ModelDesk

Automation

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

Contents

This document introduces you to the tool automation of ModelDesk and the built-in Interpreter. It provides basic information of ModelDesk's automation interface.

Required knowledge

You must have experience with the Python programming language or programming in MATLAB.

Tip

To learn more about Python, refer to http://www.python.org/ for a tutorial and other documents on Python.

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.

Symbol	Description
📖	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

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 for Automating ModelDesk

Introduction

The ModelDesk automation interface can be used in Python scripts and M files.

Where to go from here

Information in this section

ModelDesk Automation Interface

Introduction

Gives an overview of the ModelDesk automation interface.

ModelDesk automation interface

The ModelDesk automation interface allows you to control ModelDesk using scripts. It is a COM interface which you can use in several programming languages. In this document, the Python programming language is primarily used. The ModelDesk automation interface consists of classes with attributes and methods. The scripts can be run in a Python interpreter, for example, PythonWin or the built-in Interpreter, or can be executed in AutomationDesk. You can also use the automation interface in M files for automation under MATLAB.

User requirements

You must have experience with the Python programming language. To learn more about Python, refer to http://www.python.org/ for a tutorial and other documents on Python.

Overview

For an overview of the object model, refer to

- Overview of the Object Model for Accessing ModelDesk Experiments (ModelDesk Project and Experiment Management 🕮)
- Overview of the Object Model for Parameterizing (ModelDesk Parameterizing 🕮)
- Classes for Managing Custom Libraries (ModelDesk Parameterizing 🕮)
- Overview of the Object Model for Processing (ModelDesk Processing 🕮)

- Overview of the Object Model for Plotting (ModelDesk Plotting 🕮)
- Overview of the Classes for Working with Roads (ModelDesk Road Creation 🕮)
- Overview of the Classes for Creating Scenarios (ModelDesk Scenario Creation 🕮)
- Overview of the Object Model of Testing (ModelDesk Testing 🕮)
- Overview of the Object Model for the Traffic Object Manager (ModelDesk Traffic Object Management 🕮)

Related topics

Basics

Features of ModelDesk Automation Interface

Introduction

This topic describes the features of the ModelDesk automation interface.

Automation features

The ModelDesk automation interface provides the following features:

- Projects
  - Opening and saving projects
- Experiments
  - Managing experiments
  - Opening and closing experiments
  - Activating and saving experiments
  - Activating and downloading parameter sets, roads, and maneuvers
  - Managing simulation models
- Parameter sets
  - Reading and setting parameters of several types (scalar, vector, matrix, look-up table 1-D, look-up table 2-D)
  - Activating, downloading, and saving parameter sets
- Processing
  - Creating and specifying measurement types
  - Creating and specifying measurement data
  - Specifying conversion rules.
  - Mapping raw data to variables
  - Assigning function and setting files to parameters.
  - Specifying additional functions
  - Configuring the plotting in Matlab

  - Executing the processing for the parameter set, parameter pages, parameters, or additional functions
- Plotting
  - Creating configurations and layouts
  - Managing layout connections
  - Specifying start trigger and stop trigger
  - Starting and stopping of the plotting
  - Storing of simulation results
- Roads
  - Reading and setting friction and tire parameter set values of roads, junctions and included surfaces
  - Modifying road and junction connections
  - Saving and downloading roads
- Scenarios
  - Activating a scenario
  - Adding a maneuver and specifying its properties, longitudinal profile, and lateral profile
  - Adding fellows and specifying their properties and behaviors
  - Adding global user signals and specifying their properties
  - Specifying traffic driver objects
  - Saving and downloading the active scenario
- Testing
  - Executing all the concrete test cases of all enabled logical test cases.
  - Executing all the concrete test cases of a logical test case.
  - Executing a single concrete test case.
- Maneuvers (in the maneuver compatibility mode if they are created using the **Maneuver Editor**)
  - Reading and setting the driver settings
  - Adding and removing maneuver segments
  - Specifying longitudinal profile (final velocity, pedal stimulus, braking)
  - Specifying lateral profile (several kinds of steering or following the road)
  - Saving and downloading maneuvers

Automation in Python or MATLAB

Introduction

You can use the ModelDesk automation interface in different programming languages. The Python programming language is primarily used in this document, but you can also use M files for MATLAB.

Python scripts

Python scripts can be executed, for example, in PythonWin or the Interpreter in ControlDesk. The ModelDesk automation interface provides classes to handle ModelDesk experiments. You can use the classes in your Python scripts for automation. For information on how to write these Python scripts, refer to Automation Using Python Scripts on page 12.

Tip

You can use AutomationDesk to execute the Python scripts. Refer to Using Python in AutomationDesk (AutomationDesk Basic Practices 🕮).

M files

The object model of the ModelDesk automation interface can also be used in MATLAB. MATLAB can create a COM automation server to access ModelDesk via its automation interface. The ModelDesk automation interface provides classes to handle the ModelDesk experiments. These classes, with attributes and methods, can also be used in M files for ModelDesk automation. For information on using the interface in M files and Python, refer to Automating Using M Files on page 16. However, you should also read the instructions on programming in Python, refer to Automation Using Python Scripts on page 12.

Related topics

Basics

Workflow for Automating in Python.. 12

Automation Using Python Scripts

Introduction

The following topics give information on how you can automate ModelDesk by using Python scripts.

Where to go from here

Information in this section

Information in other sections

Workflow for Automating in Python

Introduction

This topic describes the workflow for automating ModelDesk using Python scripts and shows the structure of a Python script.

Workflow for automation

You must perform some tasks for automation in ModelDesk

1. Create roads, maneuvers, and scenarios

   The roads, maneuvers to be driven, and scenarios must be available in the project's Pool.
2. Implement the Python script

   The Python script containing all the automation tasks must be implemented. For information on the structure of the scripts, see below.
3. Start the Python scripts

   You can execute the script in the **Interpreter**, refer to Using the Interpreter on page 21. You can also use AutomationDesk to execute the scripts, refer to Using Python in AutomationDesk (AutomationDesk Basic Practices 🕮).

Structure of Python scripts

To change parameter values, select a road, or specify driving maneuvers, the Python script must perform several steps:

1. Start ModelDesk, open the project, and activate the experiment. Refer to Handling Projects and Experiments in Python (ModelDesk Project and Experiment Management 🕮).
2. Change the parameter values or modify the road, maneuver, or scenario. Refer to
   - Modifying the Values of Model Parameters in Python (ModelDesk Parameterizing 🕮).
   - Automatic Processing (ModelDesk Processing 🕮).
   - Modifying a Road in Python (ModelDesk Road Creation 🕮).
   - Automating Scenarios in Python (ModelDesk Scenario Creation 🕮).
   - Automated Plotting of Simulation Signals in MATLAB (ModelDesk Plotting 🕮).
3. Download the modified parameter values.
4. When automation is finished, delete the used objects to free memory.

Related topics

Basics

Setting Values of Properties Using Alias Variables

Introduction

When alias variables are created in an experiment, you can set their values in a script.

Preconditions

- Alias variables must exist. Refer to How to Create Alias Variables (ModelDesk Testing 🕮).
- The road, maneuver, or scenario to which the properties belong must be active.

Using alias variables

The following scripts show how to work with alias variables.

Getting the Alias object When ModelDesk runs and a project and experiment is loaded, you can access the `Aliases` object using the following lines in the Interpreter:

```
Project = Application.ActiveProject
Experiment = Project.ActiveExperiment
Aliases = Experiment.Aliases
```

You can also automate starting ModelDesk and loading a project and experiment. Refer to Handling Projects and Experiments in Python (ModelDesk Project and Experiment Management 📖).

Accessing the list of alias variables When you have an `Aliases` object, you can access the lists of alias variables in the road, maneuver (created by Maneuver Editor), maneuver (part of a scenario), and traffic (part of a scenario):

```
AliasLists = Aliases.Lists
print "Number of lists: ", AliasLists.Count
# Access an item via index
AliasList = AliasLists.Item(0)
# Access item via context if road is active
# AliasList = AliasLists.ItemByContext("Road")
# Access item via context if maneuver of a scenario is active
# AliasList = AliasLists.ItemByContext("Maneuver")
# Access item via context if traffic of a scenario is active
# AliasList = AliasLists.ItemByContext("Traffic")
# Access item via context if maneuver created by the Maneuver Editor
# in maneuver compatibility mode is active
# AliasList = AliasLists.ItemByContext("ManeuverDeprecated")
# Print name and context of the list
print "Name of list: ", AliasList.Name
print "Context: ", AliasList.Context
```

To access an `AliasList`, you can use the `Item` method or the `ItemByContext` method if the appropriate road, maneuver, or scenario is active.

