AutomationDesk

Accessing ModelDesk

For AutomationDesk 6.5

Release 2021-A – May 2021

dSPACE

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

Contents

About This Document

Content

This document gives you information on how to access ModelDesk via AutomationDesk.

Required knowledge

Working with AutomationDesk requires:

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

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

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
⚠ 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

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files

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

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

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

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

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

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

Basics and Instructions

Where to go from here

Information in this section

Basics on ModelDesk

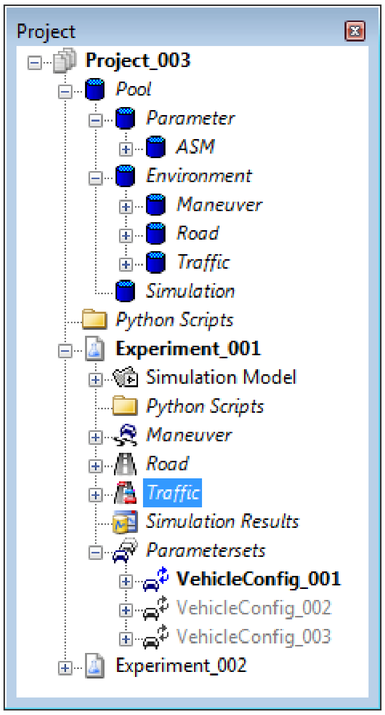
Introduction

ModelDesk is a user interface for parameterizing and managing the parameter sets of ASM models. It contains graphical tools for creating roads and planning maneuvers, and has parameter pages with illustrations for each modeled component as an aid to entering parameters.

ModelDesk elements

You can group parameterization tasks that belong together in a ModelDesk project.

The illustration below shows an example of a ModelDesk project. The main structure is always the same and is realized by specific ModelDesk elements.



Project

- Project_003
  - Pool
    - Parameter
      - ASM
    - Environment
      - Maneuver
      - Road
      - Traffic
    - Simulation
  - Python Scripts
  - Experiment_001
    - Simulation Model
    - Python Scripts
    - Maneuver
    - Road
    - Traffic
    - Simulation Results
    - Parametersets
      - VehicleConfig_001
      - VehicleConfig_002
      - VehicleConfig_003
  - Experiment_002

Pool This is the container for all the project-specific files, such as the parameter files of the simulation models and the ASM environment model files. These files can be used in each experiment of the project.

Experiment This is the basis for carrying out a parameterization task on one specific simulation model. To experiment with another simulation model, you have to add a second experiment to the project.

An experiment defines one or more parameter sets that contain the parameterization of the simulation model.

Simulation Model An Automotive Simulation Model (ASM) is a set of Simulink® models for hardware-in-the-loop testing of electronic control units or for early validation of controller algorithms during the design phase.

In ModelDesk, you can parameterize the following simulation model types:

- Real-time simulation

  A simulation running on a real-time system based on a DS1005 or DS1006, or a SCALEXIO system
- VEOS simulation

  An offline simulation running with VEOS on a host PC
- Simulink simulation

  An offline simulation running in Simulink®

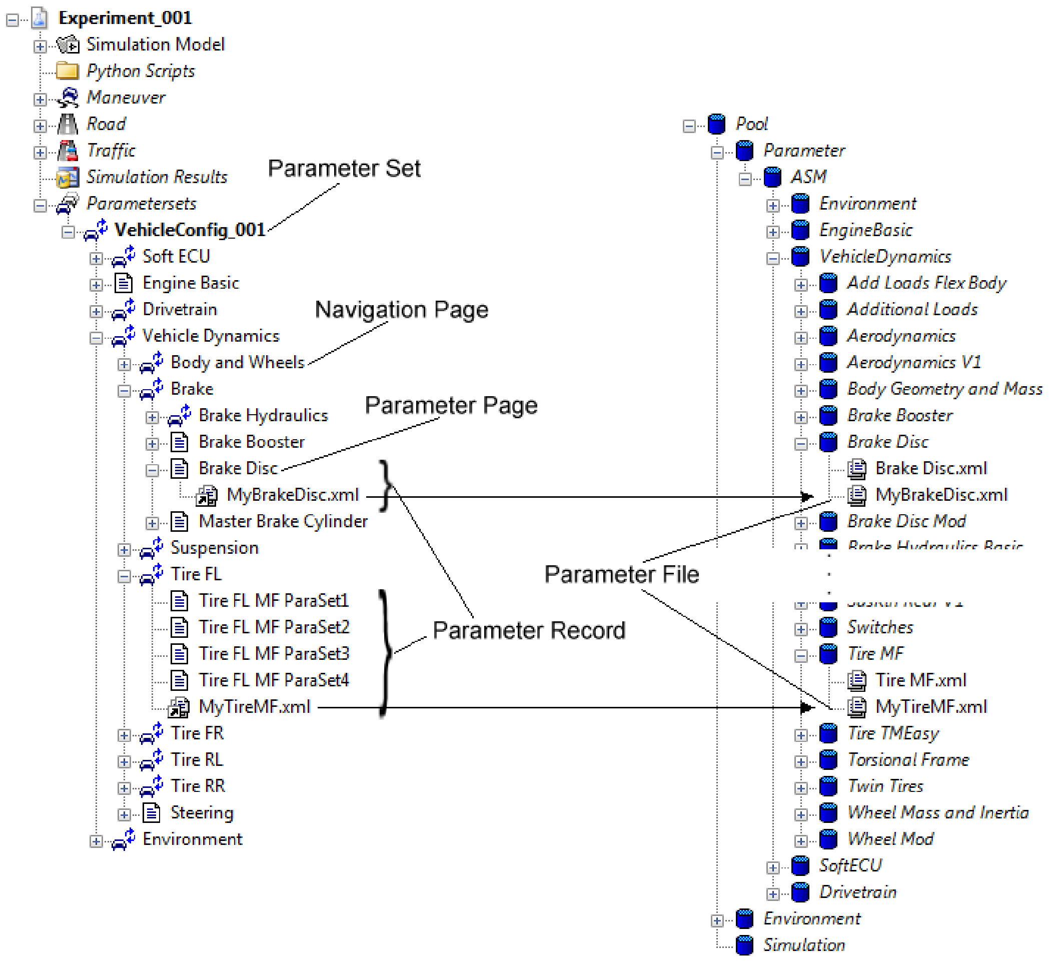
Maneuver You can create maneuvers with ModelDesk's Maneuver Editor. In a maneuver you can define a vehicle's movements on a road or across an unspecified area.

Road You can create roads with different surfaces, profiles and courses, such as junctions, with ModelDesk's Road Generator.

Traffic scenario You can create traffic scenarios that define the movements of fellow vehicles relative to the ASM vehicle with ModelDesk's Traffic Editor.

Parameter set A parameter set defines one parameterization of the experiment's ASM vehicle model. It is an aggregation of parameter records, one for each model component. A parameter record links a model component to a parameter file in the project's Pool. A parameter file is an XML file that contains the parameters of a model component with their values.

ModelDesk parameter pages are a convenient way to modify a parameter value in a parameter file. Parameter pages visualize the parameters of a model component. You can navigate to them via graphics that reflect the model structure in navigation pages. The modified parameters can be stored in the project's Pool in a parameter file with a new name.



Activating ModelDesk elements

Although a ModelDesk project can contain several experiments, you can only work with one experiment at a time. To select an experiment for further processing, you have to activate it. The same applies to roads, maneuvers, traffic scenarios and parameter sets.

Automating ModelDesk

You can write scripts to control ModelDesk via a COM-based automation interface. For more details, refer to ModelDesk Automation 🕮.
AutomationDesk's **ModelDesk Access** library uses this interface to provide automation blocks for some basic ModelDesk features. The blocks offer only a subset of the automation interface, but additionally, there are blocks in the ManeuverControl library folder to automate functionality of ControlDesk for experimenting.

Demo Projects

For an example of a ModelDesk project, refer to `<RCP_HIL_InstallationPath>\Demos\ASM\Models\VehicleDynamics\Parameterization.current`.

Related topics

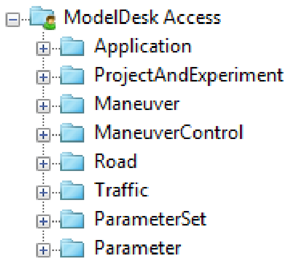
Basics

ModelDesk Automation
Features of ModelDesk (ModelDesk Basics 🕮)

Overview of the ModelDesk Access Library

Library overview

AutomationDesk's **ModelDesk Access** library is grouped in library folders which represent the elements of a ModelDesk project.



ModelDesk Access
- Application
- ProjectAndExperiment
- Maneuver
- ManeuverControl
- Road
- Traffic
- ParameterSet
- Parameter

The library folders group the provided automation blocks in the following categories.

