplicated.

Related topics

References

Append.....41
Insert.....45

Execute

Access

You can access this command via:

Ribbon	Testing – Test
Context menu of	▪ Logical test case on the Logical Test Cases pane ▪ Concrete test case on the Concrete Test Cases pane[1)]
Shortcut key	None
Icon	

[1)] The command is divided in Simulate and Validate. To execute both, use Simulate & Validate

Purpose

To execute the active test, the selected logical test case, or the selected concrete test case.

Result

The test or test case is executed.

Description

If you click the Execute button in the ribbon, all the enabled logical test cases are executed.

If you select the Execute command from the context menu of a logical test case, only the selected logical test case is executed.

For the concrete test case, you can execute the simulation or validation only using the Simulate or Validate command or both using the Simulate & Validate command. However, this is only done for the selected concrete test case.

Related topics

HowTos

How to Execute Tests 34

References

Activate 39

Insert

Access

You can access this command via:

Ribbon	None
Context menu of	▪ Logical Test Cases pane ▪ Concrete Test Cases pane ▪ Validation pane
Shortcut key	None
Icon	None

Purpose

To insert a new element to a list.

Result

A new element is added at the current position of the list.

Description

The order of the list can not be modified. If you want a specific order, you must execute the Insert command at the desired position.

Related topics

References

Append.....41
Duplicate.....43

Load Data - Load <Document>

Access

You can access this command via:

Ribbon	None
Context menu of	Logical Test Cases pane
Shortcut key	None
Icon	None

Purpose

To get the content of a scenario, road, parameter set, plotting, or all documents of a logical test case.

Result

The documents are loaded.

Description

You can load a single document of the selected scenario, road, parameter set, or plotting, or all selected documents.

New

Access

You can access this command via:

Ribbon	Testing – Test
Context menu of	Testing node in the Project Navigator
Shortcut key	None
Icon	

Purpose

To create a new test.

Result

A new test is created.

Related topics

HowTos

How to Create a Test and Logical Test Cases..... 19

Open

Access

You can access this command via:

Ribbon	Testing – Test
Context menu of	Testing node in the Project Navigator
Shortcut key	None
Icon	

Purpose

To open the active test document.

Result

The test document is opened.

Related topics

References

Save.....49
Save As.....50

Open from Pool

Access

You can access this command via:

Ribbon	Testing – Test
Context menu of	Testing node in the Project Navigator
Shortcut key	None
Icon	

Purpose

To open a test from the Pool.

Result

A test is opened.

Description

ModelDesk opens a dialog for you to select a test.

Related topics

Basics

Basics of Logical and Concrete Test Cases.....12

HowTos

How to Create a Test and Logical Test Cases.....19

Open Road

Access

You can access this command via:

Ribbon	None
Context menu of	Cell of a road on **Logical Test Cases** pane
Shortcut key	None
Icon	None

Purpose

To open the road.

Result

The road is opened but not activated.

Related topics

References

Logical Test Cases Pane..... 54

Open Scenario

Access

You can access this command via:

Ribbon	None
Context menu of	Cell of a scenario on **Logical Test Cases** pane
Shortcut key	None
Icon	None

Purpose

To open the scenario.

Result

The scenario is opened but not activated.

Related topics

References

Logical Test Cases Pane..... 54

Remove Markings

Access

You can access this command via:

Ribbon	Testing – Test
Context menu of	None
Shortcut key	None
Icon	

Purpose

To remove the markings of the consistency check.

Result

The markings are removed.

Related topics

HowTos

How to Check the Consistency of the Test.....33

References

Check Consistency.....41

Save

Access

You can access this command via:

Ribbon	Testing – Test
Context menu of	Testing node in the Project Navigator
Shortcut key	None
Icon	

Purpose

To save the test document.

Result

The test document is saved.

Related topics

References

Open....46
Save All....50
Save As....50

Save All

Access

You can access this command via:

Ribbon	Testing – Test
Context menu of	None
Shortcut key	None
Icon	

Purpose

To save all open test documents.

Result

All open test documents are saved.

Related topics

References

Save....49
Save As....50

Save As

Access

You can access this command via:

Ribbon	Testing – Test
Context menu of	Testing node in the Project Navigator
Shortcut key	None
Icon	

Purpose

To save the test document using a new name.

Result

The test document is saved using a new name.

Description

The command opens a Specify File Name dialog for you to specify a new name. You can also modify the path and save the test document into the Tests folder or one of its subfolder. Other folders are not allowed.

Only the following characters are allowed: **0** … **9**, **a** … **z**, **A** … **Z**, **_**, **-**, **Space**

Related topics

References

Open....46
Save....49
Save All....50

Settings

Access

You can access this command via:

Ribbon	Testing – Test
Context menu of	None
Shortcut key	None
Icon	

Purpose

To specify the general settings for testing.

Result

The command opens the Test Settings dialog for you to specify the general settings for testing.

Related topics

References

Test Settings Dialog....55

Show Context (Concrete Test Case)

Access

You can access this command via:

Ribbon	None
Context menu of	Header with a variable name on Concrete Test Cases pane
Shortcut key	None
Icon	None

Purpose

To show the context of an alias variable.

Result

The context of the alias variable is shown.

Description

The document for which the variable is created is opened. The property related to the alias variable is marked.

Testing Panes and Dialogs

Introduction

The following topics gives you information on the panes and dialogs of ModelDesk Testing.

Where to go from here

Information in this section

Concrete Test Cases Pane 53
To specify concrete test cases for a logical test case.

Logical Test Cases Pane 54
To specify logical test cases.

Test Settings Dialog 55
To create a template for the file names of the capture and report files.

Variables Configuration Dialog 56
To select an alias variable or parameter of the parameter set for a concrete test case.

Validation Pane 56
To specify the validation rules for a logical test case.

Concrete Test Cases Pane

Access

You can open the pane via the New, Open, and Open from Pool commands.

Purpose

To specify concrete test cases for a logical test case.

