

ASM

# User Guide

For Automotive Simulation Models 9.7

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:	<a href="mailto:info@dspace.de">info@dspace.de</a>
Web:	<a href="http://www.dspace.com">http://www.dspace.com</a>

## 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](mailto:support@dspace.de)

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

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

## Software Updates and Patches

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

## Important Notice

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

© 2006 - 2021 by:  
dSPACE GmbH  
Rathenaustraße 26  
33102 Paderborn  
Germany

This publication and the contents hereof are subject to change without notice.

AUTERA, ConfigurationDesk, ControlDesk, MicroAutoBox, MicroLabBox, SCALEXIO, SIMPHERA, SYNECT, SystemDesk, TargetLink and VEOS are registered trademarks of dSPACE GmbH in the United States or other countries, or both. Other brand names or product names are trademarks or registered trademarks of their respective companies or organizations.

# Contents

About This Document	11
Introduction	13
Introduction to Automotive Simulation Models.....	13
ASM Model Structure	15
Modeling Philosophy of the ASM.....	16
Introduction to the ASM Model Structure.....	16
ASM Module.....	17
Plant Subsystem.....	20
Control Subsystem.....	22
ASM Submodule.....	24
ASM Task Submodule.....	25
Domain Control.....	25
ASM Model Tools.....	27
MotionDesk Interface.....	29
User Interface.....	30
ASMSignalBus.....	31
Communication Signals.....	33
Model Initialization	35
Overview of Model Initialization.....	35
Model Initialization with go.m File.....	36
Handling of Main and Variant-Depending Initialization Files.....	38
General Information	41
Folder Structure.....	42
Basics on Folder Structure.....	42
ASM Installation Folder.....	43
ASM Project Folder.....	43
Variable Structure.....	47
Basics on Variable Structure.....	47
General Structure of Variables.....	48
Scalar, Vector, Matrix.....	49

1-D Look-up Table.....	49
2-D Look-up Table.....	50
Model Modifications.....	52
Model Modifications.....	52
Multi-Instance.....	53
Multi-Instance.....	53
ASM Blocks Not Supporting the Multi-Instance Feature.....	55
Dimension Parameters.....	57
Dimension Parameters.....	57
Operator Version.....	61
Operator Version.....	61
Numerical Aspects of ASMs.....	63
Basics on Numerical Aspects.....	63
Local Subsystem Oversampling.....	63
Semi-Implicit Euler Stabilization.....	65
<b>Migrating ASM Models</b>	<b>67</b>
Basics on Model Migration.....	68
Former Versions of ASM Blocks.....	69
Cases Which Require Migration.....	71
Steps Performed During a Migration.....	72
Migration Variants.....	73
Project Consistency.....	75
How to Migrate Blocks of Custom Component Libraries.....	75
<b>Tools</b>	<b>77</b>
ASM Engine Testbench.....	78
Basics of ASM Engine Testbench.....	78
Basic Concepts of ASM Engine Testbench.....	78
User interface of ASM Engine Testbench.....	79
Execute Page.....	79
General Settings Page.....	82
Stimulus Signals Page.....	83
Plots Page.....	85
Online Settings Page.....	87
Working with ASM Engine Testbench.....	90
How to Load an Experiment.....	90