Application This folder provides automation blocks to start and close ModelDesk. You can specify to save all project files when you close ModelDesk.

ProjectAndExperiment This folder provides automation blocks to open and close a ModelDesk project and to activate and close an experiment in it.

If you want to keep the modifications you made to the active experiment, you can specify to save it. The former version of the experiment is overwritten.

Road, Maneuver and Traffic Each of these folders provides automation blocks to activate and download an environment element, i.e., a road, a maneuver or a traffic scenario.

First activate the environment element you want to work with. Activation means that the environment element is selected for later downloading and it is registered as an active element in the ASM environment model configuration.

If you modify an environment element, for example, by using ModelDesk's automation interface, you can save it by overwriting its former version.

You can download an active element to the simulation platform that is specified in the active experiment.

ParameterSet This folder provides automation blocks to activate, save and download ModelDesk parameter sets and parameter records.

You can activate one parameter set, i.e. select it for later modification and downloading. If you modified a parameter set by using the automation blocks of the **Parameter** folder, you can save it by overwriting its former version. You can download an active parameter set to the simulation platform that is specified in the active experiment.

You can also change the parameterization of a model component by linking the parameter record to a different parameter file.

Parameter This folder provides automation blocks to read and write parameter values of the active parameter set.

If the parameter type is known, you can use type-specific automation blocks. Otherwise, you can use blocks for reading and writing parameter values generically.

ManeuverControl This folder provides automation blocks to control the execution of ModelDesk maneuvers. The blocks extent the functionality offered by ModelDesk's automation interface.

You can start, stop and reset a downloaded maneuver and retrieve its execution state.

The automation blocks to control ModelDesk maneuvers are only applicable to maneuvers running the **ASM Vehicle Dynamics** model.

For detailed information on the automation blocks, refer to Automation Blocks (AutomationDesk Basic Practices 📖).

The ModelDesk Access library is implemented as a custom library. This means that you can manage it like any other custom library, for example, you can open, close, export and import it.

Note

In this document there are some cross-references to the ModelDesk documentation. If a linked topic is not found, open dSPACE Help via the Help shortcut in the Windows Start menu for dSPACE RCP and HIL software.

Access via Exec block

The Python module `audmodeldeskaccess` provides methods for all automation blocks except the maneuver control blocks. Each method has the same name as its block, its arguments correspond to the block’s input data objects, and its return value corresponds to the output data object.

Example

```
import audmodeldeskaccess
MyVehicleMass = \
    audmodeldeskaccess.GetScalarParameter( \
        "VehicleDynamics.VEHICLE_MASS_AND_ADDITIONAL_LOADS.Const_m_Vehicle")
```

Demo Projects

For an example of automating ModelDesk access, refer to the AutomationDesk demo project at `<DocumentsFolder>\ModelDesk Access`.

Related topics

HowTos

References

Automation Blocks (AutomationDesk Basic Practices 📖)

How to Build a Basic Sequence for Accessing ModelDesk

Objective

Using AutomationDesk's ModelDesk Access library, you can build a basic sequence that contains generic steps to automate ModelDesk access.

Generic steps of ModelDesk access

ModelDesk must be opened and an experiment must be activated for each ModelDesk use case. Roads, maneuvers and traffic scenarios can be optionally activated and downloaded. After testing, for example, by analyzing different parameter values, ModelDesk must be closed.

Preconditions

- ModelDesk as of version 2.5p1 must be installed on the host PC. You need the same licenses as to execute the tasks manually using ModelDesk.
- The ModelDesk elements you want to access must exist in the ModelDesk project.
- The simulation application that is configured in the ModelDesk experiment must be loaded to the simulation platform, for example, by using the automation blocks from the Platform Management library.
- The following information is required as input data:
  - The name and path of the ModelDesk project file (CDP) you want to open
  - The name of the ModelDesk experiment you want to access
  - The names of the road, maneuver and traffic scenario you want to download

Method

To build a basic sequence for accessing ModelDesk

1 Add the following data objects to your project to parameterize the input data:
  - A File data object

    In the Data Object Editor, parameterize it with the file name and path of the project file (CDP) of the ModelDesk project you want to open.

- A String data object

  Parameterize it with the name of the experiment you want to activate.
- A String data object

  Parameterize it with the name of the maneuver you want to activate and download.
- A String data object

  Parameterize it with the name of the road you want to activate and download.

2 From the Library Browser, drag a StartModelDesk block from the ModelDesk Access library to the Sequence Builder to get an instance of ModelDesk. If ModelDesk is already running, the existing instance is used.

3 Drag an OpenProjectAndExperiment block from the ModelDesk Access library to the Sequence Builder. This opens the specified project and activate the specified experiment.

4 In the Data Object Editor, set the block's ProjectFile and InitialExperiment data objects as references to the project-specific File and String data objects that contain the ModelDesk project file and the experiment name.

5 Drag an ActivateManeuver block from the ModelDesk Access library to the Sequence Builder to activate the specified maneuver.

6 Set the block's ManeuverName data object as a reference to the project-specific String data object that contains the maneuver name.

7 Drag a DownloadActiveManeuver block from the ModelDesk Access library to the Sequence Builder. This downloads the active maneuver to the simulation platform.

8 Drag an ActivateRoad block from the ModelDesk Access library to the Sequence Builder to activate the specified road.

9 Set the block's RoadName data object as a reference to the project-specific String data object that contains the road name.

10 Drag a DownloadActiveRoad block from the ModelDesk Access library to the Sequence Builder to download the active road to the simulation platform.

11 Drag a CloseProjectAndExperiment block from the ModelDesk Access library to the Sequence Builder. This closes the active experiment and the ModelDesk project.

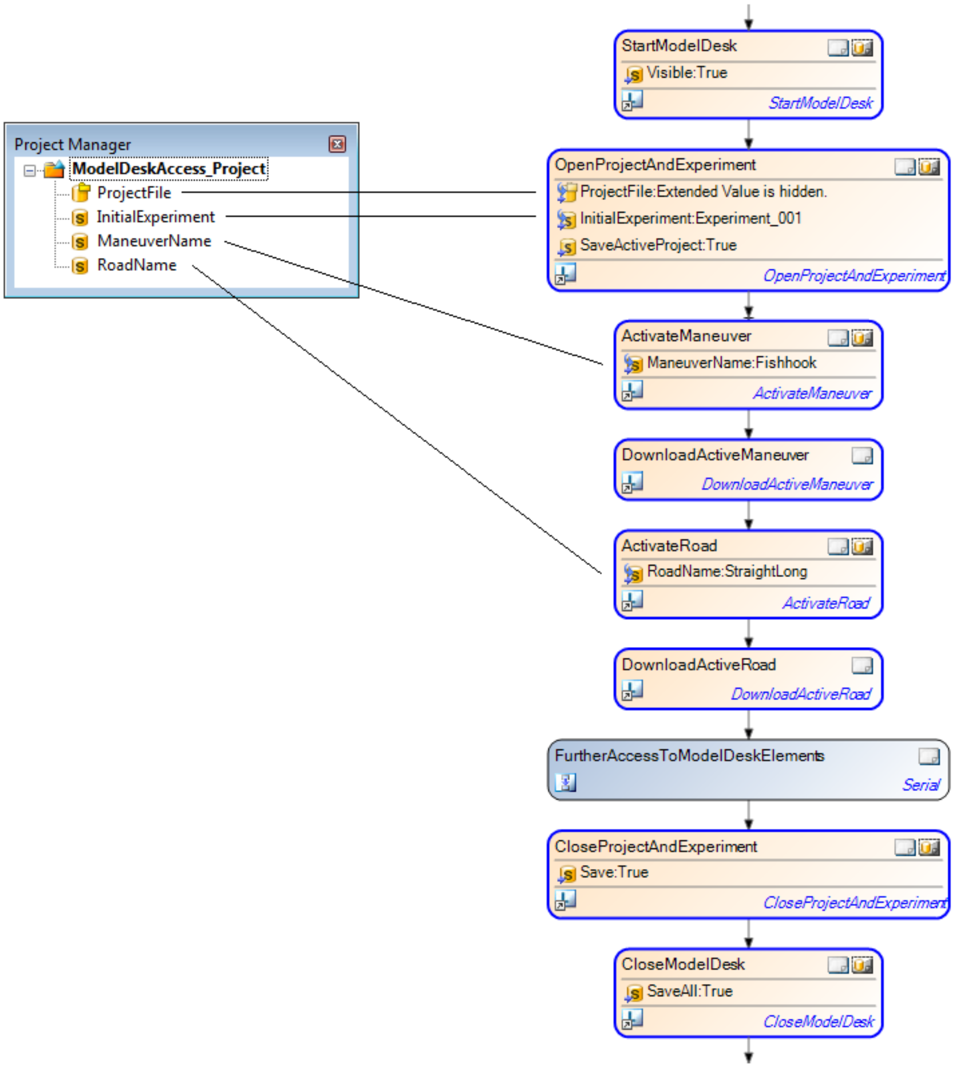
12 Drag a CloseModelDesk block from the ModelDesk Access library to the Sequence Builder. This closes the existing ModelDesk instance.