Description

In the Concrete Test Case pane, you can specify the concrete test case that is based on a logical test case.

Simulate Lets you enable or disable the simulation of the concrete test case.

Validate Lets you enable or disable the validation of the concrete test case.

Verdict Displays the verdict of the last validation.

Report Displays a link to the report file.

Scenario Lets you select alias variables of the scenario to be modified for the test case and specify their values.

Road Lets you select alias variables of the road to be modified for the test case and specify their values.

Parameter Set Lets you select parameters of the parameter set to be modified for the test case and specify their values.

Related topics

References

Logical Test Cases Pane....54
New....46
Open....46
Open from Pool....47
Validation Pane....56

Logical Test Cases Pane

Access

You can open the pane via the New, Open, and Open from Pool commands.

Purpose

To specify logical test cases.

Description

On the Logical Test Cases pane, you can specify the logical test cases.

Name Lets you specify the name of the logical test case. Only the following characters are allowed: **0** … **9**, **a** … **z**, **A** … **Z**, **_**, **-**, **Space**

Enabled Lets you enable or disable the execution of the logical test case.

Verdict Displays the verdict of the last validation of the logical test case.

Report Provides a link to the report of the last validation of the logical test case.

Scenario Lets you select the scenario used in the logical test case.

Road Lets you select the road used in the logical test case.

Parameter Set Lets you select the parameter set used in the logical test case.

Plotting Lets you select the plotting configuration used in the logical test case.

Duration Lets you specify the maximum duration of every concrete test case in the logical test case. The duration is measured in seconds of the capture time.

Comment Lets you specify a comment.

Related topics

References

Concrete Test Cases Pane.....53
New.....46
Open.....46
Open from Pool.....47
Validation Pane.....56

Test Settings Dialog

Access

You can open the dialog via the **Settings** command

Purpose

To create the file names of the capture and report files.

Description

The dialog has two pages to create the templates for the file names of the captures and reports. The file names can consist of several elements that are defined by placeholders.

Tip

To avoid overwriting of previous test, you must add the TimeDate placeholder to the file name.

Captures page

The first row displays the template of the file name. The file name always starts with the index of the concrete test case. You can add the time/date and a free text to the template using drag & drop from the row below the template. If the template includes free text, an edit field for the text is displayed. The elements are separated by underscores.

File name Lets you configure the template of the file name.

Free text Lets you specify a text.

Reports page

The first row displays the template of the file name. The file name can consists of four different elements: Time/date, logical text case name, test name, and free text. You can add the elements text to the template using drag & drop from the

row below the template. If the template includes free text, an edit field for the text is displayed. The elements are separated by underscores.

File name Lets you configure the template of the file name.

Free text Lets you specify a text.

Related topics

References

Settings.....51

Variables Configuration Dialog

Access

The dialog opens when you select an alias variable or a parameter of the parameter set on the Concrete Test Case pane or select the Configure Variables command.

Purpose

To select an alias variable or parameter of the parameter set for a concrete test case.

Description

The dialog displays all the alias variables that you have created for the scenario and road and the parameter of the parameter set of the simulation model. You can select the variables that are varied in the test case.

Related topics

References

Concrete Test Cases Pane.....53
Configure Variables.....42

Validation Pane

Access

You can open the pane via the New, Open, and Open from Pool commands.

Purpose

To specify the validation rules for a logical test case.

Description

On the Validation pane, you can specify the validation function of logical test cases.

No. Displays the index.

Type Lets you select whether the validation function is used for evaluating or reporting.

Function Lets you select the function used for validation. For a description of the function type, refer to Basics of Validation and Reporting on page 13

For evaluation, you can use a predefined function or a Python custom script.

For reporting, you can insert text, image or a twodimensional plot.

Comment Lets you specify a comment.

Summary Displays a summary of the validation function.

Inputs - Signal or Parameter Lets you specify a signal or captured signal that is used for validation.

Type	Description
Signal	A signal must be used in a plotting configuration of the experiment. To select the signal, you must specify the layout and then the signal.
Captured signal	A captured signal must be available in a MAT file. To select the captured signal, you must specify the MAT file and then the signal.

Inputs - Bound Lets you specify the bound that is used for validation. It depends on the selected function how the bound is interpreted. To specify the bound, you can use 4 different types:

Type	Description
Constant	The bound is a constant value.
Variable	The bound is specified by an alias variable. The alias variable must be selected in the logical test case. To select the alias variable, you must specify the list and then the alias variable. The value of the alias variable is the value which has been used in the simulation.
Signal	The bound is specified by a signal that is used in a plotting configuration of the experiment. To select the signal, you must specify the layout and then the signal. The signal values are the result of the simulation.
Captured signal	The bound is specified by a captured signal that is available in a MAT file. To select the captured signal, you must specify the MAT file and then the signal. The signal values are independent of the simulation.

Inputs – Absolute/Relative Tolerance Lets you specify tolerance values that are required by some function types.

Related topics

HowTos

How to Create a Validation Function for a Report.....27
How to Create a Validation Function for Evaluation.....22

References

Logical Test Cases Pane.....54
New.....46
Open.....46
Open from Pool.....47

Testing Properties

Introduction

The following topics give you information on the properties of ModelDesk testing elements.

Where to go from here

Information in this section

Concrete Test Case Properties.....59
To specify a concrete test case.

Logical Test Case Properties.....60
To specify a logical test case.

Test Properties.....61
To specify the global properties of the test document.

Validation Function Properties.....62
To specify a validation function.

Concrete Test Case Properties

Properties

To specify a concrete test case.

Concrete Test Case properties

Comment Lets you specify a comment for the concrete test case.

ID Displays the identifier of the concrete test case.

Simulate Lets you enable or disable the concrete test case for simulation.

Validate Lets you enable or disable the concrete test case for validation.

Simulation Result properties

Capture Provides a link to the capture of the last simulation of the concrete test case.

Validation Result properties

Verdict Displays the verdict of the last validation of the concrete test case.