How to View Simulation Results.....	91
How to Test an Engine Parameterization.....	91
How to Setup a New Test Bench Project.....	92
ASM RoadConverter.....	93
Introduction to ASM RoadConverter.....	93
Basics on ASM RoadConverter.....	93
Functional Description.....	94
Folder Structure of ASM RoadConverter.....	95
Data Formats.....	95
User interface of ASM RoadConverter.....	97
User Interface of ASM RoadConverter.....	97
Working with ASM RoadConverter.....	100
How to Start ASM RoadConverter.....	101
How to Manage Projects with ASM RoadConverter.....	101
How to Run Existing RoadConverter Projects.....	102
How to Import a Road in ModelDesk.....	102
How to Create a New Road Conversion Project.....	104
How to Convert Roads from ADAS RP.....	105
Tips & Tricks.....	106
ASM Controller Adaption.....	107
Introduction to ASM Controller Adaption.....	107
Basics on ASM Controller Adaption.....	107
Plant Types.....	109
User interface of ASM Controller Adaption.....	111
Working with ASM Controller Adaption.....	114
Tuning Methods.....	119
Overview of Tuning Methods.....	119
Amplitude Optimum.....	119
Symmetrical Optimum.....	122
Ziegler-Nichols Method.....	125
Adams2ASM Converter.....	128
Basics on the Adams2ASM Converter.....	128
Parameters in Adams.....	130
How to Use the Adams2ASM Converter.....	135
ASM LabellInterface.....	137
Basics of ASM LabellInterface Library.....	137
Basics on the ASM LabellInterface.....	137
How to Open the ASM LabellInterface Library.....	138

Bus To Vector Block.....	139
Description of Bus2Vector Block.....	140
Options Page.....	141
Special Options Page.....	142
Signal Page.....	146
Datatypes & Dimensions Page.....	148
Change Labels Block.....	151
Description of ChangeLabels Block.....	151
ASM Utils.....	153
Basics on the ASM Utils Library.....	153
How to Open the ASM Utils Library.....	153
For Iterator Subsystem.....	155
Forward Euler.....	155
For Iterator Delay.....	156
Transfer Functions.....	157
First Order Dynamics.....	157
Second Order Dynamics.....	158
Memory With Reset.....	159
ModelDesk Interface.....	160
Maneuver and Reset Control.....	160
Overview of the Maneuver and Reset Control.....	161
Maneuver Start.....	162
Maneuver Stop.....	163
Maneuver State.....	164
Reset.....	164
Multi-Instance Blocks.....	165
Multi-Instance Overview.....	165
ModelDesk Parameter Group.....	166
ModelDesk Parameter Group Overview.....	166
ModelDesk Parameter Group Block.....	167
ModelDesk Write Access.....	167
ModelDesk Main Component Block.....	168
Unit Conversion.....	169
Unit Conversion.....	169
Matrix Operations.....	170
Unit Vector.....	171
Unit Vectors.....	171

Transpose.....	172
Skew Symmetric Matrix.....	173
Rotation Matrix Angles.....	173
Rotation Matrix Angles Small.....	174
Rotation Matrix to Angles.....	174
Transform GammaAlphaBeta.....	175
Cross Product 3x1.....	176
Cross Product 6x1.....	176
Dot Product.....	177
Calc Caster Left.....	177
Coordinate Transformation B to A.....	178
Inverse Coordinate Transformation A to B.....	179
 Miscellaneous.....	180
ASM Function Call Generator.....	181
Compare Value.....	182
Test Cycle.....	184
Engine OP Selector.....	185
Flush Denormals to Zero.....	185
Select Maneuver.....	186
Variable Delay.....	187
 Extended Look-Up Table.....	188
1-D Look-Up Table.....	188
2-D Look-Up Table.....	189
Shift Look-Up Table.....	190
1-D Reciprocal Integral Lookup.....	191
 Replace Scopes.....	192
Scope Handling User Interface.....	192
 Open Experiment.....	193
Open Experiment.....	193
 Operator.....	193
Activate Developer Version.....	194
Activate Operator Version.....	194
 Block Replacement.....	195
Replace Tire TMEasy.....	195
Replace Tire MagicFormula.....	196
 Code Generation.....	196
Generate VEOS.....	196
 Random Numbers.....	197
Random Numbers with Normal Distribution.....	197