Note

This will also close a ModelDesk instance that you started manually.

Result

You created a basic sequence to automate the generic steps to access ModelDesk.



When you run the sequence, a ModelDesk instance is created or accessed to open the specified project and to activate the specified experiment. The ASM maneuver and road elements are downloaded to the simulation platform. The ModelDesk access is finished by saving and closing the project with its experiment and by closing ModelDesk.

Next steps

You can now parameterize your ASM model. Refer to How to Work with Parameters on page 16.

Related topics

Basics

References

How to Work with Parameters

Objective

You can modify and save a parameter set of an ASM model and download it to the simulation platform with AutomationDesk's ModelDesk Access library.

Access to model parameters

Model parameters are not accessed directly on the simulation platform. You can activate a parameter set to load it to your host PC's memory. There you can modify the parameters using the automation blocks from the Parameter folder. Then you can download the active parameter set from the PC's memory to the simulation platform to parameterize the current simulation application.

Format of parameter values

In AutomationDesk, the different model parameter types are represented by Python tuples, except for Scalars, which are represented as Float values.

Parameter Type	Representation
Scalar	v
Vector	(v1, v2, v3, ...)
Matrix	((v_a1, v_a2, ...), (v_b1, v_b2, ...), (v_c1, v_c2, ...)) (where a, b, c is the row index and 1, 2 is the column index)
LUT1D[1)]	((v1, v2, v3, ...), (x1, x2, x3, ...))
LUT2D[2)]	((v1, v2, v3, ...), (x1, x2, x3, ...), (y1, y2, y3, ...))

[1)] One-dimensional look-up-table
[2)] Two-dimensional look-up-table

If the parameter type is known, you can use type-specific automation blocks to read or write parameter values, such as GetVectorParameter or SetLUT1DParameter.

If the parameter type is unknown, you can use the automation blocks GetGenericParameterValue and SetGenericParameterValue to access parameters generically.

Preconditions

- You have built a basic sequence for accessing ModelDesk. For instructions, refer to How to Build a Basic Sequence for Accessing ModelDesk on page 13.
- The parameter set you want to access must exist in the ModelDesk project.
- The following information is required as input data:
  - The name of the parameter set you want to access and download
  - The addresses of the parameters you want to access

Method

To work with parameters

1 Add the following data objects to your project to parameterize the input data:
   - A String data object

     Parameterize it with the name of the parameter set you want to modify, save and download.
   - A String data object for every parameter you want to access

     Parameterize them with the addresses of the parameters, i.e., with their variable paths in the simulation model.

     For example, `VehicleDynamics.VEHICLE_MASS_AND_ADDITIONAL_LOADS.Const_PosVec_CoG_Vehicle` is the address of the Const_PosVec_CoG_Vehicle parameter in the ASM Vehicle Dynamics model.

2 For every parameter you want to access, add a data object to your project in order to reference the parameter value. The data type of the data object depends on the parameter type. For a Scalar parameter, add a Float data object. For all other parameter types, add a Tuple data object.

3 Drag an ActivateParameterSet block from the ModelDesk Access library to the Sequence Builder to activate the specified parameter set.

4 Set the block's ParameterSetName data object as a reference to the project-specific String data object that contains the parameter set name.

5 Add an automation block for the required parameter type to your sequence to read a parameter.

   For example, to read a Vector parameter, drag a GetVectorParameter block from the ModelDesk Access library to the Sequence Builder to read a Vector parameter.

6 Set the block's ParameterAddress data object as a reference to the project-specific String data object that contains the address of the parameter you want to read.

7 Set the block's V data object as a reference to the project-specific object to store the parameter value that you want to read.

In case of one-dimensional look-up tables, in addition, you must parameterize the x-axis. In case of two-dimensional look-up tables, in addition, you must parameterize the x- and the y-axis.

8 Add an automation block for the required parameter type to your sequence to write a parameter.

For example, to write a Vector parameter, drag a SetVectorParameter block from the ModelDesk Access library to the Sequence Builder.

9 Set the block's ParameterAddress data object as a reference to the project-specific String data object that contains the address of the parameter you want to write.

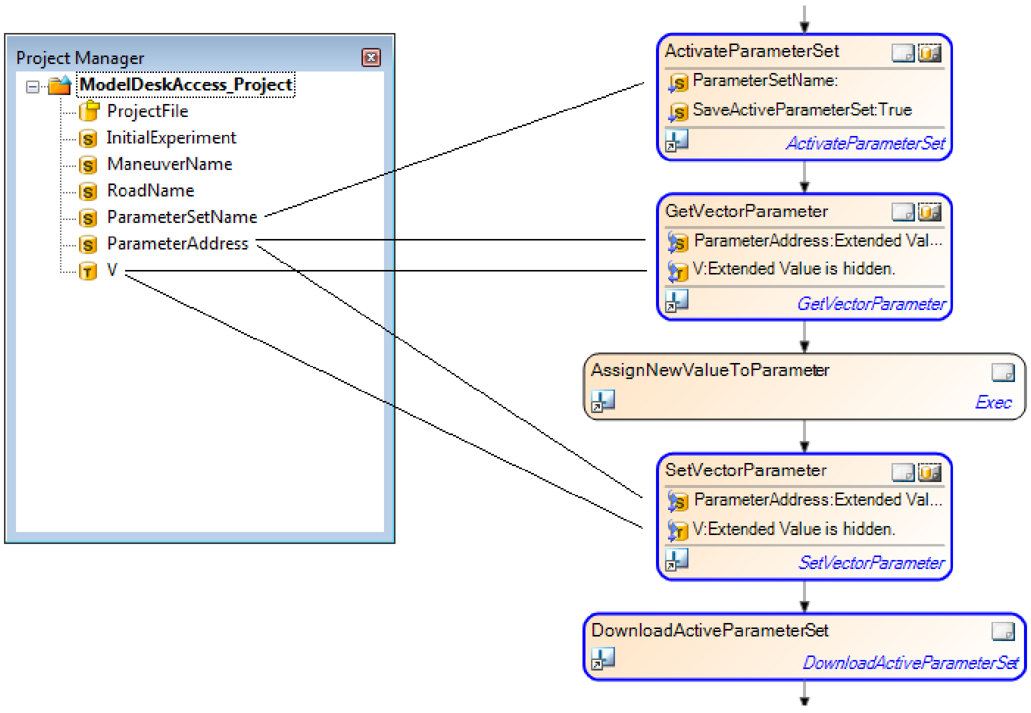
10 Set the block's V data object as a reference to the project-specific data object to store the parameter value you want to write.

In case of one-dimensional look-up tables, in addition, you must parameterize the x-axis. In case of two-dimensional look-up tables, in addition, you must parameterize the x- and the y-axis.

11 Drag a DownloadActiveParameterSet block from the ModelDesk Access library to the Sequence Builder. This downloads the currently active parameter set to the simulation platform.

Result

You created a sequence to automate the reading and writing of a parameter set. If you access a Vector parameter, the sequence looks like this.



When you run the sequence, the specified parameter set is activated. The specified Vector parameter is read. After the value is modified, the parameter is written to the active parameter set, which is then downloaded to the simulation platform.

For an example of modifying a parameterization iteratively, refer to the AutomationDesk demo project at `<DocumentsFolder>\ModelDesk Access`.

Next steps

You can execute ModelDesk maneuvers. Refer to How to Control Maneuvers on page 19.

Related topics

Basics

References

How to Control Maneuvers

Objective

You can control the execution of a maneuver with AutomationDesk's ModelDesk Access library.

Note

- The automation blocks to control ModelDesk maneuvers are only applicable to maneuvers running the ASM Vehicle Dynamics model.
- Up to and including ModelDesk 5.2, you can use the maneuver compatibility mode in ModelDesk to work with maneuvers created with ModelDesk 4.6 and earlier.
  As of ModelDesk 5.3, the maneuver compatibility mode is discontinued. Maneuvers created with ModelDesk 4.6 and earlier are no longer supported. Executing ModelDesk Access blocks to activate and download these maneuvers leads to exceptions.

Controlling the maneuver execution

The automation blocks to control maneuvers use the following parameters of the maneuver scheduler in the ASM environment model.