Setting the value of an alias variable The following script shows how to get information on all the alias variables in a context and modify their values.

```
# Print information on the alias variable
Variables = AliasList.Variables
print "Number of alias variables: ", Variables.Count
for Variable in Variables:
    print "Name of alias variable: ", Variable.Name
    print "Comment of alias variable: ", Variable.Comment
    Variable.SetValue(42)
```

Getting information on property references The folllowing script shows how to get information on the property references that are assigned to an alias variable.

```
# Print information on property references
Variables = AliasList.Variables
# Get an alias variable by specifying its index
Variable = Variables.Item(0)
# Or get an alias variable by specifying its name
# Variable = Variables.Item("AliasVariable1")
for PropertyReference in Variable.References:
    print "Name of property reference: ", PropertyReference.Name
    print "Comment of property reference: ", PropertyReference.Comment
    print "Value of property reference: ", PropertyReference.GetValue()
```

Examples

You can use the following examples in the Interpreter in ModelDesk if a project and experiment is loaded.

The first example shows you how to set the `MyAlias` alias variable in the active scenario.

```
MyLists = Application.ActiveProject.ActiveExperiment.Aliases.Lists
MyLists.ItemByContext("Traffic").Variables.Item("MyAlias").SetValue(9)
```

The second example shows you how to set the first alias variable (index 0) in the first alias list (index 0).

```
MyLists = Application.ActiveProject.ActiveExperiment.Aliases.Lists
MyLists.Item(0).Variables.Item(0).SetValue(9)
```

Related topics

Basics

Basics of the Alias Support (ModelDesk Testing 🕮)

References

Overview of the Object Model for the Alias Support (ModelDesk Testing 🕮)

Automating Using M Files

Introduction

The following topics give information on how you can automate ModelDesk using M files in MATLAB.

Where to go from here

Information in this section

Information in other sections

Workflow for Automating Using M Files

Introduction

This topic describes the workflow for automating ModelDesk using M files and shows the structure of the M file.

Workflow for automation

You must perform some tasks for automation in ModelDesk

1. You can use an existing project containing all the experiments to be performed or create a new project using tool automation. The roads to be driven must exist and be linked to the pool.
2. The M file containing all the automation tasks must be implemented. For information on the structure of the M files, see below.
3. Start the M file in the MATLAB Command Window.

Structure of M files

To change parameter values, select a road, or specify driving maneuver, the M file must perform several steps:

1. Access ModelDesk.
2. Open the project.
3. Activate the experiment.
4. Change the parameter values.
   Select a road.
   Change the maneuver.
5. Download the parameter values.
6. When the automation is finished, you should delete the objects to free memory.

Using the Object Model in MATLAB

Introduction

You can use the object model in MATLAB, so that you can automate ModelDesk using M files. This topic gives some basic information on how to write such M files.

Accessing ModelDesk

Start ModelDesk using the following command in the M file:

```
MyApplication = actxserver('ModelDesk.Application')
set(MyApplication,'Visible',true)
```

The `actxserver` function starts ModelDesk but does not display it. To make it visible, you must set the `Visible` attribute.

Python versus MATLAB

The ModelDesk object model can be used in Python scripts and M files, but with different syntax. The following table compares the syntax in Python and MATLAB.

Python Syntax	MATLAB Syntax	Description
`Object.Method()`	`Method(Object)`	Calling a method of an object without parameter
`Object.Method(Parameter)`	`Object.Method(Parameter)` or `Method(Object,Parameter)`	Calling a method of an object with parameter
`ReturnValue = Object.Method(Parameter)`	`ReturnValue = Object.Method(Parameter)` or `ReturnValue = Method(Object,Parameter)`	Calling a method of an object with parameter and return value
`Object.Attribute = 1.1`	`Object.Attribute = 1.1` or `set(Object,'AttributeName',1.1)`	Setting the value of an attribute of an object
`Value = Object.Attribute`	`Value = Object.Attribute` or `Value = get(Object,'Attribute')`	Reading the value of an attribute of an object

To write your own M files for automation, see the description of programming in Python and rewrite the syntax.

Multidimensional parameters

Multidimensional parameters are vectors, 1-D or 2-D look-up tables.

Reading parameter values When you read the values of multidimensional parameters, they are returned in the cell array format. The following script shows some examples.

```
% Reading a vector
V = MyVectorParameter.V
MyVectorValue = V.Value
% Reading an 1-D look-up table
X = MyLUT1DParameter.X
V = MyLUT1DParameter.V
MyLUT1DValueX = X.Value
MyLUT1DValueV = V.Value
% Reading a 2-D look-up table
X = MyLUT2DParameter.X
Y = MyLUT2DParameter.Y
V = MyLUT2DParameter.V
MyLUT2DValueX = X.Value
MyLUT2DValueY = Y.Value
MyLUT2DValueV = V.Value
```

Writing parameter values To write the values of multidimensional parameters, you can specify the values in the form of vectors (row or column vectors) or matrices depending on the parameter type. It is also possible to specify the values as cell arrays. The following script shows some examples.

```
% Writing a vector
V = get(MyVectorParameter, 'V')
% Specify the values in row vector
V.Value = [1 2 3]
% Or specify the values in a column vector
V.Value = [1; 2; 3])
% Or specify the values in a cell array
V.Value = {1, 2, 3}
% Writing an 1-D look-up table
X = MyLUT1DParameter.X
V = MyLUT1DParameter.V
X.Value = [1 2 3]
V.Value = [1 2 3]
% Or specify the values in a cell array
V.Value = {1, 2, 3}
```

```
% Writing a 2-D look-up table
X = MyLUT2DParameter.X
Y = MyLUT2DParameter.Y
V = MyLUT2DParameter.V
X.Value = [1 2 3]
Y.Value = [1 2 3]
V.Value = [1 2 3; 4 5 6; 7 8 9]
% Or specify the values in a cell array
V.Value = {1, 2, 3; 4, 5, 6; 7, 8, 9}
```

The example shows only how to read or write parameter values. For a complete example, refer to Example of Using the Object Model in MATLAB on page 19.

Related topics

References

Overview of the Object Model for Accessing ModelDesk Experiments (ModelDesk Project and Experiment Management 📖)

Example of Using the Object Model in MATLAB

Example

The following M file is a short example of automating ModelDesk. It shows how to start ModelDesk, load a project, and change a parameter.

```
% ProjectPath must be the full project path of the ModelDesk project
ProjectPath = 'E:\ExamplePath\ExampleProject_005\ExampleProject_005.CDP'
% The ParameterSetName is a name of a parameter set
% It must already be used in the project
ParameterSetName = 'ExampleParameterSet'
% Starting ModelDesk
MyApplication = actxserver('ModelDesk.Application')
% ModelDesk starts invisible by default. It can be set visible
MyApplication.Visible = true
% Opening the ModelDesk project
MyProject = MyApplication.OpenProject(ProjectPath)
% Accessing the Experiments collection in the project
MyExperiments = MyProject.Experiments
% Accessing the first experiment
MyExperiment = MyExperiments.Item(0)
% Activating the experiment
MyActiveExperiment = MyExperiment.Activate(false)
% Accessing the ParameterSets collection
MyParameterSets = MyActiveExperiment.ParameterSets
% Accessing the parameterset 'ParameterSetName'
MyParameterSet = MyParameterSets.Item(ParameterSetName)
% Activating the parameter set
MyActiveParameterSet = MyParameterSet.Activate(false)
```

```
% Accessing a scalar parameter using the Find method
MyScalarParameter = MyActiveParameterSet.Find(
'VehicleDynamics.Aerodynamics.Const_dens_air')
% Read the parameter value
MyValue = MyScalarParameter.V
% Specify a parameter value
MyScalarParameter.V = 1.58
% Accessing a vector parameter
MyVectorParameter = MyActiveParameterSet.Find(
'VehicleDynamics.SENSOR_MOTION.Const_PosVec_Sensor1')
V = MyVectorParameter.V
% It returns a cell array
MyVectorValue = V.Value
% To change the value, you can use a vector or a cell array.
% For example, V.Value = {1, 2, 3}
% You can use row or column vectors
V.Value = [1 2 3]
unit = V.Unit
size = V.Size
% Accessing a 1D look-up table parameter
MyLUT1DParameter = MyActiveParameterSet.Find(
'Drivetrain.GEARBOX_AT.Map_GearRatio')
X = MyLUT1DParameter.X
V = MyLUT1DParameter.V
% It returns a cell array
MyLUT1DValueX = X.Value
X.Value = [1 2 3]
V.Value = [1 2 3]
% Accessing a 2D look-up table parameter
MyLUT2DParameter = MyActiveParameterSet.Find(
'EngineBasic.ESP_TORQUE_INTERVENTION_SLOW.Map_Trq_Engine_Inv')
X = MyLUT2DParameter.X
Y = MyLUT2DParameter.Y
V = MyLUT2DParameter.V
X.Value = [1 2 3 4]
Y.Value = [5 6 7 8]
V.Value = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]
% It returns a cell array
MyLUT2DValueV = V.Value
% You must delete the Application object. Otherwise you can
% close ModelDesk only using Window's Task Manager
MyApplication.delete()
```