<b>Useful Functions</b>	<b>199</b>
asm_eng_drivingcycles.....	200
asm_eng_drivingcycles.....	200
asm_multiinstance.....	201
asm_multiinstance.....	201
asm_plot_table.....	203
asm_plot_table.....	203
asm_plot_geosusp.....	204
asm_plot_geosusp.....	204
asm_tablegenerator.....	205
2D_Table Method.....	205
Basics of 2D_Table Method.....	206
Function Call to the 2D_Table Method.....	206
Execution Order of the 2D_Table Method.....	209
Optional Parameters of the 2D_Table Method for Extrapolation.....	211
Optional Parameters of the 2D_Table Method for Plotting.....	214
1D_PsiTable Method.....	215
Basics of 1D_PsiTable Method.....	216
Function Call of 1D_PsiTable Method.....	216
asm_traffic_mdl_update_num_fellows.....	219
asm_traffic_mdl_update_num_fellows.....	219
CreateIndependentParameterxml.py.....	220
CreateIndependentParameterxml.py.....	220
CreateGeoOptTrajectory.py.....	223
CreateGeoOptTrajectory.py.....	223
<b>ModelDesk Processing</b>	<b>227</b>
asm_proc Function.....	228
Description of the asm_proc Function.....	229
calc Method.....	230
check_parameter Method.....	230
get_paramchilds Method.....	231
get_parameter Method.....	232
set_parameter Method.....	234
set_measurementnumber Method.....	235
check_measurement Method.....	236

get_measurement Method.....	237
set_measurement Method.....	238
set_round Method.....	238
write2log Method.....	239
get_plotconfig Method.....	240
Parameter Functions.....	242
Example of Calculating a Parameter in a Function.....	242
ImportParamfromWorkspace.m Function.....	246
Files for ModelDesk Processing.....	247
General Settings File.....	247
Additional Functions.....	250
Measurement Functions.....	252
Initialization Files.....	253
<b>ModelDesk Utility Library</b>	<b>257</b>
Introduction to the ModelDesk Utility Library.....	258
Introduction to the ModelDesk Utility Library.....	258
SignallInterface.....	259
ASMSignallInterface.....	259
ASMSignallInterface Manager.....	261
Custom Component.....	263
ModelDesk Custom Component.....	263
Custom Component Migration.....	264
<b>Tutorial</b>	<b>265</b>
ASM Tutorial Videos.....	266
How to Open an ASM Library.....	266
How to Start with an ASM Demo Model via MATLAB.....	268
How to Start a Simulink Simulation.....	270
How to Prepare a Model for Real-Time Simulation.....	270
How to Generate Code for an ASM.....	272
Specific Tutorials.....	273
<b>Troubleshooting</b>	<b>275</b>
Troubleshooting with Rising Turnaround Times at Steady Operating Points.....	276
Troubleshooting in MATLAB/Simulink.....	277
Troubleshooting after ControlDesk Project Migration.....	279

Troubleshooting with Measurement Function Definitions in the General Settings.....	279
Troubleshooting with Parameters in ModelDesk Processing After Migrating ModelDesk Projects.....	284
Troubleshooting with Dimension Parameters Error Messages.....	285
Troubleshooting for Task Overruns with a MotionDesk/Animation Interface on SCALEXIO Platforms.....	287
Troubleshooting for Task Overruns with a MotionDesk/Animation Interface on PHS Bus-based Platforms.....	289
<b>Limitations</b>	<b>293</b>
Limitations of All ASM Blocksets.....	293
<b>New Features/Migration History of the ASM Blocksets</b>	<b>295</b>
Changes to all ASM Products.....	295
History of the ASM Utils Library.....	300
<b>Appendix</b>	<b>303</b>
Bibliography.....	303
<b>Index</b>	<b>305</b>

# About This Document

## Content

This guide introduces you to the features provided by the Automotive Simulation Models (ASM). It provides information on the model structure, the migration process, useful functions and tools, and gives an overview of working with ASM.

## Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
 <b>DANGER</b>	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
 <b>WARNING</b>	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
 <b>CAUTION</b>	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
 <b>NOTICE</b>	Indicates a hazard that, if not avoided, could result in property damage.
 <b>Note</b>	Indicates important information that you should take into account to avoid malfunctions.
 <b>Tip</b>	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.
 (book)	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](http://www.dspace.com/go/help).