Parameter	Parameter Path
ResetVehicleVariable	`()://Model Root/Environment/Maneuver/UserInterface/PAR_Plant/ManeuverControl/RESET/MDLDCtrl_Reset`
ManeuverStartVariable	`()://Model Root/Environment/Maneuver/UserInterface/PAR_Plant/ManeuverControl/MANEUVER_START/MDLDCtrl_ManeuverStart`
ManeuverStateVariable	`()://Model Root/Environment/Maneuver/UserInterface/DISP_Plant/ManeuverState[]/Out1`
ManeuverStopVariable	`()://Model Root/Environment/Maneuver/UserInterface/PAR_Plant/ManeuverControl/MANEUVER_STOP/MDLDCtrl_ManeuverStop`

Note

For an MP model, the variable path has to be adapted accordingly.

The automation blocks to control maneuvers use preconfigured references to project-specific String data objects. If you create your project's data structure according to the preconfiguration, you do not have to adjust the references. For details, refer to ManeuverControl on page 35.

Tip

The required project contents are also in the ModelDeskAccessDemo project in `<DocumentsFolder>\ModelDesk Access`.

Retrieving the maneuver state

You can retrieve the state of the maneuver that is currently downloaded to the simulation platform by using the GetManeuverState automation block. The following maneuver states are defined.

Maneuver State1)	State Number2)	Description
Init	2	The maneuver is in its initialization phase.
Run	3	The maneuver is executing.
Stopping	4	The maneuver has just stopped or finished and the vehicle is coasting.
Wait	5	The maneuver stopped or finished and the simulation is waiting for restart.
Init Manual3)	6	The maneuver is in its initialization phase for manual driving.
Manual3)	7	The maneuver is executing in manual driving mode.

1) In ModelDesk Access library
2) In Maneuver Scheduler
3) Not applicable in terms of automation.

For details, refer to Maneuver Scheduler (ASM Environment Reference 🕮).

Preconditions

- The XIL API Convenience library must be opened in AutomationDesk's Library Browser, because its functionality is used internally by the ModelDesk Access library.
- You have built a basic sequence for accessing ModelDesk. For instructions, refer to How to Build a Basic Sequence for Accessing ModelDesk on page 13
- You have set up and downloaded the parameterization of your simulation. For instructions, refer to How to Work with Parameters on page 16.
- The following information is required as input data:
  - The configuration of the model access port. For details, refer to InitMAPort (AutomationDesk Accessing Simulation Platforms 🕮).
  - The variable paths of the control parameters of the maneuver scheduler in the ASM environment model.

Method

To control a maneuver

1. Add the following data objects to your project to parameterize the input data:
   - A data container named SimulationPlatform containing an MAPortConfiguration data object. Parameterize it with the platform identifier and the application path of the simulation platform specified in the ModelDesk experiment
   - A data container named VariablePool containing the following data objects:
     - A String data object named Reset_VehicleStates. Parameterize it with the variable path of the ResetVehicleVariable parameter.
     - A String data object named ManeuverStart. Parameterize it with the variable path of the ManeuverStartVariable parameter.
     - A String data object named ManeuverState. Parameterize it with the variable path of the ManeuverStateVariable parameter.
     - A String data object named ManeuverStop. Parameterize it with the variable path of the ManeuverStopVariable parameter.
2. Add the following data objects to your project to provide data objects for referencing:
   - An MAPort data object in the SimulationPlatform data container to store the instantiated model access port object.
   - A String data object to store the execution state of the maneuver.
3. Drag an InitMAPort block from the XIL API library's MAPort folder to the Sequence Builder. This initializes the access to the application that is running on the simulation platform.
4. Set the block's ConfigurationDict data object as a reference to the project-specific MAPortConfiguration data object that contains the platform identifier and the application path.
5. Set the block's MAPort data object as a reference to the project-specific MAPort data object in the SimulationPlatform data container.

If the project-specific data objects are created as described above, the data objects of the ManeuverControl blocks are automatically referenced.

6 Drag a ResetVehicle block from the ModelDesk Access library to the Sequence Builder. This sets the currently downloaded vehicle and maneuver on the simulation platform to its initial state.

7 Drag a StartManeuver block from the ModelDesk Access library to the Sequence Builder. This is starts the currently downloaded maneuver.

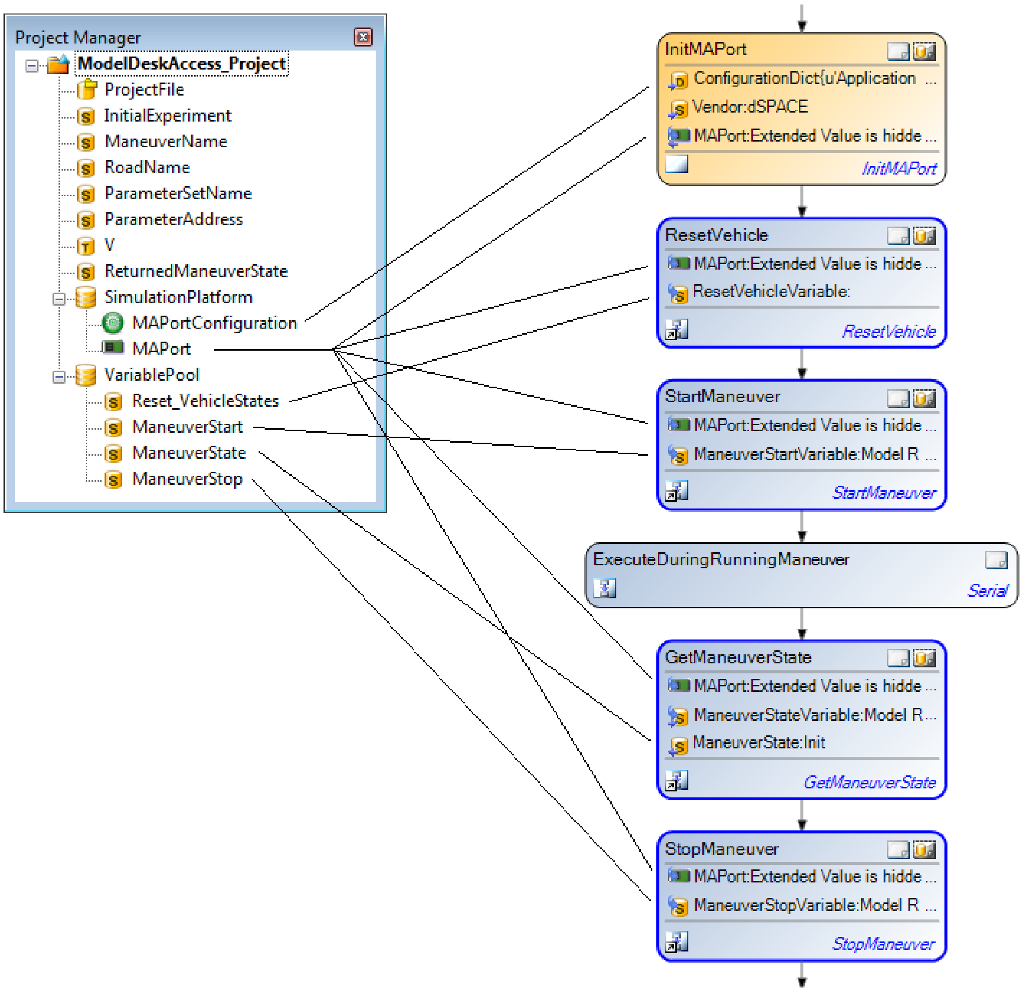
8 Drag a GetManeuverState block from the ModelDesk Access library to the Sequence Builder. This starts the currently downloaded maneuver.

9 Set the block's ManeuverState data object as a reference to the project-specific String data object to store the state of the currently downloaded maneuver.

10 Drag a StopManeuver block from the ModelDesk Access library to the Sequence Builder. This stops the currently downloaded maneuver.

Result

You created a sequence to automate the execution of the downloaded ModelDesk maneuver.



When you run the sequence, the vehicle is reset and the maneuver is started. The maneuver's execution state is retrieved and the maneuver is stopped.

Related topics

Basics

Basics on ModelDesk........ 7
Overview of the ModelDesk Access Library........ 11

References

GetManeuverState........ 41
InitMAPort (AutomationDesk Accessing Simulation Platforms 📖)
ResetVehicle........ 39
StartManeuver........ 35

StopManeuver..37
XIL API (Model Access) (AutomationDesk Accessing Simulation Platforms 🕮)

Reference Information

Automation Blocks

Introduction

The Signal-ModelDesk Access library is a custom library. It is write-protected to prevent modifications to its blocks.

Using ModelDesk Access library features in Python scripts

You can use functions and other definitions of the ModelDesk Access library in Python scripts after you imported the `audmodeldeskaccess` module to the currentname space.

Where to go from here

Information in this section

Application

Introduction

The Application folder in the ModelDesk Access library provides blocks to access the ModelDesk application.