Related topics

Basics

Automation Using Python Scripts.. 12

References

Overview of the Object Model for Accessing ModelDesk Experiments (ModelDesk Project and Experiment Management 🕮)

Using the Interpreter

Introduction

The Interpreter lets you edit and execute Python commands and run Python scripts.

Where to go from here

Information in this section

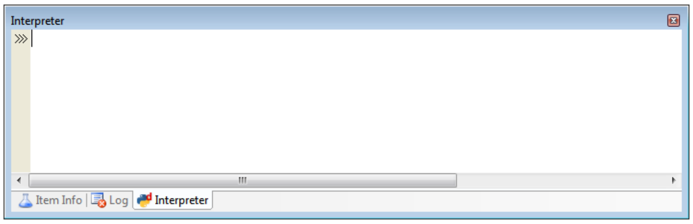
Basics on the Interpreter

Introduction

The Interpreter lets you edit and execute Python commands and run Python scripts.

Interpreter user interface

You can access ModelDesk's automation interface by entering commands interactively or running scripts in the Interpreter.



Editing features

You can use the Interpreter to edit and run Python commands. ModelDesk provides a set of features that makes editing easier and more efficient.

Syntax highlighting Helps you distinguish between the Python syntax items in your commands by highlighting them. For instructions, refer to How to Specify Syntax Highlighting on page 24.

Auto completion Completes Python variables, functions, and object attributes automatically so that you can save time and avoid spelling mistakes. For instructions, refer to How to Enable Auto Completion on page 25 and How to Use Auto Completion on page 27.

Auto indentation In Python, multi-line commands are introduced by a colon and their scopes are declared by indentation. The indentation indicates the structure of a multi-line command. The Interpreter indents the next line automatically. The indent depth of the line depends on the number of control structures or command blocks. You can decrease the indent depth by entering an empty line or pressing the **`Backspace`** key: for example, to close an **`If`** branch.

Command history The Interpreter stores a command history to let you quickly execute already executed commands during the current work session. During a work session, you might often repeat some commands with only minor changes. All the commands entered at the command prompt are stored in a command history. You can navigate through the command history by using the shortcut key combinations **`Ctrl+Up/Down`** to move up or down in the command history, and **`Ctrl+Home/End`** to go to the first or last command in the command history.

Shortcut keys The Interpreter supports the use of shortcut keys for Interpreter commands. For a list of the supported shortcut keys, refer to Interpreter on page 46.

Find/Copy/Cut/Paste You can use the standard Windows commands Find, Copy, Cut and Paste from the context menu of the Interpreter window or via shortcut keys. This feature helps you edit your commands more efficiently.

Drag & Drop You can move or copy selected text easily via drag & drop in the Interpreter controlbar.

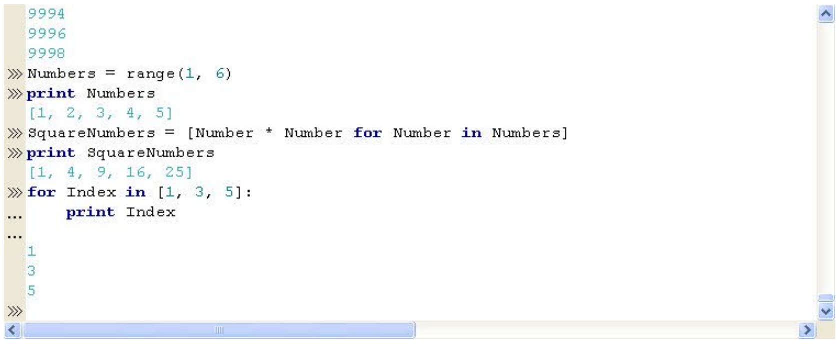
The **Interpreter** controlbar provides two kinds of drag & drop methods according to the position of the text:

- Input area

  Text in the input area is the text in the current input lines. You can select text in the input area and move or copy it via drag & drop. You can also copy text from other applications to the input area via drag & drop.
- History area

  Text in the history area is the text before the current input lines. You can select text from the history area and copy it to your current input line by pressing the *Ctrl* key and using drag & drop at the same time.

The following illustration shows an output example:



```
  9994
  9996
  9998
» Numbers = range(1, 6)
» print Numbers
  [1, 2, 3, 4, 5]
» SquareNumbers = [Number * Number for Number in Numbers]
» print SquareNumbers
  [1, 4, 9, 16, 25]
» for Index in [1, 3, 5]:
...     print Index
...
  1
  3
  5
»
```

The lowest line is the current input line. The lines above are history lines. The following symbols are used:

Symbol	Description
»	Input line. The lowermost input line is the current input line. The input lines above are history input lines.
...	Input line continuation. If you enter, for example, a control loop such as `for` and finish the line with a colon, the **Interpreter** automatically adds a new indented line. To add further lines enter code and then press **`Enter`**. To execute the multi line command leave the line empty and press **`Enter`**.
⬜	Output line.

Running scripts

You can run Python scripts directly in the **Interpreter**. For instructions, refer to How to Run Scripts on page 29.

Importing scripts

You can use external variables and methods in your commands and Python scripts by importing scripts. The variables and methods defined in the scripts are loaded into the **Interpreter**'s namespace. For instructions, refer to How to Import a Python Module to the Interpreter Namespace on page 28.

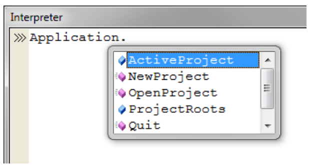
Specifying the Python path

When you import a Python module, the Interpreter searches for it in the folders of the Python path. You can list the folders and specify the Python path. For instructions, refer to How to Specify the Python Path on page 30.

Accessing the Running ModelDesk Application from the Interpreter

Introduction

The ModelDesk API lets you access the running ModelDesk application via the `Application` interface.



```
>>> Application.
```

Tip

You can use the `Application` interface as a starting point to browse the ModelDesk API reference information. Refer to Application (ModelDesk Project and Experiment Management 🕮).

How to Specify Syntax Highlighting

Objective

You can specify syntax highlighting to distinguish between Python syntax items in your commands.