Report Provides a link to the report of the last validation of the concrete test case.

Related topics

Basics

Basics of Logical and Concrete Test Cases..... 12

HowTos

How to Create Concrete Test Cases..... 20

Logical Test Case Properties

Properties

To specify a logical test case.

Logical Test Case properties

Name Lets you specify the name of the logical test case.

ID Displays the index of the logical test case.

Enabled Lets you enabled or disable the logical test case.

Duration Lets you specify the maximum duration of every concrete test case in the logical test case. The duration is measured in seconds of the capture time.

Comment Lets you specify a comment.

Experiment Contents

Content Lets you select the experiment content (scenario, road, parameter set, or plotting configuration) used in the logical test case.

Parameter Set Displays the information on the active parameter set (Content property and Use content property).

Plotting Displays the information on the active plotting configuration (Content property and Use content property). The plotting configuration cannot be disabled.

Road Displays the information on the active road (Content property and Use content property).

Scenario Displays the information on the active scenario (Content property and Use content property).

Use content Lets you enable or disable the experiment content (scenario, road, parameter set, or plotting configuration) in the logical test case. If the property is disabled, the content is not downloaded to the simulation.

Validation Result

Report Provides a link to the report of the last validation of the logical test case.

Verdict Displays the verdict of the last validation of the logical test case.

Related topics

Basics

Basics of Logical and Concrete Test Cases..... 12

HowTos

How to Create a Test and Logical Test Cases..... 19

Test Properties

Purpose

To specify the global properties of the test document.

Test properties

Comment Lets you specify a comment for the test document.

Last executed on Displays the date and time of the last execution of the test document.

Last modified on Displays the date and time of the last modification of the test document.

Name Displays the name of the test document.

Experiment Contents properties

Use Parameter Set Lets you specify to use a parameter set in the logical and concrete test cases. If it is disabled, no parameter set is downloaded.

Use Scenario Lets you specify to use a scenario in the logical and concrete test cases. If it is disabled, no scenario is downloaded.

Use Road Lets you specify to use a road in the logical and concrete test cases. If it is disabled, no road is downloaded.

Validation Result properties

Verdict Displays the verdict of the last test validation.

Related topics

HowTos

How to Create a Test and Logical Test Cases..... 19

Validation Function Properties

Purpose

To specify a validation function.

Validation Function properties

Type Lets you select whether the validation function is used for evaluation or reporting.

Function Type Lets you select the function type. The function type that can be used depends on the type of validation function. For a description of the function type, refer to Basics of Validation and Reporting on page 13.

Comment Lets you specify a comment for the validation function.

Custom Function Path Lets you specify the file path to the custom evaluation function.

Display Name Displays information on the parameter used for the validation function.

Property	Description
Display Name	Displays the parameter name.
Type	Lets you select the parameter type.
Container	Lets you select the container from where you select the signal. The container is an alias variable list or layout of a plotting configuration, for example.
Value	Lets you specify a value.

Related topics

Basics

Basics of Validation and Reporting..... 13

HowTos

How to Create a Validation Function for a Report..... 27
How to Create a Validation Function for Evaluation..... 22

Signal Class

Purpose

The `Signal` class provides methods that you can use to evaluate signals in a custom evaluation.

Where to go from here

Information in this section

Signal Class Description.....63
To provide a standardized signal used with the evaluation methods.

all (Signal).....65
To test whether all function values (y-axis) of a signal are truthful.

any (Signal).....65
To test whether at least one function value (y-axis) of a signal is truthful.

Binary Operators.....66
To create an evaluation signal by applying a binary operation to the two input evaluation signals in an element-wise manner.

Slicing (Signal).....68
To get the x-axis or y-axis of a signal as a list of values.

Information in other sections

How to Use a Python Script for a Custom Validation.....22
You can use your own Python script for validating the test results.

Signal Class Description

Library

`TestingLibrary.Validation.signal`

Syntax

```
from TestingLibrary.Validation.signal import Signal
MySignal = Signal(xList, yList)
```

Purpose

To provide a standardized signal used with the evaluation methods.

Description

The `Signal` class is a special class to be used in the evaluation. Because the evaluation methods must handle different test results, such as a capture result or values stored in a MAT file, these result values have to be prepared beforehand

to be used in a custom evaluation. The resulting object is a `Signal` object that contains discrete values for the x- and y-axis.

The `Signal` object provides a list of lists, where the first list (index 0) contains the values of the x-axis, the second list (index 1) contains the values of the y-axis. The number of list items must be the same in both lists and the values of the x-axis must be strictly increasing ($x_i < x_{i+1}$).

Note

The comparison between two signals (==, <, <=, >, >=) does not result in a Boolean value but in a new signal that contains Boolean values. Signal objects therefore cannot be used for conditions. You can use the `any` and `all` methods of the `Signal` class in conditions.

Accessing Signal objects

The Python module `TestingLibrary` provides the possibility to work with the Signal objects within a custom evaluation script as follows:

- You can assign a tuple of two lists to a Signal object.
- You can instantiate a Signal objects value with `Signal( xList, yList )`
- You can use the mathematic and comparison operators on the Signal object.

  The result you get is equivalent to the result of the binary operators that has the following presettings:
  - x-Axis = From left operand
  - Interpolation = Linear

  For more information, refer to Binary Operators on page 66.
- You get the number of sample points with: `len(Signal)`

Parameters

An object is constructed using the following parameters:

Parameter	Type	Description
xList	list[1)]	Values of the x-axis. The values of the x-axis must be strictly increasing ($x_i < x_{i+1}$).
yList	list[1)]	Values of the y-axis.

[1)] The number of list items must be the same in both lists

Related topics

Basics

Implementing a Python Script for Validation.....24

References

all (Signal).....65
any (Signal).....65
Binary Operators.....66

all (Signal)

Class

`Signal`

Syntax

```
RetVal = all(self)
```

Purpose

To test whether all function values (y-axis) of a signal are truthful.