Where to go from here

Information in this section

StartModelDesk

Graphical representation

Purpose

To start ModelDesk.

Description

This block creates an instance of ModelDesk. If a ModelDesk process is already running, the existing process is used. You can specify to open ModelDesk in visible or invisible mode.

Note

If ModelDesk is not available, for example, if it is not installed, AutomationDesk throws an exception.

Data objects

This automation block provides the following data object:

Name	In / Out	Data Type	Default Value	Description
Visible	In	String	"True"	Lets you specify the visible mode of the application: ▪ True ModelDesk starts with the user interface displayed. ▪ False ModelDesk starts in hidden mode.

Related topics

HowTos

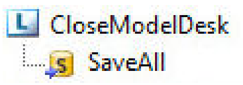
How to Build a Basic Sequence for Accessing ModelDesk.. 13

References

CloseModelDesk.. 28

CloseModelDesk

Graphical representation



Purpose

To close ModelDesk.

Description

This block exits ModelDesk. You can specify whether to save the modifications made during automated access. If ModelDesk is already closed, AutomationDesk starts the ModelDesk application in invisible mode to close it correctly afterwards.

This block also closes a ModelDesk instance, if it was opened manually.

Data objects

This automation block provides the following data object:

Name	In / Out	Data Type	Default Value	Description
SaveAll	In	String	"True"	Lets you specify whether to save your modifications: ▪ True Modifications are saved before closing ModelDesk. ▪ False Modifications are discarded without confirmation.

Related topics

HowTos

References

ProjectAndExperiment

Introduction

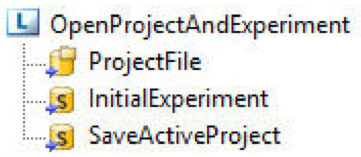
The ProjectAndExperiment folder in the ModelDesk Access library provides blocks to access ModelDesk's Project Navigator for managing projects and experiments.

Where to go from here

Information in this section

OpenProjectAndExperiment

Graphical representation



Purpose

To load a project and activate an experiment.

Description

This block loads the specified project. If the project contains several experiments, you can specify which experiment is to be activated. Because you can load only one project at the same time in ModelDesk, any already loaded project is closed if it differs from the specified one.You can specify whether to save the project before it is closed. An already loaded project is not saved and closed if it matches the specified one, i.e. the project paths are identical. The block then refers to this project without further actions.

If you have not used the **StartModelDesk** block beforehand, ModelDesk is automatically started in invisible mode.

If the specified project file is not available, AutomationDesk throws an exception.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
ProjectFile	In	File	" "	Lets you specify the ModelDesk project file (CDP) to be loaded.
InitialExperiment	In	String	" "	Lets you optionally specify the experiment to be activated.

Name	In / Out	Data Type	Default Value	Description
SaveActiveProject	In	String	"True"	Lets you specify whether to save an already loaded project if it differs from the specified one. ▪ True Modifications in the already loaded project are saved. ▪ False Modifications in the already loaded project are discarded.

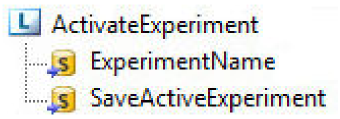
Related topics

HowTos

References

ActivateExperiment

Graphical representation



Purpose

To activate an experiment in the current ModelDesk project.

Description

This block is used to activate another experiment from the loaded ModelDesk project. You can specify whether to save the modifications in the currently active experiment before switching to the new active experiment.

If the specified experiment is not available, AutomationDesk throws an exception.

Because the block requires an already loaded project, you have to execute the OpenProjectAndExperiment block beforehand.

Data objects

This automation block provides the following data objects:

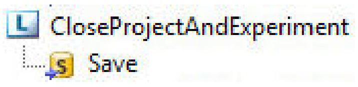
Name	In / Out	Data Type	Default Value	Description
ExperimentName	In	String	" "	Lets you specify the experiment to be activated.
SaveActiveExperiment	In	String	"True"	Lets you specify whether to save modifications in the currently active experiment before switching to the new one. ▪ True Modifications are saved. ▪ False Modifications are discarded.

Related topics

Basics

CloseProjectAndExperiment

Graphical representation



Purpose

To close a ModelDesk project.

Description

This block closes the active ModelDesk project. You can specify whether to save modifications in the project before closing it. If the project is already closed, the block executes with no action.

Data objects

This automation block provides the following data object:

Name	In / Out	Data Type	Default Value	Description
Save	In	String	"True"	Lets you specify whether to save modifications in the active experiment and project before closing it. ▪ True Modifications are saved.

Name	In / Out	Data Type	Default Value	Description
				▪ False Modifications are discarded.

Related topics

HowTos

References

Maneuver

Introduction

The **Maneuver** folder in the ModelDesk Access library provides blocks to activate and download a maneuver.

Note

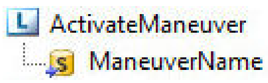
Up to and including ModelDesk 5.2, you can use the maneuver compatibility mode in ModelDesk to work with maneuvers created with ModelDesk 4.6 and earlier.
As of ModelDesk 5.3, the maneuver compatibility mode is discontinued. Maneuvers created with ModelDesk 4.6 and earlier are no longer supported. Executing ModelDesk Access blocks to activate and download these maneuvers leads to exceptions.

Where to go from here

Information in this section

ActivateManeuver

Graphical representation


ActivateManeuver
ManeuverName

Purpose

To activate a maneuver.

Description

This block activates the specified maneuver. If the maneuver is not available in the active experiment, AutomationDesk throws an exception.

The related project and experiment must be activated before you execute this block.

Data objects

This automation block provides the following data object:

Name	In / Out	Data Type	Default Value	Description
ManeuverName	In	String	" "	Lets you specify the name of the maneuver to be activated.

Related topics

HowTos

How to Build a Basic Sequence for Accessing ModelDesk.. 13

References

ActivateExperiment.. 30
DownloadActiveManeuver... 33

DownloadActiveManeuver

Graphical representation

DownloadActiveManeuver

Purpose

To download the active maneuver to Simulink, VEOS or the real-time hardware.

Description

This block downloads to the simulation platform the maneuver that you activated beforehand by using the **ActivateManeuver** block.

The simulation platform to which the maneuver is downloaded has to be configured in ModelDesk beforehand.

Data objects

None

Related topics

HowTos

How to Build a Basic Sequence for Accessing ModelDesk 13

References

ActivateManeuver 33

SaveActiveManeuver

Graphical representation

SaveActiveManeuver

Purpose

To save the settings of the active maneuver.

Description

This block saves the settings of the maneuver that you activated by using the ActivateManeuver on page 33 block.

Data objects

None

Related topics

HowTos

How to Build a Basic Sequence for Accessing ModelDesk 13

References

ActivateManeuver 33
DownloadActiveManeuver 33

ManeuverControl

Introduction

The ManeuverControl folder in the ModelDesk Access library provides blocks to control maneuver actions such as starting a maneuver.

Note

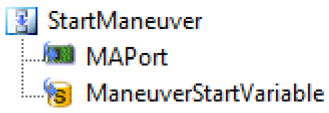
- The XIL API Convenience library must be loaded in AutomationDesk because the ModelDesk Access library contains dependencies to it.
- The automation blocks to control ModelDesk maneuvers are only applicable to maneuvers running the ASM Vehicle Dynamics model.

Where to go from here

Information in this section

StartManeuver

Graphical representation



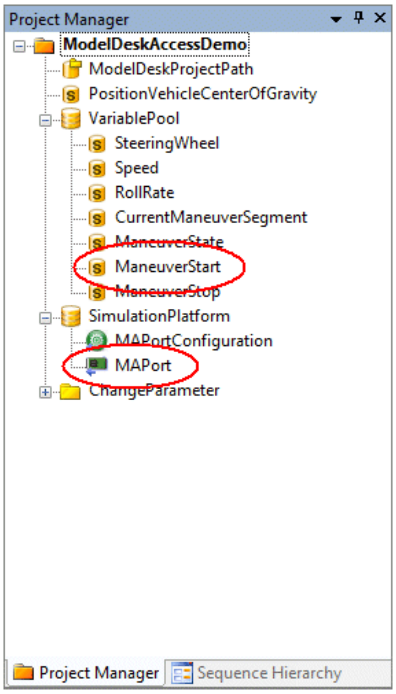
Purpose

To start a maneuver.

Description

This block starts the maneuver that you downloaded to the simulation platform by using the DownloadActiveManeuver block.

The block's data objects have preconfigured reference names. You have to add the referencing data objects to the project as shown below.



Note

The MAPort data object has to be initialized in the AutomationDesk sequence by using the InitMAPort block from the XIL API library.