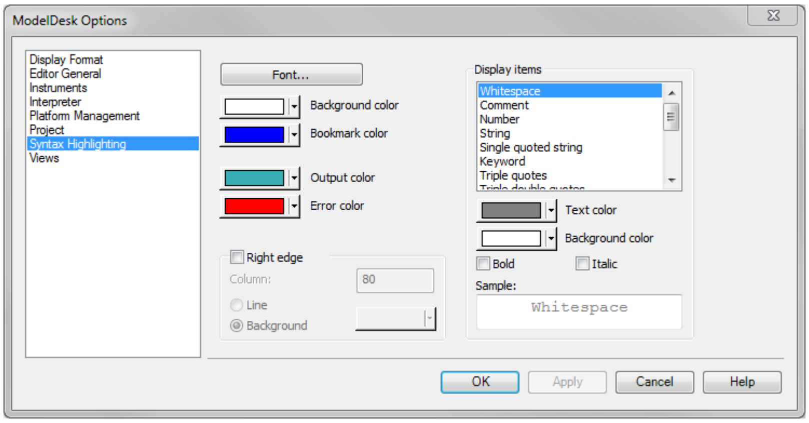
Basics

The Interpreter lets you specify general font and color settings, such as the interpreter's background color or the text color for errors. You can also specify how to display specific syntax items in the Python code. The syntax highlighting feature makes it easy to distinguish between the syntax items in the command lines.

Method

To specify syntax highlighting for a display item

1 On the File ribbon, click Options and change to the Syntax Highlighting page.



2 Specify general settings, such as the font or background color, to be used in the Interpreter.

3 From the Display Items list, select an item such as Comment or String, and configure the item's color and font settings.

4 Specify if and how a right edge is visualized in the Interpreter. The Interpreter can display a line or a background color to visualize that a code line exceeds a specified column limit.

5 Click OK to close the dialog.

Result

Syntax highlighting of a display item is specified.

Related topics

Basics

Basics on the Interpreter.. 21

References

Syntax Highlighting Page.. 57

How to Enable Auto Completion

Objective

To complete commands automatically in ModelDesk's Interpreter window, you have to enable the auto completion feature.

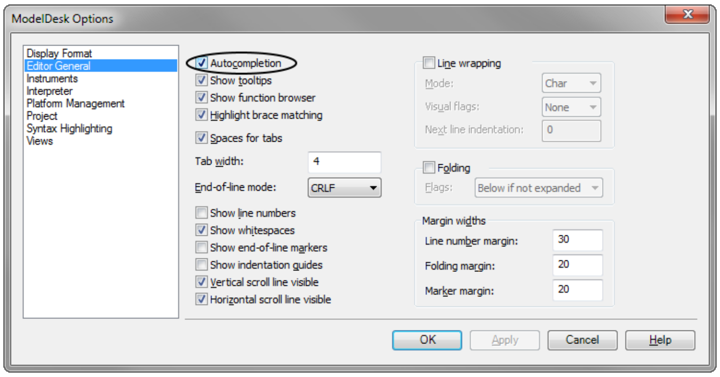
Basics

ModelDesk's Interpreter includes an auto completion feature that makes typing commands easy. This feature helps you enter the names of variables, methods, and objects correctly and quickly. You do not have to remember the full name or worry about the spelling mistakes.

Method

To enable/disable auto completion

1. On the File ribbon, click Options and change to the Editor General page.
2. On the Editor General page, select the Autocompletion checkbox to enable autocompletion or clear the checkbox to disable it.



✓ Autocompletion
✓ Show tooltips
✓ Show function browser
✓ Highlight brace matching
✓ Spaces for tabs
Tab width: 4
End-of-line mode: CRLF
☐ Show line numbers
✓ Show whitespaces
☐ Show end-of-line markers
☐ Show indentation guides
✓ Vertical scroll line visible
✓ Horizontal scroll line visible
☐ Line wrapping
Mode: Char
Visual flags: None
Next line indentation: 0
☐ Folding
Flags: Below if not expanded
Margin widths
Line number margin: 30
Folding margin: 20
Marker margin: 20

3. Click OK.

Result

You have enabled/disabled auto completion in the Interpreter.

Related topics

Basics

HowTos

References

How to Use Auto Completion

Objective

To edit commands in the Interpreter easily and more efficiently, you can use the auto completion feature in ModelDesk.

Method

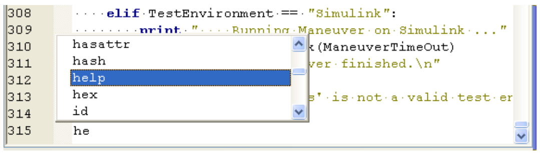
To use auto completion

1 In the Interpreter controlbar, enter the beginning letters of a Python variable/method/object of any length.

2 Press **Ctrl** + **SPACE**.

The Interpreter completes the matching Python variable/method/object name automatically.

If the result is ambiguous, a drop-down list appears with the Python variables/methods/objects that are currently known to the Interpreter's namespace.



```
    elif TestEnvironment == "Simulink":
        print "   Running Maneuver on Simulink ..."
                            (ManeuverTimeOut)
                            ver finished.\n"

                            s' is not a valid test er

he
```

3 To complete the command, select the entry and press **Return**.

> **Tip**
>
> You can also use the **Tab** key to select the entry and press **Return** or you can double-click the entry.

To close the selection list, press **Esc**.

Result

You have completed the Python variable, method, or object automatically.

Related topics

Basics

Basics on the Interpreter 21

HowTos

How to Enable Auto Completion 25

How to Import a Python Module to the Interpreter Namespace

Objective

You can import a Python module to use its variables and methods in the current command and script.

Basics

You can use external variables and methods directly in your current commands and Python scripts. To do so, you have to import Python modules defining variables and methods.

Note

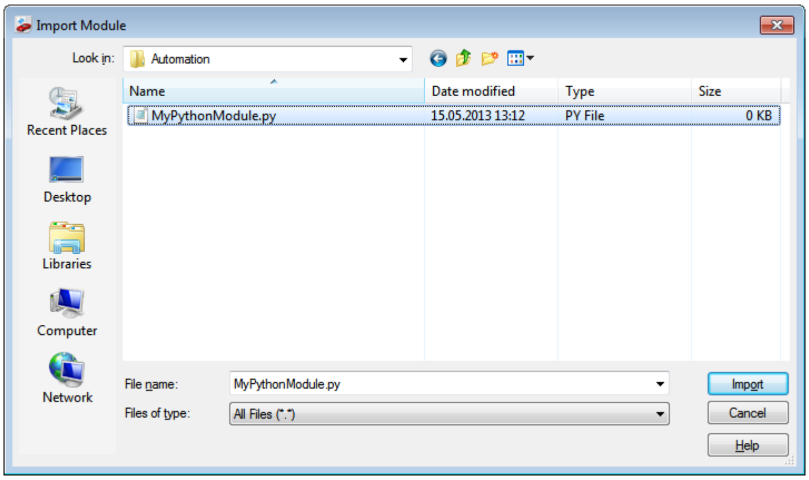
Unlike the `import <module_name>` Python command, the Import Module command overwrites the module if it was imported before, i.e., you do not need to clear the Interpreter namespace to reload a module. For details about namespace, refer to Clear Namespace on page 37.

Method

To import a Python module to the Interpreter namespace

1 On the Automation ribbon, click Interpreter – Import Module to open the Import Module dialog.

2 In the Import Module dialog, specify the Python file.



Name	Date modified	Type	Size
MyPythonModule.py	15.05.2013 13:12	PY File	0 KB

File name: MyPythonModule.py
Files of type: All Files (*.*)

3 Click OK.

Result

The module is imported into the Interpreter's namespace.

Related topics

Basics

References

How to Run Scripts

Objective

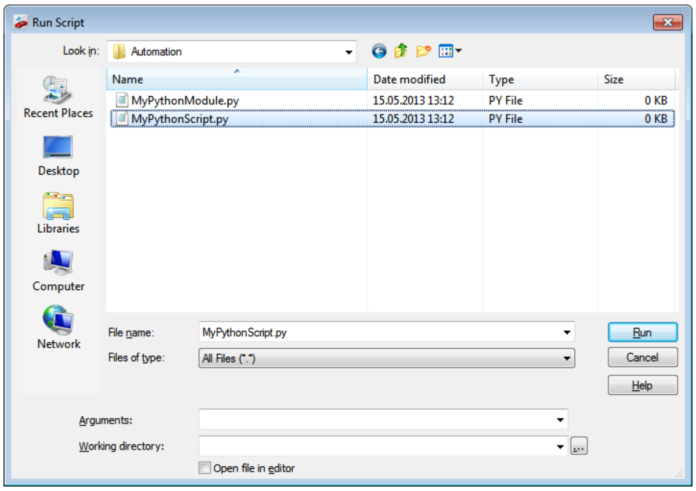
You can run a Python script in ModelDesk's Interpreter to execute Python commands automatically to do tasks such as building a project or validating specified elements.