Description

A numerical value is truthful if it is nonzero.

Parameters

–

Return value

The method returns the following parameter:

Type	Description
Boolean	True if all function values are truthful, False otherwise.

Related topics

Basics

Implementing a Python Script for Validation 24

References

any (Signal) 65
Signal Class Description 63

any (Signal)

Library

`Signal`

Syntax

```
RetVal = any(self)
```

Purpose

To test whether at least one function value (y-axis) of a signal is true.

Description

A numerical value is true if it is nonzero.

Parameters

–

Return value

The method returns the following parameter:

Type	Description
Boolean	True if at least one function value is true, false otherwise

Related topics

Basics

Implementing a Python Script for Validation 24

References

all (Signal) 65
Signal Class Description 63

Binary Operators

Class

`Signal`

Syntax

```
Signal = Signal1 <operator> Signal2
```

Purpose

To create an evaluation signal by applying a binary operation to the two input evaluation signals in an element-wise manner.

Description

A binary operation is represented by `y = f(a,b)` where:

a = y-value of the left operand signal

b = y-value of the right operand signal

y = y-value of the resulting signal

The function `f` is specified by an operation. The new signal uses the x-axis of the left operand. For calculating the y-values at the new sample points, a linear interpolation method is used.

Operations

Specifies the operation to be used for creating the new evaluation signal.
Possible operations:

Name	Operator	Description
Addition	+	$y_i = y_{Left,i} + y_{Right,i}$
Subtraction	–	$y_i = y_{Left,i} - y_{Right,i}$
Multiplication	*	$y_i = y_{Left,i} \cdot y_{Right,i}$
Division	/	$y_i = y_{Left,i} / y_{Right,i}$ The division always returns a float value. A division by zero results in ∞.
Floor division	//	$y_i = y_{Left,i} // y_{Right,i}$ The floor division returns the result of a division rounded to the next lower integer, e.g.: 5 // 3 = 1
Modulus	%	$y_i = y_{Left,i} \% y_{Right,i}$ The modulus returns the remainder of a division, e.g.: 5 % 3 = 2
Power	**	$y_i = y_{Left,i}^{Right,i}$ A domain error ($-1^{0.5}$) results in NaN (not a number).
Less than	<	$y_i = y_{Left,i} < y_{Right,i}$ $y_i = 0.0$, if the condition is false $y_i = 1.0$, if the condition is true
Less than or equal to	<=	$y_i = y_{Left,i} \leq y_{Right,i}$ $y_i = 0.0$, if the condition is false $y_i = 1.0$, if the condition is true
Equal	==	$y_i = y_{Left,i} == y_{Right,i}$ $y_i = 0.0$, if the condition is false $y_i = 1.0$, if the condition is true
Not equal	!=	$y_i = y_{Left,i} != y_{Right,i}$ $y_i = 0.0$, if the condition is false $y_i = 1.0$, if the condition is true
Greater than or equal to	>=	$y_i = y_{Left,i} \geq y_{Right,i}$ $y_i = 0.0$, if the condition is false $y_i = 1.0$, if the condition is true
Greater than	>	$y_i = y_{Left,i} > y_{Right,i}$ $y_i = 0.0$, if the condition is false $y_i = 1.0$, if the condition is true

Return value

The method returns the following parameter:

Type	Description
Signal Class Description[1)]	Result of the operation.

[1)] Refer to Signal Class Description on page 63.

Examples

The example shows how to add two signals:

```
Signal1 = [0],[1]
Signal2 = [0],[2]
Signal = Signal1 + Signal2
```

The example shows how to add a constant value to all y-values of a signal:

```
Signal1 = [0],[1]
AddValue = 42.0
Signal = Signal1 + AddValue
```

Related topics

Basics

Implementing a Python Script for Validation 24

References

Signal Class Description 63

Slicing (Signal)

Library

`Signal`

Syntax

```
signal = Signal(x, y)
x_axis = signal[0]
y_axis = signal[1]
```

Purpose

To get the x-axis or y-axis of a signal as a list of values.

Related topics

Basics

Implementing a Python Script for Validation..... 24

References

Signal Class Description..... 63

Alias Support Commands and Panes

Introduction

The following topics gives you information on all the commands and panes for the alias support.

Where to go from here

Information in this section

Add Alias.....70
To create an alias variable that references the selected property.

Alias Overview.....71
To manage the alias variables and their property references.

Remove Alias.....72
To remove all property references of the selected property from all alias variables.

Show Context.....73
To show the context of an alias variable.

Add Alias

Access

You can access this command via:

Ribbon	None
Context menu of	In the **Road Generator** and the **Scenario Editor**: ▪ **Properties** pane – property
Shortcut key	None
Icon	None

Purpose

To create an alias variable that references the selected property.

Result

A new alias variable is created and listed in the **Alias Overview**.

Related topics

Basics

Basics of the Alias Support.....29

HowTos

How to Create Alias Variables.....30

References

Alias Overview.....71
Remove Alias.....72

Alias Overview

Access

You can access this command via:

Ribbon	View – Controlbar – Switch Controlbars – Alias Overview
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To manage the alias variables and their property references.

Description

The controlbar lists all the available alias variables and their property references.

You can modify the names of the alias variables and the comments of the alias variables and property references. All other displayed values are read-only.

Each time you add an alias variable using the Add Alias command, a new alias variable is created and displayed in the Alias Overview. If the property reference has to be assigned to an existing alias variable, you can drag & drop the referenced parameter to an existing alias variable on the controlbar. Alias variables without property references are automatically removed. It is also possible to cut a property reference and paste it to another alias variable using the commands on the controlbar, refer to the following table.

Commands

The controlbar provides the following commands in a toolbar:

Icon	Command	Purpose
	Toggle Column Header	To show or hide the column header.
	Expand All	To expand the list of alias variables.
	Collapse All	To collapse the list of alias variables.
	Cut Reference	To cut a property reference.
	Copy Reference	To copy a property reference.
	Paste Reference	To paste a property reference to the selected alias variable.
	Remove	To remove the selected alias variable or property reference.
	Show Context	To highlight the referenced property in the Road Generator or Scenario Editor.