Tip

The required project contents are also in the ModelDeskAccessDemo project in `<DocumentsFolder>\ModelDesk Access`.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
MAPort	In	MAPort	None	The data object references an MAPort data object of the XIL API library in the AutomationDesk project. The reference name is preconfigured with `SimulationPlatform.MAPort`.
ManeuverStartVariable	In	String	" "	The data object references a String data object in the AutomationDesk project. The reference name is preconfigured with `VariablePool.ManeuverStart`.

Name	In / Out	Data Type	Default Value	Description
				The referenced String data object in the project has to provide the path of the ManeuverStart parameter in the maneuver scheduler. For example, the parameter path for an SP model is: `()://Model Root/Environment/Maneuver/UserInterface/ PAR_Plant/ManeuverControl/ MANEUVER_START/MDLDCtrl_ManeuverStart` Note: For an MP model, the variable path has to be adapted accordingly.

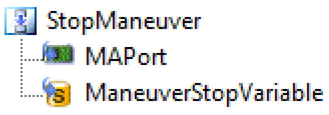
Related topics

HowTos

References

StopManeuver

Graphical representation



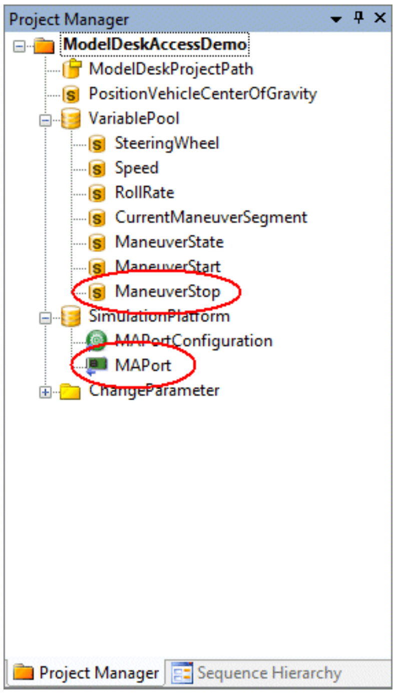
Purpose

To stop a running maneuver.

Description

This block stops a maneuver that you started by using the StartManeuver block.

The block's data objects have preconfigured reference names. You have to add the referencing data objects to the project as shown below.



Tip

The required project contents are also in the ModelDeskAccessDemo project in `<DocumentsFolder>\ModelDesk Access`.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
MAPort	In	MAPort	None	The data object references an MAPort data object of the XIL API library in the AutomationDesk project. The reference name is preconfigured with `SimulationPlatform.MAPort`.
ManeuverStopVariable	In	String	" "	The data object references a String data object in the AutomationDesk project. The reference name is preconfigured with `VariablePool.ManeuverStop`. The referenced String data object in the project has to provide the path of the ManeuverStop parameter in the maneuver scheduler. For example, the parameter path for an SP model is: `()://Model Root/Environment/Maneuver/UserInterface/ PAR_Plant/ManeuverControl/ MANEUVER_STOP/MDLDCtrl_ManeuverStop` Note: For an MP model, the variable path has to be adapted accordingly.

Related topics

HowTos

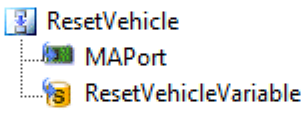
How to Control Maneuvers 19

References

StartManeuver 35

ResetVehicle

Graphical representation



ResetVehicle
MAPort
ResetVehicleVariable

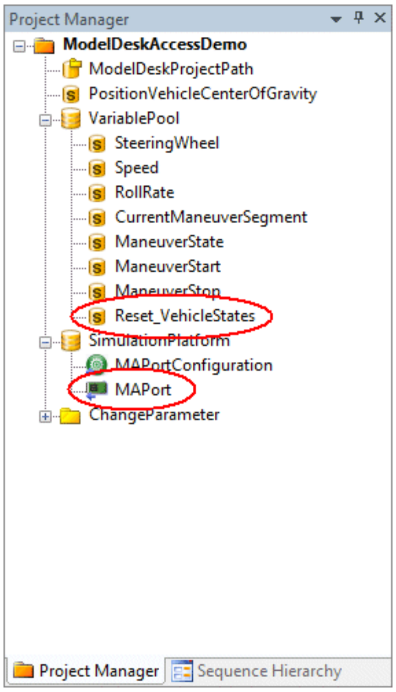
Purpose

To reset the vehicle to its initial state.

Description

This block resets the vehicle specified in the ModelDesk maneuver to its initial state.

The block's data objects have preconfigured reference names. You have to add the referencing data objects to the project as shown below.



Tip

The required project contents are also in the ModelDeskAccessDemo project in `<DocumentsFolder>\ModelDesk Access`.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
MAPort	In	MAPort	None	The data object references an MAPort data object of the XIL API library in the AutomationDesk project. The reference name is preconfigured with `SimulationPlatform.MAPort`.

ResetVehicleVariable	In	String	" "	The data object references a String data object in the AutomationDesk project. The reference name is preconfigured with `VariablePool.Reset_VehicleStates`.
				The referenced String data object in the project has to provide the path of the Reset_VehicleStates parameter in the maneuver scheduler. For example, the parameter path for an SP model is:
				`()://Model Root/Environment/Maneuver/UserInterface/`
				`PAR_Plant/ManeuverControl/`
				`RESET/MDLDCtrl_Reset`
				Note: For an MP model, the variable path has to be adapted accordingly.

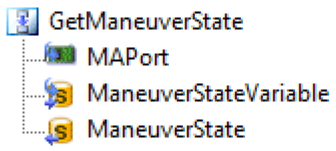
Related topics

HowTos

References

GetManeuverState

Graphical representation



GetManeuverState
MAPort
ManeuverStateVariable
ManeuverState

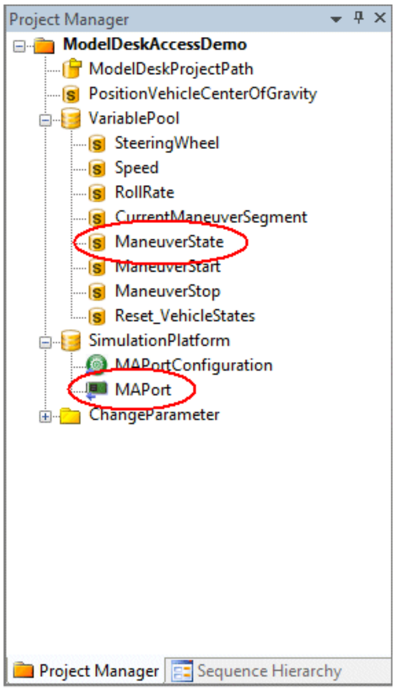
Purpose

To get the current state of a maneuver.

Description

This block reads the current state of the active maneuver.

The block's data objects have preconfigured reference names. You have to add the referencing data objects to the project as shown below.



Tip

The required project contents are also in the ModelDeskAccessDemo project in `<DocumentsFolder>\ModelDesk Access`.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
MAPort	In	MAPort	None	The data object references an MAPort data object of the XIL API library in the AutomationDesk project. The reference name is preconfigured with `SimulationPlatform.MAPort`.
ManeuverStateVariable	In	String	" "	The data object references a String data object in the AutomationDesk project. The reference name is preconfigured with `VariablePool.ManeuverState`. The referenced String data object in the project has to provide the path of the ManeuverState parameter in the maneuver scheduler. For example, the parameter path for an SP model is : `()://Model Root/Environment/Maneuver/UserInterface/DISP_Plant/ManeuverState[]/Out1`

Name	In / Out	Data Type	Default Value	Description
ManeuverState	Out	String	" "	Note: For an MP model, the variable path has to be adapted accordingly. Returns the value of the maneuver state variable as a string. The mapping is: ▪ 2: "Init" ▪ 3: "Run" ▪ 4: "Stopping" ▪ 5: "Wait" ▪ 6: "Init Manual" ▪ 7: "Manual"

Related topics

HowTos

References

Road

Introduction

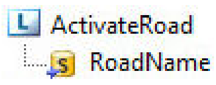
The **Road** folder in the ModelDesk Access library provides blocks to activate and download a road.

Where to go from here

Information in this section

ActivateRoad

Graphical representation


ActivateRoad
RoadName

Purpose

To activate a road.

Description

This block activates the road specified in the current ModelDesk experiment.

Data objects

This automation block provides the following data object:

Name	In / Out	Data Type	Default Value	Description
RoadName	In	String	" "	Lets you specify the name of the road to be activated.

Related topics

HowTos

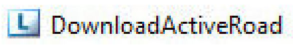
How to Build a Basic Sequence for Accessing ModelDesk.. 13

References

DownloadActiveRoad.. 44

DownloadActiveRoad

Graphical representation


DownloadActiveRoad

Purpose

To download the active road to the simulation platform.