Method

To run a Python script

1 From the context menu of the Interpreter, select Run Script, or press `Ctrl+R`, or go to the Automation ribbon and click Interpreter – Run Script.

The Run Script dialog opens.



Look in: Automation

Name	Date modified	Type	Size
MyPythonModule.py	15.05.2013 13:12	PY File	0 KB
MyPythonScript.py	15.05.2013 13:12	PY File	0 KB

File name: MyPythonScript.py
Files of type: All Files (*.*)
Arguments:
Working directory:
☐ Open file in editor

2 In the Run Script dialog, search the file system or use the drop-down lists to select an item that was recently used.

3 If necessary, specify additional arguments.

4 If necessary, specify a working directory. If you specify one, the Interpreter sets this as the current directory before executing the script.

> **Tip**
>
> The Python path includes the current directory automatically. You can use the drop-down list to select a directory that was recently used.

5 If you want to display the source code in the Source Code Editor, activate Open file in editor.

6 Click OK to run script execution.

Result

The script is executed. Standard and error outputs are displayed in the Interpreter controlbar.

Related topics

Basics

References

How to Specify the Python Path

Objective

You have to specify the Python path to tell ModelDesk's Interpreter where to find the imported scripts.

Basics

The Python path is the list of directories Python goes through to search for modules and files.

When you import a Python module using the `import <module name>` command, the Interpreter searches the folders of the Python path for Python files of the same name with the extension PY, PYC, or PYD.

If you have developed reusable modules as libraries, you do not have to keep them in your local working folder. You can create a subfolder for them and add it to the Python path so that you can use the functions in your main script.

Note

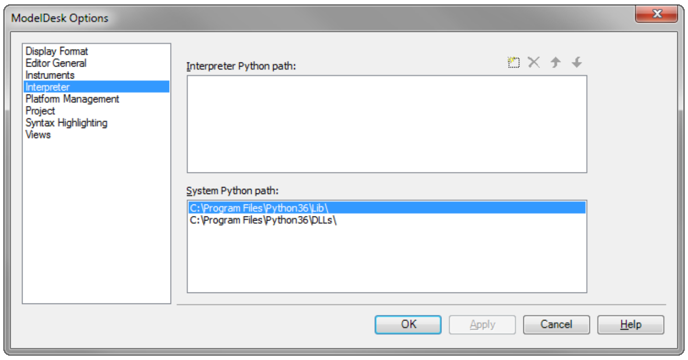
Additional Python paths specified for the Interpreter are ignored by external Python interpreters. They can only use System Python paths. System Python paths are not editable in this dialog.

Method

To specify the Python path

1 On the File ribbon, click Options.

2 Select the Interpreter page.



C:\Program Files\Python36\Lib\
C:\Program Files\Python36\DLLs\

3 On the Interpreter page, click to select a folder and click OK. The selected folders, such as `C:\TEMP` or `C:\OwnPythonDir`, are added to the Interpreter Python path list.

4 To change the search order of the paths, select it and click ↑ and ↓ to move the entry to the required position.

5 To remove a path, click an entry and click ✕.

6 Click OK to apply the changes.

Result

The specified path is appended to the list of directories which the Interpreter searches for Python modules.

Related topics

Basics

Basics on the Interpreter.. 21

Reference Information

Interpreter and Source Code Editor

Overview

ModelDesk has an internal Python interpreter and Source Code Editor for the automation task. They provide the following commands and dialogs.

Where to go from here

Information in this section

Information in other sections

Check Syntax

Access

You can access this command via:

Ribbon	Automation – Edit Script
Context menu of	Source Code Editor
Shortcut key	`Shift+Ctrl+C`
Icon	✓

Purpose

To check the syntax of the code in the selected Python script.

Result

If the syntax of your Python code is correct, a dialog informs you that the syntax check finished successfully. Otherwise a dialog shows you the syntax errors in the code.

Description

To avoid error messages concerning syntax errors in your Python scripts when executing, you should check the syntax beforehand directly after editing in the Interpreter or Source Code Editor.

Clear Bookmarks

Access

You can access this command via:

Ribbon	Automation – Edit Script – Advanced
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To remove all of the bookmarks in the source code file shown in the Source Code Editor.

Result

The Source Code Editor removes all of the bookmarks in the source code file that you toggled via Toggle Bookmark (see Toggle Bookmark on page 60).

Related topics

References

Toggle Bookmark.. 60

Clear Namespace

Access

You can access this command via:

Ribbon	None
Context menu of	Interpreter
Shortcut key	None
Icon	❌

Purpose

To clear all user-defined variables and all the imported modules from the interpreter's namespace since the last application start.

Result

- All the variables that were defined in the Interpreter namespace since the last start of the application are deleted.

- All the imported Python modules are unloaded.

> **Note**
>
> Imported PYD modules, i.e., Python modules programmed in C/C++, and modules used by the Interpreter itself, are *not* unloaded.

Description

You can define variables and import modules when using the Interpreter. To reset the Interpreter to its initial state, use the **Clear Namespace** command. Clearing the namespace is useful if a script error occurred due to user-defined variables or imported modules.

Related topics

Basics

Clear Window

Access

You can access this command via:

Ribbon	None
Context menu of	Interpreter
Shortcut key	None
Icon	❎

Purpose

To clear the current contents in the **Interpreter**.

Result

The contents of the Interpreter are cleared. The command prompt in the first line remains.

> **Tip**
>
> The commands entered in the Interpreter remain valid and are accessible via the command history.

Related topics

Basics

Basics on the Interpreter 21

Comment out Region

Access

You can access this command via:

Ribbon	Automation – Edit Script – Advanced
Context menu of	Source Code Editor – Source Code
Shortcut key	`Alt+3`
Icon	None

Purpose

To insert double comment character(s) at the beginning of all the lines selected in the source code file displayed in the Source Code Editor.

Note

A double comment character (##) is inserted to distinguish between a user comment and a comment-out region.

Result

The Source Code Editor places comment character(s) at the beginning of the lines you selected in the source code file.

Related topics

References

Uncomment Region 62

Convert End-of-Line Characters

Access

You can access this command via:

Ribbon	Automation – Edit Script – Advanced
Context menu of	Source Code Editor - Source Code

Shortcut key	None
Icon	None

Purpose

To convert all end-of-line characters according to operating system conventions.

Description

Operating systems have different conventions for end-of-line characters (CRLF, LF, or CR). The Source Code Editor converts the existing end-of-line characters according to the End-of-line mode setting in the ModelDesk Properties dialog (see Editor General Page on page 40). To display the end-of-line characters in the Source Code Editor, activate Show end-of-line markers on the same dialog page.

Editor General Page

Access

This page is part of the ModelDesk Options dialog.

Purpose

To specify settings for the Source Code Editor and the Interpreter.

Editor General page

Autocompletion Lets you enable/disable ModelDesk's auto-completion function.

Show tooltips Lets you enable/disable the display of tooltips.

Show function browser Lets you enable/disable the display of a drop-down list at the top of the editor. You can jump to a function in the code via this list.

Highlight brace matching Lets you enable/disable the highlighting of matching braces.

Spaces for tabs Lets you enable/disable tabs-to-spaces conversion.

Tab width Lets you define the tab width.

End-of-line mode Lets you select one of the following end-of-line modes:

Mode	Description
CRLF	Carriage return - line feed, used by MS-DOS and Windows operating systems
LF	Line feed, used by Unix operating systems
CR	Carriage return, used by Macintosh operating systems

The mode is only used for new end-of-line characters that are typed in. Existing end-of-line characters are not converted.

Show line numbers Lets you specify whether to display line numbers.

Show whitespaces Lets you specify whether to display symbols for whitespaces and tabulators.

Show end-of-line markers Lets you specify whether to display end-of-line markers.

Show indentation guides Lets you specify whether to display indentation lines.

Vertical scroll line visible Lets you specify whether to display a vertical scroll line.

Horizontal scroll line visible Lets you specify whether to display a horizontal scroll line.

Line wrapping Lets you enable/disable line wrapping.