Related topics

Basics

Basics of the Alias Support.....29

HowTos

How to Create Alias Variables.....30

References

Add Alias.....70
Remove Alias.....72

Remove Alias

Access

You can access this command via:

Ribbon	None
Context menu of	In the Road Generator and the Scenario Editor: ▪ Properties pane – property
Shortcut key	None
Icon	None

Purpose

To remove all property references of the selected property from all alias variables.

Description

When all the property references are removed from an alias variable, the alias variable is deleted.

For maneuvers: Note that you also have to click the **Apply** button to execute the command.

Result

All property references of the property are removed.

Related topics

Basics

Basics of the Alias Support.....29

HowTos

How to Create Alias Variables.....30

References

Add Alias.....70
Alias Overview.....71

Show Context

Access

You can access this command via:

Ribbon	None
Context menu of	Header with a variable name on **Alias Overview** pane
Shortcut key	None
Icon	None

Purpose

To show the context of an alias variable.

Result

The context of the alias variable is shown.

Description

The document for which the variable is created is opened. The property related to the alias variable is marked.

Automation

Where to go from here

Information in this section

General Information on Automation of Testing.....76
Gives you an overview and example of the automation of testing

Classes for Testing.....79
To get information on the classes that are necessary for working with ModelDesk testing.

Constants for Testing.....91

General Information on Automation of Alias Support.....92
You get a quick overview of the classes for working with alias variables.

Classes for Alias Support.....94
To get information on all the classes that are necessary for working with alias variables.

General Information on Automation of Testing

Where to go from here

Information in this section

Basics on the Automation of Testing.....76
Provides basic information on the automation of testing.

Example of Automating Testing.....76
The example shows how you can execute tests using the tool automation.

Overview of the Object Model of Testing.....77
You get a quick overview of the classes for working with testing.

Basics on the Automation of Testing

Introduction

Provides basic information on the automation of testing.

Features

In this version, you can use the automation for executing the test cases:

- You can execute all the concrete test cases of all enabled logical test cases.
- You can execute all the concrete test cases of a logical test case.
- You can execute a single concrete test case.

Before you can execute the test cases, you must prepare them in ModelDesk.

Workflow of testing automation

For automated testing, perform the following steps:

1. Load the project and experiment that you want to use for testing.
2. Configure all the logical test cases and concrete test cases in ModelDesk using the graphical user interface. Refer to Preparing the Tests on page 17.
3. Write and start the script for testing.

   For an overview of the classes and methods, refer to Overview of the Object Model of Testing on page 77.

   For an example, refer to Example of Automating Testing on page 76.

Example of Automating Testing

Intoduction

The example shows how you can execute tests using the tool automation.

Example

Preconditions To use the example, ModelDesk must be running and the experiment containing a specified test must be loaded. Refer to Handling Projects and Experiments in Python (ModelDesk Project and Experiment Management 🕮).

```
import dspace.com
Enums = dspace.com.Enums(Application)
# Access the active test
Test = Application.ActiveProject.ActiveExperiment.Test
# Execute the complete test
MyValidationVerdict = Test.Execute()
if MyValidationVerdict == Enums.ValidationVerdict.Passed:
    print('The complete test successfully passed.')
# Access the first logical test case
MyLogicalTestCase = Test.LogicalTestCases.Item(0)
# Execute the concrete test cases of the first logical test case
MyValidationVerdict = MyLogicalTestCase.Execute()
if MyValidationVerdict == Enums.ValidationVerdict.Passed:
    print('The first logical test case successfully passed.')
# Access the first concrete test case
MyConcreteTestCase = MyLogicalTestCase.Item(0)
# Execute the concrete test case
MyValidationVerdict = MyConcreteTestCase.Execute()
if MyValidationVerdict == Enums.ValidationVerdict.Passed:
    print('The concrete test case successfully passed.')
elif MyValidationVerdict == Enums.ValidationVerdict.Undefined:
    print('The verdict is undefined.')
elif MyValidationVerdict == Enums.ValidationVerdict.Failed:
    print('The test failed.')
elif MyValidationVerdict == Enums.ValidationVerdict.Exception:
    print('An exception has occured.')
```

Overview of the Object Model of Testing

Introduction

You get a quick overview of the classes for working with testing.

Symbols

The following symbols are used in the object model overview:

Symbol	Description
	Method, function
	Attribute (property, class)
	Collection
0, 1, 2, …	Level of dependency (0, 1, 2, …)
⊗	Read only

Overview

The following table gives an overview of the classes, methods, and attributes of the object model.

Class	Level
Application.ActiveProject.ActiveExperiment.Test	0
String Name ⊗	1
ValidationVerdict Verdict ⊗	
ValidationVerdict Execute()	
LogicalTestCases	
Integer Count ⊗	2
LogicalTestCase Add()	
LogicalTestCase Item(object Index)	
Boolean Enabled	3
Integer ID ⊗	
String Name	
ValidationVerdict Verdict ⊗	
ValidationVerdict Execute()	
ConcreteTestCases	
Integer Count ⊗	4
ConcreteTestCase Add()	
ConcreteTestCase Item(object Index)	
Integer ID ⊗	5
ValidationVerdict Verdict ⊗	
ValidationVerdict Execute()	

Related topics

References

ActiveExperiment (ModelDesk Project and Experiment Management 🕮)
Classes for Testing.....79
Constants for Testing.....91
Project (ModelDesk Project and Experiment Management 🕮)

Classes for Testing

Purpose

To get information on all the classes that are necessary for working with ModelDesk testing.

Where to go from here

Information in this section

ConcreteTestCases 79
To access the ConcreteTestCases collection of a logical test case.

ConcreteTestCase 82
To access a concrete test case.

LogicalTestCases 84
To access the LogicalTestCases collection of a test.

LogicalTestCase 86
To access a logical test case.