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

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

# Introduction

## Introduction to Automotive Simulation Models

### Introduction

Automotive Simulation Models (ASM) are open Simulink® models used as environment models in simulation. You can use the models for testing electronic control units or for the early validation of controller algorithms during the design phase.

You can use ASM for real-time simulation and for offline simulation in Simulink and on VEOS.

### Model structure

The ASM include models for:

- Vehicle dynamics
- Engine
- Electric components
- Environment

You can combine the models and easily modify them by replacing blocks and specifying different parameters.

**Demo models** The ASM installation includes demo models, which you can easily access via the ASM library in Simulink. You can use the demo models as the basis for your own model.

### Supported platforms

ASM supports the following simulation platforms:

- For real-time simulation:
  - SCALEXIO
  - DS1006 Processor Board
  - DS1007 PPC Processor Board (not supported by all ASM models)
- For offline simulation:
  - Simulink
  - VEOS (on Linux and on Windows) on the same PC and on a remote PC

---

**Parameterization in ModelDesk**

You can parameterize the ASM models via the graphical user interface in ModelDesk. In ModelDesk, you can navigate through the model via navigation pages. For more information on ModelDesk and parameterizing the ASM, refer to [Features of ModelDesk \(ModelDesk Basics\)](#) and [Using Automotive Simulation Models \(ModelDesk Parameterizing\)](#).

---

**User documentation**

For most ASM blocksets, the following documents are available:

- A *Reference* containing descriptions of all blocks included in the blockset.
  - A *Model Description* describing how to work with the model.
- 

**Related topics**

**Basics**

[Features of ModelDesk \(ModelDesk Basics\)](#)  
[Using Automotive Simulation Models \(ModelDesk Parameterizing\)](#)

# ASM Model Structure

## Where to go from here

## Information in this section

<a href="#">Modeling Philosophy of the ASM</a> .....	16
The model structure of the ASM has several advantages.	
<a href="#">Introduction to the ASM Model Structure</a> .....	16
All ASM demo models have the same structure.	
<a href="#">ASM Module</a> .....	17
Every ASM demo model contains multiple ASM modules. All ASM modules have the same structure.	
<a href="#">Plant Subsystem</a> .....	20
The Plant subsystem contains the plant model of the ASM module.	
<a href="#">Control Subsystem</a> .....	22
The Control subsystem contains the controller model of the ASM module.	
<a href="#">ASM Submodule</a> .....	24
An ASM module can contain further ASM submodules.	
<a href="#">ASM Task Submodule</a> .....	25
An ASM task submodule can be oversampled in simulation.	
<a href="#">Domain Control</a> .....	25
It is also possible that several ASM modules are grouped in one block of the ASM model. This is done in the DomainControl block.	
<a href="#">ASM Model Tools</a> .....	27
The ASM Model Tools subsystem contains different common functionalities for working with the ASM models.	
<a href="#">MotionDesk Interface</a> .....	29
This subsystem calculates the motion data and transfers it to MotionDesk.	

**User Interface.....** 30

The UserInterface subsystem is used as a central access point inside the model.

**ASMSignalBus.....** 31

The ASMSignalBus is used inside each ASM module. It contains signals of ASM components.

**Communication Signals.....** 33

For the communication between and inside the ASM modules, different types of signals are used.

## Modeling Philosophy of the ASM

### Introduction

The model structure of the ASM has several advantages. It is designed to meet the requirements of today's model-based development.

### Advantages

The model structure of the ASM provides the following advantages:

**Exchanging model parts** The ASM is divided into different *ASM modules*, all of which have the same model structure and the same communication interfaces. This lets you easily exchange parts of the model and integrate submodels with different characteristics into an overall model.

The uniform structure also allows for distributed model development for the different parts of the model.

**Different model complexities** The modularity of the ASM modules lets you use model parts with different degrees of detail.