Line wrapping - Mode Lets you select one of the following modes for line wrapping:

Mode	Description
Char	The line is wrapped after the last visible character in the current view.
Word	The line is wrapped after complete words only.

Line wrapping - Visual flags Lets you select a visual flag for wrapped lines:

Flag	Description
None	No visual flag is used.
End	A symbol ↵ is displayed at the end of the wrapped line.
Start	A symbol ↳ is displayed at the beginning of the wrapped line.

Line wrapping - Next line indentation Lets you specify the number of characters by which the next line is indented.

Folding Lets you activate folding. If folding is active, you can fold parts of the source code to get a better overview. Folded parts are marked with a right arrow ▶ in the folding margin. Expanded parts are marked with a down arrow ▼. The folding margin must be greater than 0 (see Folding margin).

Folding - Flags Lets you select where the folding line is displayed.

Line number margin Lets you specify the width of the line number margin in pixels.

Folding margin Lets you specify the width of the folding margin in pixels (see Folding).

Marker margin Lets you specify the width of the marker margin in pixels.

Related topics

References

Options (ModelDesk Basics 🕮)

Export Script

Access

This command is available only if the selected Python script belongs to the active project/experiment. You can access this command via:

Ribbon	Automation – Python Scripts
Context menu of	Project Manager
Shortcut key	None
Icon	

Purpose

To export a Python script.

Result

Opens a standard file dialog to specify path and file name.

Related topics

References

Import Script.. 44

Find (Interpreter)

Access

You can access this command via:

Ribbon	Automation – Edit Script
Context menu of	▪ Interpreter ▪ Source Code Editor
Shortcut key	`Ctrl+F`
Icon	

Purpose

To find text in the Interpreter or Source Code Editor.

Result

ModelDesk opens the Find dialog for you to locate the text in the source code.

Find dialog

Find what Lets you specify the text you wish to find in the code.

Match whole word only Lets you search only for text that is separated from the surrounding text by whitespace or tabs.

Match case Lets you search only for expressions which have the same case as the text entered in Find what.

Use regular expressions Lets you use regular expressions for your search:

Regular Expression	Purpose	Example
`.`	Matches any single character.	`ho.se` matches horse and house
`\<`	Matches the start of a word.	`\<art` matches artist but not start.
`\>`	Matches the end of a word.	`\>art` matches start but not artist.
`[...]`	Matches a single character that is contained within the brackets. For example, `[abc]` matches `"a"`, `"b"`, or `"c"`. `[a-z]` specifies a range which matches any lowercase letter from `"a"` to `"z"`.	`a[ur]t` matches auto or artist but not alternate
`[^...]`	Matches a single character that is not contained within the brackets. For example, `[^abc]` matches any character other than `"a"`, `"b"`, or `"c"`. `[^a-z]` matches any single character that is not a lowercase letter from `"a"` to `"z"`.	`a[^ur]t` matches alternative but not automobile or artist
`^`	Matches the start of a line.	`^dig` matches digit at the start of a line.
`$`	Matches the end of a line.	`git$` matches digit at the end of a line.
`*`	Matches the preceding character zero or more times.	`di*git` matches dgit, digit, diigit, diiigit, etc.
`+`	Matches the preceding character one or more times.	`di+git` matches digit, diigit, diiigit, etc.
`\x`	Allows you to use a character x that would otherwise have a special meaning.	`2\+2<5` matches 2+2<5

Direction Lets you select Up or Down as the search direction.

Find Next Starts or continues the search.

Related topics

Basics

Using the Interpreter.. 21

Go to Line

Access

You can access this command via:

Ribbon	Automation – Edit Script
Context menu of	Source Code Editor
Shortcut key	`Ctrl+G`
Icon	

Purpose

To go to any line in the source code shown in the Source Code Editor.

Result

The cursor goes to the line number you selected in the dialog.

Import Script

Access

You can access this command via:

Ribbon	Automation – Python Scripts
Context menu of	None
Shortcut key	None
Icon	

Purpose

To import a Python script to the active project or experiment.

Result

Opens the Import Python Script dialog for you to select a Python script to import.

Dialog settings

File name Enter the file name of the Python script you want to import.

Insert into Lets you select whether the Python script is imported to the project or the active experiment.

Related topics

References

Export Script.. 42

Import Module

Access

You can access this command via:

Ribbon	Automation – Interpreter
Context menu of	Interpreter
Shortcut key	`Ctrl+I`
Icon	🖼

Purpose

To add variable and function definitions of the selected Python module to the Interpreter namespace.

Result

The variables and function definitions in the specified module are added to the namespace of the Interpreter.

If the script is not a PYD module (a module programmed in C/C++), the script variables are reloaded to the Interpreter namespace.

Unlike the `import <module_name>` Python command, the Import Module command overwrites the module if it was imported before, i.e., you do not need to clear the Interpreter namespace to reload a module.

Dialog settings

File name Enter the file name of the Python script you want to import or select the file from the respective folder.

Related topics

HowTos

How to Import a Python Module to the Interpreter Namespace... 28

References

Clear Namespace.. 37

Interpreter Page

Access

This page is part of the ModelDesk Options dialog.

Purpose

To specify additional search paths for Python script files inside ModelDesk.

Interpreter Settings page

The page lets you specify additional search paths for Python script files.

Interpreter Python path Lists the user-defined Python directories in which the Interpreter searches for modules. The search order can be changed by moving directories up and down the list by the Up/Down buttons.

Buttons

Button	Description
⬚	Lets you open a directory selection dialog. After a directory is selected, it is added to the list of user-defined Python directories.
✕	Lets you delete the selected directory from the list.
↑	Lets you move the selected directory up the search order of the user-defined Python path.
↓	Lets you move the selected directory down the search order of the user-defined Python path.

System Python path Lists the permanent system Python directories. You cannot change the system Python path in this dialog.

For instructions, refer to How to Specify the Python Path on page 30.

Related topics

References

Options (ModelDesk Basics 📖)

Interpreter

Access

You can access this command via:

Ribbon	View – Controlbar – Switch Controlbars
Context menu of	None
Shortcut key	`Alt+Shift+6`
Icon	▨

Purpose

To show or hide ModelDesk's Interpreter.

Description

ModelDesk provides the Interpreter for you to run Python scripts and execute line-based commands.

Editing assistance ModelDesk's Interpreter provides editing assistance that helps you complete Python variables, functions, and object attributes automatically.

The editing assistance feature in ModelDesk's Interpreter comprises the following aspects:

- Auto completion
- Tooltip
- Member completion

The following table compares the aspects:

Aspect	Action	Meaning	Available
Auto completion	Enter `Ctrl+SPACE` after the beginning letter of a string	Lists all the possible names for you to choose from to complete the string.	Can be enabled or disabled
Tooltip	Enter "(" after a method	Shows a tooltip for the method.	Can be enabled or disabled
Member completion	Enter "." after an object	List the members of the object for you to choose from.	Always available

Note

The auto completion feature does not complete automatically or show you a list to choose from. Python is a dynamic language. The character of an object can only be read after the object has been instantiated. No editing assistance is shown if the string describing the object contains parenthesis.

Syntax highlighting The Interpreter features syntax highlighting for standard input. You can specify different colors and fonts for the syntax tokens using the Properties dialog of the Interpreter. This feature makes it easy to distinguish between the syntax tokens in the command lines.

Auto indentation In Python, multi-line commands are introduced by a colon, for example, the command `def f():` starts a command block that defines a function, and their scopes are declared by indentation. Each command line that is typed in is checked by the Interpreter. At the start of a multi-line command, the Interpreter automatically indents the next line. The indent depth of the line depends on the number of control structures or command blocks started. In addition, you can start a multi-line command by pressing the `Shift+ENTER` keys. The indent depth is decreased by entering an empty line or pressing the `BACKSPACE` key. For example, you have to do this to finish an `If` or `Else` branch.

Notation in the Interpreter controlbar

The Interpreter's ">>>" command prompt allows you to enter a command. A command is interpreted when you press the `Enter` key.

When you enter a multi-line command such as an If-clause or a function definition, the command prompt changes to "...".

Prompt	Meaning
>>>	Input line, start of command
...	Continued multi-line command
(Blanks)	Output text

Context menu of the Interpreter control bar

The Interpreter controlbar's context menu provides the following commands.