Test 88
To access the test.

ConcreteTestCases

Purpose

To access the ConcreteTestCases collection of a logical test case.

Where to go from here

Information in this section

Class Description (ConcreteTestCases) 80
To describe the class and its attributes.

Add 80
To add a concrete test case.

Item 81
To get a specific concrete test case.

Class Description (ConcreteTestCases)

Syntax

```
ConcreteTestCases = LogicalTestCase.ConcreteTestCases
```

Purpose

To access the collection of concrete test cases.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
Count	Integer	To get the number of concrete test cases.

Methods

The class contains the following methods:

Method	Purpose
Add	To add a concrete test case. Refer to Add on page 80.
Item	To get a specific concrete test case. Refer to Item on page 81.

Related topics

References

LogicalTestCase 86

Add

Class

ConcreteTestCases

Syntax

```
ConcreteTestCase = ConcreteTestCases.Add()
```

Purpose

To add a concrete test case.

Parameters

–

Return value

The method returns the following parameter:

Type	Description
ConcreteTestCase[1)]	The new concrete test case.

[1)] Refer to ConcreteTestCase on page 82.

Related topics

References

Class Description (ConcreteTestCases)..... 80

Item

Class

ConcreteTestCases

Syntax

```
ConcreteTestCase = ConcreteTestCases.Item(object Index)
```

Purpose

To get a specific concrete test case.

Parameters

The method uses the following parameters:

Parameter	Type	Description
Index	object	The index of the specific concrete test case. **Note** This is the index used in the ConcreteTestCases collection. This index and the ID attribute of the ConcretTestCase are different.

Return value

The method returns the following parameter:

Type	Description
ConcreteTestCase[1)]	The specific concrete test case.

[1)] Refer to ConcreteTestCase on page 82.

Related topics

References

Class Description (ConcreteTestCases).....80

ConcreteTestCase

Purpose

To access a concrete test case.

Where to go from here

Information in this section

Class Description (ConcreteTestCase).....82
To describe the class and its attributes.

Execute.....83
To execute the concrete test case.

Class Description (ConcreteTestCase)

Syntax

```
ConcreteTestCase = ConcreteTestCases.Item()
ConcreteTestCase = ConcreteTestCases.Add()
```

Purpose

To access a concrete test case.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
ID	Integer	To get the index of the concrete test case. **Note** Note that this index and the index of the ConcreteTestCase in the ConcreteTestCase collection are different.

Attributes	Type	Purpose
Verdict	ValidationVerdict[1)]	To get the verdict of the concrete test case.

[1)] Refer to ValidationVerdict on page 91.

Methods

The class contains the following methods:

Method	Purpose
Execute	To execute the concrete test case. Refer to Execute on page 83.

Related topics

References

ConcreteTestCases.....79

Execute

Class

ConcreteTestCase

Syntax

```
ValidationVerdict = ConcreteTestCase.Execute()
```

Purpose

To execute the concrete test case.

Parameters

–

Return value

The method returns the following parameter:

Type	Description
ValidationVerdict[1)]	The verdict of the concrete test case.

[1)] Refer to ValidationVerdict on page 91.

Related topics

HowTos

How to Execute Tests.....34

References

Class Description (ConcreteTestCase).....82

LogicalTestCases

Purpose

To access the LogicalTestCases collection of a test.

Where to go from here

Information in this section

Class Description (LogicalTestCases).....84
To describe the class and its attributes.

Add.....85
To add a logical test case.

Item.....86
To get a specific test case.

Class Description (LogicalTestCases)

Syntax

```
LogicalTestCases = Test.LogicalTestCases
```

Purpose

To access the collection of logical test cases.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
Count	Integer	To get the number of logical test cases.

Methods

The class contains the following methods:

Method	Purpose
Add	To add a logical test case. Refer to Add on page 85.
Item	To get a specific test case. Refer to Item on page 86.

Related topics

References

Test..... 88

Add

Class

LogicalTestCases

Syntax

```
LogicalTestCase = LogicalTestCases.Add()
```

Purpose

To add a logical test case.

Parameters

–

Return value

The method returns the following parameter:

Type	Description
LogicalTestCase[1)]	The new logical test case.

[1)] Refer to LogicalTestCase on page 86.

Related topics

References

Class Description (LogicalTestCases)..... 84

Item

Class

LogicalTestCases

Syntax

```
LogicalTestCase = LogicalTestCases.Item(object Index)
```

Purpose

To get a specific logical test case.

Parameters

The method uses the following parameters:

Parameter	Type	Description
Index	object	The index of the logical test case.

Return value

The method returns the following parameter:

Type	Description
LogicalTestCase[1)]	The specific logical test case.

[1)] Refer to LogicalTestCase on page 86.

Related topics

References

Class Description (LogicalTestCases).....84

LogicalTestCase

Purpose

To access a logical test case.

Where to go from here

Information in this section

Class Description (LogicalTestCase).....87
To describe the class and its attributes.

Execute.....87
To execute all the concrete test cases of a logical test case.

Class Description (LogicalTestCase)

Syntax

```
LogicalTestCase = LogicalTestCases.Item()
LogicalTestCase = LogicalTestCases.Add()
```

Purpose

To access a logical test case.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
Enabled	Boolean	To get/set the flag that enables the logical test case when the whole test is executed.
ID	Integer	To get the index of the logical test case.
Name	String	To get/set the name of the logical test case.
Verdict	ValidationVerdict[1)]	To get the verdict.
ConcreteTestCases	ConcreteTestCases[2)]	To get the collection of the concrete test cases.

1) Refer to ValidationVerdict on page 91.
2) Refer to ConcreteTestCases on page 79.

Methods

The class contains the following methods:

Method	Purpose
Execute	To execute all the concrete test cases of the logical test case. Refer to Execute on page 87.

Related topics

References

LogicalTestCases.....84

Execute

Class

LogicalTestCase

Syntax

```
ValidationVerdict = LogicalTestCase.Execute()
```

Purpose

To execute all the concrete test cases of a logical test case.