Description

This block downloads the road that you activated by using the ActivateRoad block to Simulink, VEOS or the specified real-time hardware.

Data objects

None

Related topics

HowTos

References

SaveActiveRoad

Graphical representation

SaveActiveRoad

Purpose

To save the settings of the active road.

Description

This block saves the settings of the road that you activated by using the ActivateRoad on page 44 block.

Data objects

None

Related topics

HowTos

References

Traffic

Introduction

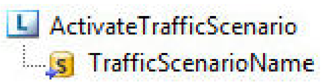
The Traffic folder in the ModelDesk Access library provides blocks to activate and download a traffic scenario.

Where to go from here

Information in this section

ActivateTrafficScenario

Graphical representation


ActivateTrafficScenario
TrafficScenarioName

Purpose

To activate a traffic scenario.

Description

This block activates the traffic scenario specified in the current ModelDesk experiment.

Data objects

This automation block provides the following data object:

Name	In / Out	Data Type	Default Value	Description
TrafficScenarioName	In	String	" "	Lets you specify the name of the traffic scenario to be activated.

Related topics

HowTos

How to Build a Basic Sequence for Accessing ModelDesk....13

References

DownloadActiveTrafficScenario....47

DownloadActiveTrafficScenario

Graphical representation	DownloadActiveTrafficScenario
Purpose	To download the active traffic scenario to the simulation platform.
Description	This block downloads the traffic scenario that you activated by using the ActivateTrafficScenario block to Simulink, VEOS or the specified real-time hardware.
Data objects	None
Related topics	HowTos How to Build a Basic Sequence for Accessing ModelDesk........ 13 References ActivateTrafficScenario........ 46

SaveTrafficScenario

Graphical representation	SaveActiveTrafficScenario
Purpose	To save the settings of the active traffic scenario.
Description	This block saves the settings of the traffic scenario that you activated by using the ActivateTrafficScenario on page 46 block.
Data objects	None

Related topics

HowTos

References

Parameter

Introduction

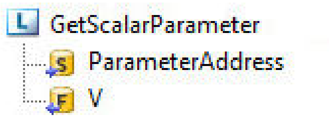
The **Parameter** folder in the ModelDesk Access library provides blocks to manage read values from parameters and to write values to parameters.

Where to go from here

Information in this section

GetScalarParameter

Graphical representation


GetScalarParameter
ParameterAddress
V

Purpose

To read a scalar parameter value.

Description

This block reads the parameter at the specified address and returns its value as a single float value. If the specified address does not provide a scalar value, an error message is displayed.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
ParameterAddress	In	String	" "	Lets you specify the address of the parameter to read from.
V	Out	Float	0.0	Returns the parameter value.

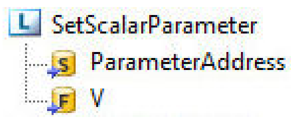
Related topics

HowTos

References

SetScalarParameter

Graphical representation



Purpose

To write a scalar parameter value.

Description

This block writes the specified scalar value to the specified parameter address. If the parameter is configured for a different data type, an error message is displayed.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
ParameterAddress	In	String	" "	Lets you specify the parameter address to write to.
V	In	Float	0.0	Lets you specify the parameter value you want to write.

Related topics

HowTos

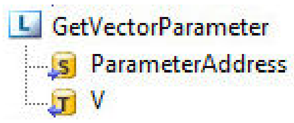
How to Work with Parameters........ 16

References

GetScalarParameter........ 49
SetGenericParameterValue........ 59

GetVectorParameter

Graphical representation



Purpose

To read a vector parameter value.

Description

This block reads the parameter at the specified address and returns its vector value as a tuple of float values. If the specified address does not provide a vector value, an error message is displayed.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
ParameterAddress	In	String	" "	Lets you specify the address of the parameter to read from.
V	Out	Tuple	()	Returns the values of a vector parameter.

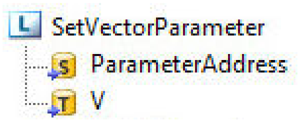
Related topics

HowTos

References

SetVectorParameter

Graphical representation



Purpose

To write a vector parameter value.

Description

This block writes the specified vector value as a tuple of float values to the specified parameter address. If the parameter is configured for a different data type, an error message is displayed.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
ParameterAddress	In	String	" "	Lets you specify the parameter address to write to.
V	In	Tuple	()	Lets you specify the parameter value you want to write.

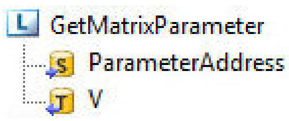
Related topics

HowTos

References

GetMatrixParameter

Graphical representation



Purpose

To read a matrix parameter value.

Description

This block reads the parameter at the specified address and returns its matrix value as a tuple of float values. Each row of the matrix is stored in a separate tuple. For example, a 3x3 matrix is stored at ((v_a1, v_a2, v_a3), (v_b1, v_b2, v_b3), (v_c1, v_c2, v_c3)), where a, b, c is the row index and 1, 2, 3 is the column index. If the specified address does not provide a matrix value, an error message is displayed.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
ParameterAddress	In	String	" "	Lets you specify the address of the parameter to read from.
V	Out	Tuple	()	Returns the values of a matrix parameter.

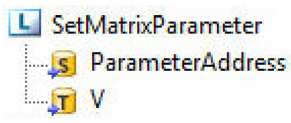
Related topics

HowTos

References

SetMatrixParameter

Graphical representation



Purpose

To write a matrix parameter value.

Description

This block writes the specified matrix value to the specified parameter address as a tuple of float values. You have to add a separate tuple for each row of the matrix. For example, a 3x3 matrix is defined by ((v_a1, v_a2, v_a3), (v_b1, v_b2, v_b3), (v_c1, v_c2, v_c3)), where a, b, c is the row index and 1, 2, 3 is the column index. If the parameter is configured for a different data type, an error message is displayed.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
ParameterAddress	In	String	" "	Lets you specify the parameter address to write to.
V	In	Tuple	()	Lets you specify the parameter value you want to write.

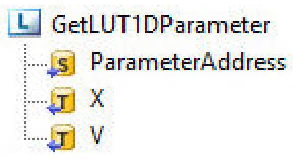
Related topics

HowTos

References

GetLUT1DParameter

Graphical representation



Purpose

To read a one-dimensional look-up table (LUT1D) parameter.

Description

This block reads the parameter at the specified address and returns its one-dimensional look-up table (LUT1D) value as two tuples of float values. The first tuple X provides the values of the x-axis. The second tuple V provides the values at the related x position. If the specified address does not provide a LUT1D value, an error message is displayed.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
ParameterAddress	In	String	" "	Lets you specify the address of the parameter to read from.
X	Out	Tuple	()	Returns the values of the x-axis of the specified LUT1D parameter.
V	Out	Tuple	()	Returns the values of the specified LUT1D parameter.

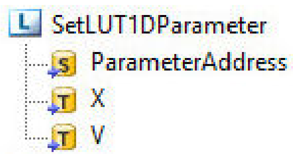
Related topics

HowTos

References

SetLUT1DParameter

Graphical representation



Purpose

To write a one-dimensional look-up table (LUT1D) parameter value.

Description

This block writes the specified one-dimensional look-up table (LUT1D) values to the specified parameter address by two tuples of float values. The first tuple X provides the values of the x-axis. The second tuple V provides the values at the related x position. If the parameter is configured for a different data type, an error message is displayed.

Note

Both tuples must have the same dimension.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
ParameterAddress	In	String	" "	Lets you specify the parameter address to write to.
X	In	Tuple	()	Lets you specify the parameter values for the x-axis you want to write.
V	In	Tuple	()	Lets you specify the parameter values you want to write.

Related topics

HowTos

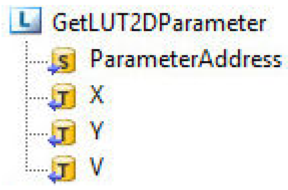
How to Work with Parameters.. 16

References

GetLUT1DParameter.. 54
SetGenericParameterValue.. 59

GetLUT2DParameter

Graphical representation



Purpose

To read a two-dimensional look-up table (LUT2D) parameter.

Description

This block reads the parameter at the specified address and returns its two-dimensional look-up table (LUT2D) value as three tuples of float values. The first tuple X provides the values of the x-axis. The second tuple Y provides the values of the y-axis. The third tuple V provides the values at the related xy position. If the specified address does not provide a LUT2D value, an error message is displayed.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
ParameterAddress	In	String	" "	Lets you specify the address of the parameter to read from.
X	Out	Tuple	()	Returns the values of the x-axis of the specified LUT2D parameter.
Y	Out	Tuple	()	Returns the values of the y-axis of the specified LUT2D parameter.
V	Out	Tuple	()	Returns the values of the specified LUT2D parameter.