Command	Purpose
Undo	To reverse the last edit action you made in the Interpreter.
Redo	To reverse the latest undo action in the Interpreter.
Cut	To cut the selected text and add it to the Clipboard.
Copy	To copy the selected text and add it to the Clipboard.
Paste	To paste the current content from the Clipboard to the Interpreter.
Delete	To delete the selected text.
Select all	To select all the text in the Interpreter.
View Whitespace	To show or hide the whitespace marks in the Interpreter.
Find	To find text in the Interpreter.
Run script	To run a Python script.
Import script	To insert variable and function definitions of Python scripts or modules in the Interpreter's namespace.
Clear Window	To clear the current contents of the Interpreter.
Clear Namespace	To clear all user-defined variables since the last start of ModelDesk and all imported modules.

Shortcut keys of the Interpreter

The following table shows the shortcuts keys that can be used in the Interpreter control bar:

Shortcut Keys	Purpose
`Ctrl+F`	To open the Find dialog.
`Ctrl+W`	To enable/disable the whitespace view in the Interpreter.
`Ctrl+A`	To select all the current text in the Interpreter. This command is also available as "Select all" in the context menu of the Interpreter.
`Ctrl+Z`	To reverse the last action made in the Interpreter. This command is also available as "Undo" in the context menu of the Interpreter.

Shortcut Keys	Purpose
`Ctrl+Y`	To reverse the latest undo action in the Interpreter. This command is also available as "Redo" in the context menu of the Interpreter.
`Ctrl+`"Numpad +"	To zoom into the Interpreter control bar.
`Ctrl+`"Numpad -"	To zoom out of the Interpreter control bar.
`Ctrl+` Mouse wheel	To zoom into or out of the Interpreter.
`Backspace`	To decrease the indent depth in multi-line commands or delete the last character.
`Ctrl+Down`	To go one command down in the command stack.
`Ctrl+Up`	To go one command up in the command stack.
`Ctrl+End`	To go to the last command that was typed in at the command prompt.
`Ctrl+Home`	To go to the first command of the command stack.
`Ctrl+Space`	To enter the auto-completion mode for Python variables.
`Ctrl+X`	To cut the selected text to the Clipboard. This command is also available as "Cut" in the context menu of the Interpreter.
`Ctrl+C`	To copy the selected text to the Clipboard. This command is also available as "Copy" in the context menu of the Interpreter.
`Ctrl+V`	To paste text from the Clipboard to the current cursor position. This command is also available as "Paste" in the context menu of the Interpreter.
`Ctrl+I`	To open the Import Script dialog. This command is also available as "Import Script" on the Automation ribbon..
`Ctrl+R`	To open the Run Script dialog. This command is also available as "Run script" in the context menu of the Interpreter and on th ribbon.
`Esc`	To delete the current command in the line if the command is available.
`Shift+Enter`	To start a multi-line command.
`Tab`	To increase the indent depth in multi-line commands.

Error tracing

The preferred error handling in Python is exception handling. When an exception occurs, the Interpreter displays a traceback.

Typical errors To correct errors that have occurred during interactive command input, examine the printed traceback. Otherwise, edit the Python source file. These are some typical errors:

```
>>>dir("1", "1")
   Traceback (innermost last):
     File "<interactive input>", line 0, in ?
   TypeError: dir requires at most 1 argument; 2 given
```

```
>>>if () && 1:
     File "<string>", line 1
        if () && 1:
               ^
   SyntaxError: invalid syntax
>>>c
   Traceback (innermost last):
     File "<interactive input>", line 0, in ?
   NameError: c
```

Note

If a file path is available, you can double-click it above the error description. ModelDesk opens the file in the **Source Code Editor** and scrolls to the error position.

Predefined exceptions The following table is an extract from the predefined exceptions (refer to *Built-in Exceptions* in the *Python Library Reference*):

Exception Type	Additional Information	Cause
NameError	Name of the variable	Local or global name not found
SyntaxError	Invalid syntax and an extra line preceded by a "^", which points to the character where the syntax error starts.	Invalid syntax
TypeError	Depends on the specific error.	Built-in operation or function was applied to an object of an inappropriate type.
AttributeError	Depends on the specific error.	Object or variable has no member with the given name.
KeyboardInterrupt	–	The script was interrupted by using the system tray menu.
IOError	Depends on the I/O error.	A file system operation failed: for example, writing a file.
IndexError	Index out of range	Access to an index which is greater than the size of a list or tuple.
KeyError	Wrong key name	Key was not found in a dictionary's set.
ZeroDivisionError, OverflowError, FloatingPointError	–	Mathematical errors

Traceback information The traceback information always has the same structure:

```
Traceback (innermost last):
    File "<filename>", line <line number>, in <function name>
    <command>
<additional line when a syntax error has occurred>
<exception type>: <additional information>
```

`<filename>`	The file name is given whenever possible.
`<line number>`	The line number specifies either the script line in which the error occurred, or the line of the calling function.
`<function name>`	If a function name is given, it relates to the function where the error occurred or to the function that called the incorrect function.
`<command>`	If a command is given, it is the command belonging to the line number specified above.

Related topics

Basics

Insert Script

Access

This command is available only if you have opened a Python script.

You can access this command via:

Purpose

To insert a new empty Python script into the active project or experiment.

Result

The **Insert Python Script** dialog is opened for you to open a new empty Python script in the working area. The new Python script is inserted into the active project or experiment.

- You can create a new Python script that is not inserted into the project /experiment. For more information, refer to New (Script File) on page 52.
- You can import a "free floating" Python script into a project. For more information, refer to Import Script on page 44.

Insert Python Script dialog

Name of new Python script Enter the file name of the Python script you want to insert.

Insert into Lets you select whether the Python script is imported to the project or the active experiment.

Related topics

References

New (Script File)

Access

To access this command via:

Ribbon	Automation – Python Scripts – Script File
Context menu of	None
Shortcut key	None
Icon	🐍

Purpose

To create a new Python script.

Result

A new empty Python script is opened in the working area.

Description

The new script is not inserted into the active project/experiment.

- To create a new Python script that is inserted into the active project/experiment, refer to Insert Script on page 51.
- To insert a "free" Python script into a project, refer to Import Script on page 44.

Related topics

References

Next Bookmark

Access

You can access this command via:

Ribbon	Automation – Edit Script – Advanced
Context menu of	None
Shortcut key	`F2`
Icon	None

Purpose

To move the cursor to the next bookmark in the source code file shown in the Source Code Editor.

Result

The Source Code Editor moves the cursor and scrolls the view to the next bookmark in the source code file. For information on how to toggle bookmarks, refer to Toggle Bookmark on page 60.

Related topics

References

Toggle Bookmark.. 60

Previous Bookmark

Access

You can access this command via:

Ribbon	Automation – Edit Script – Advanced
Context menu of	None
Shortcut key	`Shift+F2`
Icon	None

Purpose

To move the cursor to the previous bookmark in the source code file shown in the Source Code Editor.

Result

The Source Code Editor moves the cursor and scrolls the view to the previous bookmark in the source code file. This bookmark was toggled via **Toggle Bookmark**.

Related topics

References

Redo / Redo List

Access

You can access this command via:

Ribbon	None
Context menu of	▪ Interpreter ▪ Source Code Editor
Shortcut key	`Ctrl+Y`
Icon	↷ ▾
Others	Quick access toolbar

Purpose

To redo the most recent commands or actions.

Result

Any command or action that was undone via the Undo command is performed once again.

Description

If you carry the command out once, one command or action is redone; if you carry it out twice, two are redone, and so on.

Tip

You can click the down arrow to get a list of the actions that can be redone. If you select one of the actions from the list, all actions up to that are redone.

Related topics

References

Replace

Access

You can access this command via:

Ribbon	Automation – Edit Script
Context menu of	None
Shortcut key	None
Icon	abc def

Purpose

To quickly replace an expression in the Source Code Editor.

Result

The expression is replaced with the text you specified.

Description

The Replace command is available only for the Source Code Editor.

Find and Replace dialog

Lets you specify search criteria.

Find what Lets you specify the text you wish to find in the source code.

Replace with Lets you specify the expression to replace the text you are searching for.