Parameters

–

Return value

The method returns the following parameter:

Type	Description
ValidationVerdict[1)]	The verdict of the concrete test cases.

[1)] Refer to ValidationVerdict on page 91.

Related topics

HowTos

How to Execute Tests 34

References

Class Description (LogicalTestCase) 87

Test

Purpose

To access the test.

Where to go from here

Information in this section

Class Description (Test) 88
To describe the class and its attributes.

Execute 89
To execute the test.

Class Description (Test)

Syntax

```
Test = ActiveExperiment.Test
```

Purpose

To access the test.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
LogicalTestCases	LogicalTestCases[1)]	To get the collection of logical test cases.
Name	String	To get the name of the test.
Verdict	ValidationVerdict[2)]	To get the verdict of the test.

1) Refer to LogicalTestCases on page 84.
2) Refer to ValidationVerdict on page 91.

Methods

The class contains the following methods:

Method	Purpose
Execute	To execute the test. Refer to Execute on page 89.

Related topics

References

ActiveExperiment (ModelDesk Project and Experiment Management 🕮)

Execute

Class

Test

Syntax

```
ValidationVerdict = Test.Execute()
```

Purpose

To execute the test.

Parameters

–

Return value

The method returns an object of the following type:

Type	Description
ValidationVerdict[1)]	The verdict of the test.

[1)] Refer to ValidationVerdict on page 91.

Related topics

HowTos

How to Execute Tests.....34

References

Class Description (Test).....88

Constants for Testing

Constants for Testing

Introduction

You can use predefined constants in the tool automation.

Constants

ValidationVerdict The following constants are used to specify the verdict of the validation:

Value	Description
Passed = 1	The test is successful passed.
Undefined = 2	The test verdict is undefined.
Failed = 3	The test failed.
Exception = 4	An exception has occurred.

Related topics

References

Test 88

General Information on Automation of Alias Support

Overview of the Object Model for the Alias Support

Introduction

You get a quick overview of the classes for working with alias variables.

Symbols

The following symbols are used in the object model overview:

Symbol	Description
	Method, function
	Attribute (property, class)
	Collection
0, 1, 2, …	Level of dependency (0, 1, 2, …)
⊗	Read only

Overview

The following table gives an overview of the classes of the object model:

Class	Level
Aliases	0
Lists	1
Integer Count ⊗	2
List Item(Integer Index)	
List ItemByContext(String Context)	
String Context ⊗	3
String Name ⊗	
Variables	
Integer Count ⊗	4
Variable Item(Integer Index)	
String Comment	5
Double Max	
Double Min	
String Name	
SetValue(Double Value)	
References	
Integer Count ⊗	6
Reference Item(Integer Index)	

Class	Level
String Comment String Description ⊗ String Name ⊗ Integer Type ⊗ Double GetValue()	7

Related topics

Basics

Working with Alias Variables.....29

Classes for Alias Support

Purpose

To get information on all the classes that are necessary for working with alias variables.

Where to go from here

Information in this section

Aliases.....94
To access the alias variables.

List.....95
To access a list of alias variables in a context.

Lists.....96
To get the lists containing all the alias variables.

Reference.....99
To access a property reference.

References.....101
To access all the property references assigned to an alias variable.

Variable.....102
To access an alias variable.

Variables.....104
To access the alias variables of a list.

Information in other sections

Basics of the Alias Support.....29
Using alias support, you can modify the values of properties in automation scripts.

Overview of the Object Model for the Alias Support.....92
You get a quick overview of the classes for working with alias variables.

Aliases

Purpose

To access the alias variables.

Class Description (Aliases)

Syntax

```
Aliases = Experiment.Aliases
```

Purpose

To access all the alias variables specified for an experiment.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
Lists	Lists[1)]	To get the lists that contains lists of alias variables specified in one context.

[1)] Refer to Lists on page 96.

Methods

–

Related topics

References

Experiment (ModelDesk Project and Experiment Management 🕮)

List

Purpose

To access a list of alias variables in a context.

Class Description (List)

Syntax

```
List = Lists.Item(Integer Index)
List = Lists.ItemByContext(String Context)
```

Purpose

To access a list of alias variables in one context.

Description

There is one list in each context. The following contexts are defined for alias variables:

- `Road`: List of variables of a road that is created using the Road Generator
- `Maneuver`: List of variables of a maneuver that is created using the Scenario Editor
- `Traffic`: List of variables of a traffic (fellows and global user signals) that is created using the Scenario Editor

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
Context	String	To get the context.
Name	String	To get the name of the road or traffic scenario.
Variables	Variables[1)]	To get the alias variables.

[1)] Refer to Variables on page 104.

Methods

–

Related topics

References

Lists.....96

Lists

Purpose

To get the lists containing all the alias variables.

Where to go from here

Information in this section

Class Description (Lists).....97
To describe the class and its attributes.

Item.....97
To get a list containing the alias variables in a context using the index.

ItemByContext.....98
To get a list containing the alias variables in a context using the name of the context.

Class Description (Lists)

Syntax

```
Lists = Aliases.Lists
```

Purpose

To get the lists that contains all the alias variables in a context.

Description

There is one list in each context. The following contexts are defined for alias variables:

- `Road`: List of variables of a road that is created using the Road Generator
- `Maneuver`: List of variables of a maneuver that is created using the Scenario Editor
- `Traffic`: List of variables of a traffic (fellows and global user signals) that is created using the Scenario Editor

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
Count	Integer	To get the number of lists.

Methods

The class contains the following methods:

Method	Purpose
Item	To get a list containing the alias variables in a context using the index. Refer to Item on page 97.
ItemByContext	To get a list containing the alias variables in a context using the name of the context. Refer to ItemByContext on page 98.

Related topics

References

Aliases.....94

Item

Class

Lists

Syntax

```
List = Lists.Item(Index)
```

Purpose

To get a list containing the alias variables in a context using the index.