Related topics

HowTos

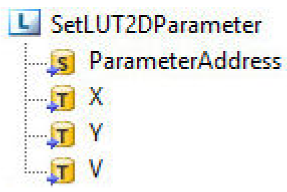
How to Work with Parameters.......... 16

References

GetGenericParameterValue.......... 58
SetLUT2DParameter.......... 57

SetLUT2DParameter

Graphical representation



Purpose

To write a two-dimensional look-up table (LUT2D) parameter value.

Description

This block writes the specified two-dimensional look-up table (LUT2D) values to the specified parameter address by three tuples of float values. The first tuple X provides the values of the x-axis. The second tuple Y provides the values of the y-axis. The third tuple V provides the values at the related xy position. If the parameter is configured for a different data type, an error message is displayed.

Note

All the tuples must have the same dimension.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
ParameterAddress	In	String	" "	Lets you specify the parameter address to write to.
X	In	Tuple	()	Lets you specify the parameter values for the x-axis you want to write.
Y	In	Tuple	()	Lets you specify the parameter values for the y-axis you want to write.
V	In	Tuple	()	Lets you specify the parameter values you want to write.

Related topics

HowTos

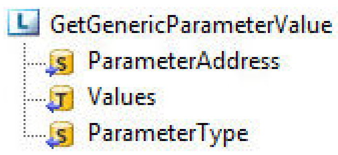
How to Work with Parameters........ 16

References

GetLUT2DParameter........ 56
SetGenericParameterValue........ 59

GetGenericParameterValue

Graphical representation


GetGenericParameterValue
ParameterAddress
Values
ParameterType

Purpose

To read a parameter of any type.

Description

This block reads the parameter at the specified address and returns the values as a tuple of float values. It also returns the parameter type that is required to implement postprocessing for reading the returned tuple correctly. A multi-dimensional parameter is returned as nested tuples. If the specified address does not provide a parameter, an error message is displayed.

Mapping example The following table shows how a specific parameter type is stored.

Parameter Type	Representation
Scalar	v
Vector	(v1, v2, v3, …)
Matrix	((v_a1, v_a2, …), (v_b1, v_b2, …), (v_c1, v_c2, …)) (where a, b, c is the row index and 1, 2 is the column index)
LUT1D[1)]	((v1, v2, v3, …), (x1, x2, x3, …))
LUT2D[2)]	((v1, v2, v3, …), (x1, x2, x3, …), (y1, y2, y3, …))

[1)] One-dimensional look-up-table
[2)] Two-dimensional look-up-table

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
ParameterAddress	In	String	" "	Lets you specify the address of the parameter to read from.
Values	Out	Tuple	()	Returns the values of the specified parameter.
ParameterType	Out	String	" "	Returns the type of the specified parameter. The following names are returned: ▪ Scalar ▪ Vector ▪ Matrix ▪ LUT1D ▪ LUT2D

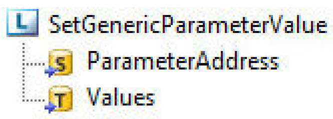
Related topics

HowTos

References

SetGenericParameterValue

Graphical representation



Purpose

To write a parameter value of any type.

Description

This block writes the specified tuple values to the specified parameter address. The tuple structure must follow the rules described in the description of the **GetGenericParameterValue** block. If the specified values do not match the type of the specified parameter, an error message is displayed.

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
ParameterAddress	In	String	" "	Lets you specify the parameter address to write to.
Values	In	Tuple	()	Lets you specify the parameter values you want to write.

Related topics

HowTos

References

ParameterSet

Introduction

The **ParameterSet** folder in the ModelDesk Access library provides blocks to manage parameter sets.

Where to go from here

Information in this section

ActivateParameterSet

Graphical representation

Purpose

To activate a parameter set.

Description

This block is used to activate a parameter set from the loaded ModelDesk project. You can specify whether to save the modifications in a currently active parameter set before switching to the new one.

If the specified parameter set name is not available, AutomationDesk throws an exception.

Data objects

This automation block provides the following data objects:

Name	**In / Out**	**Data Type**	**Default Value**	**Description**
ParameterSetName	In	String	" "	Lets you specify the name of the parameter set to be activated.
SaveActiveParameterSet	In	String	"True"	Lets you specify whether to save modifications in the currently active parameter set before switching to the new one. ▪ True Modifications will be saved. ▪ False Modifications will be discarded.

Related topics

HowTos

References

SaveActiveParameterSet

Graphical representation

SaveActiveParameterSet

Purpose

To save the settings in the active parameter set.

Description

This block saves the settings in the parameter set that you activated by using the ActivateParameterSet block.

Data objects

None

Related topics

HowTos

How to Work with Parameters 16

References

ActivateParameterSet 61
DownloadActiveParameterSet 62

DownloadActiveParameterSet

Graphical representation

DownloadActiveParameterSet

Purpose

To download the active parameter set to the simulation platform.

Description

This block downloads the parameter set that you activated by using the ActivateParameterSet block to Simulink or the specified real-time hardware.

Data objects

None

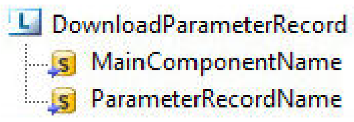
Related topics

HowTos

References

DownloadParameterRecord

Graphical representation



DownloadParameterRecord
MainComponentName
ParameterRecordName

Purpose

To download a subset of the active parameter set to the simulation platform.

Description

This block downloads a subset of the active parameter set to Simulink or the specified real-time hardware. The parameter subset is specified by its main component name, for example, *VehicleDynamics* or *Drivetrain*, and its parameter record name. A parameter record stores the values of one or more parameter pages of the specified main component in ModelDesk.

If any of the specified names cannot be found in the ModelDesk project, AutomationDesk throws an exception.

Tip

You can get the corresponding parameter record name of a parameter page by choosing **Help** in its context menu in ModelDesk. dSPACE Help opens with the reference information of the according parameter page in ASM Parameters (ModelDesk Parameterizing 🕮).

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
MainComponentName	In	String	" "	Lets you enter the name of the main component to specify the subset of the parameter set to be downloaded.

Name	In / Out	Data Type	Default Value	Description
ParameterRecordName	In	String	" "	Lets you enter the name of the parameter record to specify the subset of the parameter set to be downloaded.

Related topics

HowTos

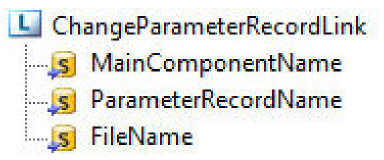
How to Work with Parameters.. 16

References

ChangeParameterRecordLink.. 64

ChangeParameterRecordLink

Graphical representation



ChangeParameterRecordLink
MainComponentName
ParameterRecordName
FileName

Purpose

To link a parameter file saved in the ModelDesk project pool to the specified parameter record.

Description

This block makes parameters that are specified in a parameter file available in the parameter record. The file must be available in the **Pool** of the active ModelDesk project.

If any of the specified names is not available, AutomationDesk throws an exception.

Tip

You can get the corresponding parameter record name of a parameter page by choosing **Help** in its context menu in ModelDesk. dSPACE Help opens with the reference information of the according parameter page in ASM Parameters (ModelDesk Parameterizing 🕮).

Data objects

This automation block provides the following data objects:

Name	In / Out	Data Type	Default Value	Description
MainComponentName	In	String	" "	Lets you specify the name of the main component the parameters from the file are to be added to.
ParameterRecordName	In	String	" "	Lets you specify the name of the parameter record the parameters from the file are to be added to.
FileName	In	String	" "	Lets you specify the name of the parameter file to be linked.

Related topics

HowTos

How to Work with Parameters.. 16

References

DownloadParameterRecord.. 63

Automation

Basics on Automating the Access to ModelDesk

Introduction	AutomationDesk provides a COM-based API to automate the handling of AutomationDesk.
Related information	The AutomationDesk COM API provides no specific objects for accessing ModelDesk. You can only use the basic automation features, such as executing a project via script. For information on the available objects with their properties and methods, refer to Basic Interface (AutomationDesk Automation 📖). For basic information and instructions, refer to Basics and Instructions on page 7.

Limitations

Limitations When Using the ModelDesk Access Library

Using the blocks of the ManeuverControl folder	The automation blocks to control ModelDesk maneuvers are only applicable to maneuvers running the ASM Vehicle Dynamics model.
Working with maneuvers created with ModelDesk 4.6 and earlier	Up to and including ModelDesk 5.2, you can use the maneuver compatibility mode in ModelDesk to work with maneuvers created with ModelDesk 4.6 and earlier. As of ModelDesk 5.3, the maneuver compatibility mode is discontinued. Maneuvers created with ModelDesk 4.6 and earlier are no longer supported. Executing ModelDesk Access blocks to activate and download these maneuvers leads to exceptions.