Match whole word only Indicates whether the text entered above is found only when it is separated from the surrounding text by white space or tabs.

Match case Indicates whether only expressions which have the same case as the text entered in Find what are found.

Use regular Expressions Indicates whether the specified string is a regular expression which the text must match.

In the search string,

- . matches any single character.
- * indicates there are 0, 1 or any number of the previous expression or character.

Find Next Starts or continues the search.

Replace Replaces the found text and then searches for the next occurrence of the text.

Replace All Replaces all occurrences of the find string in the document.

Run Script

Access

You can access this command via:

Ribbon	Automation – Interpreter
Context menu of	Interpreter
Shortcut key	`Ctrl+R`
Icon	

Purpose

To run a Python script.

Result

The selected Python script is executed in the namespace of the Interpreter. Standard outputs and error outputs are redirected to the Interpreter window.

Dialog settings

Look in Lets you specify the directory where the Python script to be run is stored.

File name Lets you specify the name of the Python script.

Files of type Lets you specify the file type. The selected file type must be Python Files.

Arguments If the Python script needs any command line arguments, you have to enter them here. The list of arguments to be entered depends on the selected script.

Working directory Lets you specify the working directory from which you want to run the Python script. You can use the Browse button to open a standard dialog to select a folder.

Open file in editor Lets you specify to open the selected file in the Source Code Editor when you start running it.

Related topics

HowTos

How to Run Scripts.. 29

Syntax Highlighting Page

Access	This page is part of the ModelDesk Options dialog.
Purpose	To alter the settings for the syntax highlighting in the Source Code Editor and the Interpreter.

Syntax Highlighting page

Font Opens the standard Windows Font dialog. You can specify the font, the font color, and the font size.

Background color Lets you select the interpreter's background color.

Bookmarks color Lets you select the color for bookmarks.

Output color Lets you select the text color for output in the Interpreter.

Error color Lets you select the text color for errors in the Interpreter.

Right edge Indicates whether the limit of the right edge is visible in the Source Code Editor or not (right edge visualization).

Right edge - Column Lets you specify the column on which right edge visualization starts.

Right edge - Line Indicates that right edge visualization is a line on the specified column.

Right edge - Background Indicates that right edge visualization is a background color for characters to the right of the specified column.

Right edge - Color Lets you choose a color for right edge visualization (background or line).

Display items Lets you choose a Python item to be configured from the list.

Display items - Text color Lets you choose the text color of the selected display item.

Display items - Background color Lets you choose the background color of the selected Python item.

Display items - Bold Indicates whether the selected Python item is displayed in bold.

Display items - Italic Indicates whether the selected Python item is displayed in italics.

Related topics

References

Options (ModelDesk Basics 🕮)

Tabify Leading Spaces

Access

You can access this command via:

Ribbon	Automation – Edit Script – Advanced
Context menu of	Source Code Editor – Source Code
Shortcut key	None
Icon	None

Purpose

To replace the leading whitespaces in the selected source code with a tabulator.

Result

The Source Code Editor replaces the leading whitespaces in the selected text with tabulators according to the number of whitespaces that was set as tabulator width in the **Editor General** page of the **ModelDesk Options** dialog.

Related topics

References

Tabify Region / Tabify Selection

Access

You can access this command via:

Ribbon	Automation – Edit Script – Advanced
Context menu of	Source Code Editor – Source Code
Shortcut key	None
Icon	None

Purpose

To replace the whitespaces in the selected source code with tabulators.

Result

The Source Code Editor replaces the whitespaces in the selected text with tabulators according to the number of whitespaces that was set as tabulator width in the **Editor General** page of the **ModelDesk Options** dialog.

Related topics

References

To Lowercase

Access

You can access this command via:

Ribbon	Automation – Edit Script – Advanced
Context menu of	Source Code Editor – Source Code
Shortcut key	None
Icon	None

Purpose

To convert all the characters in the selection to lower case.

Related topics

References

To Uppercase

Access

You can access this command via:

Ribbon	Automation – Edit Script – Advanced
Context menu of	Source Code Editor – Source Code
Shortcut key	None
Icon	None

Purpose

To convert all the characters inside the selection to upper case.

Related topics

References

Toggle all Folds

Access

You can access this command via:

Ribbon	Automation – Edit Script – Advanced
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To fold/unfold all the source code parts.

Description

The folding of source code parts must be activated in the ModelDesk Options dialog (see Editor General Page on page 40).

Result

Folds or unfolds all the source code parts in the script. A checkmark is displayed next to the command if all source code parts are folded.

Related topics

References

Toggle Bookmark

Access

You can access this command via:

Ribbon	Automation – Edit Script – Advanced
Context menu of	None
Shortcut key	`Ctrl+F2`
Icon	None

Purpose	To add/remove a bookmark to/from the line where the cursor is currently located.
Result	When you add a bookmark with this feature, it appears as a blue square in the left margin of the line where it was placed. If no margin is specified, the bookmark is indicated by a blue bar highlighting the whole line of text. When you move the cursor to a line which already contains a bookmark and activate the **Toggle Bookmark** command again, the bookmark is removed.

Related topics

References

Next Bookmark.. 53
Previous Bookmark.. 53

Toggle Fold

Access

You can access this command via:

Ribbon	**Automation – Edit Script – Advanced**
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To fold/unfold a selected source code part.

Description

The folding of source code parts must be activated in the **ModelDesk Options** dialog (see Editor General Page on page 40).

Result

Folds or unfolds the source code part the text cursor is in. A checkmark is displayed next to the command if a source code part is folded.

Related topics

References

Toggle all Folds.. 60

Uncomment Region

Access

You can access this command via:

Ribbon	Automation – Edit Script – Advanced
Context menu of	Source Code Editor – Source Code
Shortcut key	`Alt+4`
Icon	None

Purpose

To delete the comment character(s) you placed at the beginning of the line(s) selected in the source code file shown in the Source Code Editor.

Result

The leading comment characters of the selected source code lines are deleted. Further comment characters in the same line remain unchanged.

Related topics

References

Comment out Region.. 39

Undo / Undo List

Access

You can access this command via:

Ribbon	None
Context menu of	▪ Interpreter ▪ Source Code Editor
Shortcut key	`Ctrl+Z`
Icon	↶ ▾

Purpose

To undo the most recent edit commands or actions.

Result

The previously performed commands and actions are undone.

Description

This lets you undo the most recent edit commands or actions performed in the Interpreter or Source Code Editor. If you carry the command out once, one command or action is undone; if you carry it out twice, two are undone, and so on.

Tip

You can click the down arrow to get a list of the actions that can be undone. If you select one of the actions from the list, all actions up to that are undone.

Related topics

References

Redo / Redo List 54

Untabify Region / Untabify Selection

Access

You can access this command via:

Ribbon	Automation – Edit Script – Advanced
Context menu of	Source Code Editor – Source Code
Shortcut key	None
Icon	None

Purpose

To replace the tabulators in the selected source code with whitespaces.

Result

All leading tabulators in the selection are changed to a series of spaces. The number of spaces per tabulator depends on the tabulator width setting in the **Editor General** page of the **ModelDesk Options** dialog (see Editor General Page on page 40).

Related topics

References

Tabify Region / Tabify Selection 58

View Whitespace

Access

You can access this command via:

Ribbon	Automation – Edit Script
Context menu of	▪ Interpreter ▪ Source Code Editor
Shortcut key	`Ctrl+W`
Icon	a·b

Purpose

To toggle the display of whitespaces and tabulators in the source code.

Result

If you selected the View Whitespace option, the Source Code Editor or Interpreter displays the symbols used for spaces in the source code so you can see whether whitespaces or tabulators are used.

Troubleshooting

Introduction

If a problem related to the automation of ModelDesk comes up, the following topics provide a collection of possible malfunctioning scenarios and how to solve the problem.

Unexpected Error in Automation Script

Problem

In an automation script an unexpected error occurs when a time-consuming method is used.

Reason

When you use a time-consuming method, for example, the `SetActiveModel()` method of the `models` class, it can happen that the automation script continues although the method is not really completed.

Solution

Let the script pause, for example, by adding a `time.sleep(<seconds>)` command in Python or a `pause(<seconds>)` command in MATLAB.