Parameters

The method uses the following parameters:

Parameter	Type	Description
Index	Integer	Index of the list

Return value

The method returns the following parameter:

Type	Description
List[1)]	The list of alias variables specified in a context.

1) Refer to List on page 95.

Related topics

References

Class Description (Lists).....97

ItemByContext

Class

Lists

Syntax

```
List = Lists.ItemByContext(String Context)
```

Purpose

To get a list containing the alias variables in a context using the name of the context.

Parameters

The method uses the following parameters:

Parameter	Type	Description
Context	String	Context of the alias variables ▪ `"Maneuver"`: Maneuver created using the Scenario Editor ▪ `"Traffic"`: Traffic (fellows and global user signals) created using the Scenario Editor ▪ `"Road"`: Road created using the Road Generator

Return value

The method returns the following parameter:

Type	Description
List[1)]	The list of alias variables in the specified context.

[1)] Refer to List on page 95.

Related topics

References

Class Description (Lists)..... 97

Reference

Purpose

To access a property reference.

Where to go from here

Information in this section

Class Description (Reference)..... 99
To describe the class and its attributes.

GetValue..... 100
To get the value of a property reference.

Class Description (Reference)

Syntax

```
References = Variable.References
```

Purpose

To access a property reference.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
Comment	String	To get/set the comment for the property reference.
Description	String	To get the description of the property reference.
Name	String	To get the name of the property reference.

Attributes	Type	Purpose
Type	Integer	To get the type of the property reference.

Methods

The class contains the following methods:

Method	Purpose
GetValue	To get the value of a property reference. Refer to GetValue on page 100.

Related topics

References

Variable..... 102

GetValue

Class

Reference

Syntax

```
Value = Reference.GetValue()
```

Purpose

To get the value of a property reference.

Parameters

–

Return value

The method returns the following parameter:

Type	Description
Double	The value of the property reference.

Related topics

References

Class Description (Reference)..... 99

References

Purpose

To access all the property references assigned to an alias variable.

Where to go from here

Information in this section

Class Description (References)..... 101
To describe the class and its attributes.

Item..... 102
To get a specific property reference.

Class Description (References)

Syntax

```
References = Variable.References
```

Purpose

To access all the property references assigned to an alias variable.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
Count	Integer	To get the number of property references.

Methods

The class contains the following methods:

Method	Purpose
Item	To get a specific property reference. Refer to Item on page 102.

Related topics

References

Variable..... 102

Item

Class

References

Syntax

```
Reference = References.Item(Index)
```

Purpose

To get a specific property reference.

Parameters

The method uses the following parameters:

Parameter	Type	Description
Index	Integer	The index of the property reference.

Return value

The method returns the following parameter:

Type	Description
Reference[1)]	The specific property reference.

[1)] Refer to Reference on page 99.

Related topics

References

Class Description (References)..... 101

Variable

Purpose

To access an alias variable.

Where to go from here

Information in this section

Class Description (Variable)..... 103
To describe the class and its attributes.

SetValue..... 103
To set the value of the alias variable.

Class Description (Variable)

Syntax

```
Variable = Variables.Variable
```

Purpose

To access an alias variable.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
Comment	String	To get/set a comment for the alias variable.
Min	Double	To get/set a minimum value for the alias variable.
Max	Double	To get/set a maximum value for the alias variable.
Name	String	To get the name of the alias variable.
References	References[1)]	To get all the property references.

[1)] Refer to References on page 101.

Methods

The class contains the following methods:

Method	Purpose
SetValue	To set the value of the alias variable. Refer to SetValue on page 103.

Related topics

References

Variables..... 104

SetValue

Class

Variable

Syntax

```
Variable.SetValue(Double Value)
```

Purpose

To set the value of the alias variable.

Parameters

The method uses the following parameters:

Parameter	Type	Description
Value	Double	The new value of the alias variable and with that the new value of its property references.

Return value

–

Related topics

References

Class Description (Variable)..... 103

Variables

Purpose

To access the alias variables of a list.

Where to go from here

Information in this section

Class Description (Variables)..... 104
To describe the class and its attributes.

Item..... 105
To get a specific alias variable.

Class Description (Variables)

Syntax

```
Variables = List.Variables
```

Purpose

To access the alias variables of a list.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
Count	Integer	To get the number of alias variables.

Methods

The class contains the following methods:

Method	Purpose
Item	To get a specific alias variable. Refer to Item on page 105.

Related topics

References

List 95

Item

Class

Variables

Syntax

```
Variable = Variables.Item(Index)
```

Purpose

To get a specific alias variable.

Parameters

The method uses the following parameters:

Parameter	Type	Description
Index	Integer	The index of the alias variable.

Return value

The method returns the following parameter:

Type	Description
Variable[1)]	The specific alias variable.

1) Refer to Variable on page 102.

Related topics

References

Class Description (Variables)..... 104

A

alias variable 29
  creating 30
  workflow 29
alias variables
  context 96, 97
Aliases class 94

C

capture file name
  configuring 17
Common Program Data folder 8
concrete test case 13
  creating 20
  executing 34
  properties 59
configuring
  capture file names 17
  report file names 17
consistency check 33
constants
  automation 91
context for alias variables 96, 97
creating
  alias variable 30
  concrete test case 20
  logical test case 19
  test 19
  validation function for report 27

D

Documents folder 8

E

executing
  concrete test case 34
  logical test case 34
  test 34

L

List class 95
Lists class 96
Local Program Data folder 8
logical test case 12
  creating 19
  executing 34
  properties 60

O

object model overview
  alias support 92
  testing 77

P

properties
  concrete test case 59
  logical test case 60
  test 61
  validation function 62
property reference 29
  creating 30

R

Reference class 99
References class 101
report
  adding information 27
report file name
  configuring 17

T

test
  creating 19
  executing 34
  properties 61
testing
  workflow 16

V

validation function
  properties 62
Variable class 102
Variables class 104

W

workflow
  alias variable 29
  testing 16