**Oversampling** You can set the sampling rate of an ASM module independently of the global step size.

**Real-time code generation** You can also generate real-time code for only parts of the model.

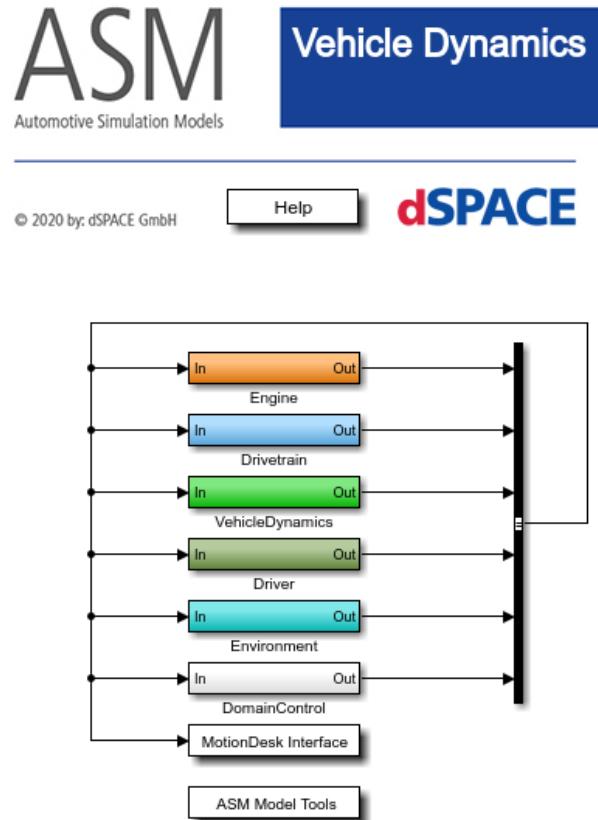
**User interface access** User interface access is possible directly in each ASM module.

## Introduction to the ASM Model Structure

### Overview

The ASM are shipped with various demo models. All ASM demo models have a similar structure.

When you open an ASM demo model in MATLAB/Simulink, it looks something like this:



The demo model contains the following components:

- ASM modules that are autonomous and constitute different parts of an ASM, such as the Engine module and the Environment module. Refer to [ASM Module](#) on page 17.
- ASM Model Tools for access to different common functionalities in ASM demo models . Refer to [ASM Model Tools](#) on page 27.
- (optionally) A MotionDesk Interface for animation in MotioDesk. Refer to [MotionDesk Interface](#) on page 29.

## ASM Module

### Introduction

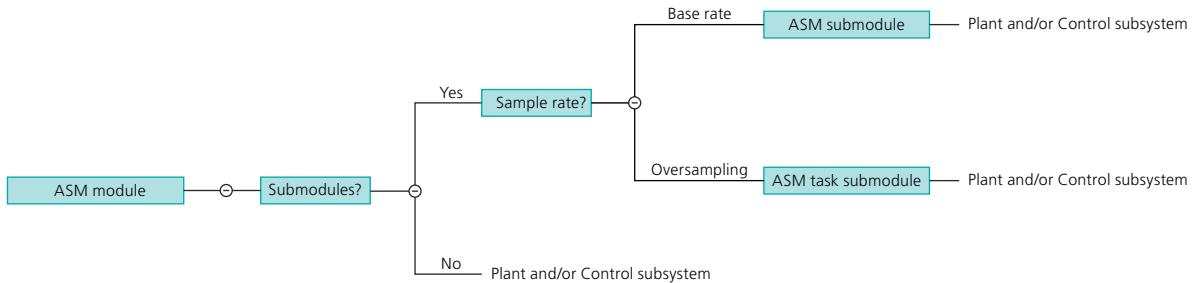
Every ASM demo model contains multiple ASM modules. All ASM modules have the same structure.

An ASM module is structured as a classical control loop with plant and controller.

There are two basic ASM module structures:

- ASM module without submodules
- ASM module with submodule(s)

The following illustration shows the possible structures of ASM modules:



## Signals

For the communication between and inside ASM modules, different types of signals are used. Refer to [Communication Signals](#) on page 33.

## Contained modules

Typically, an ASM demo model contains ASM modules of the following types:

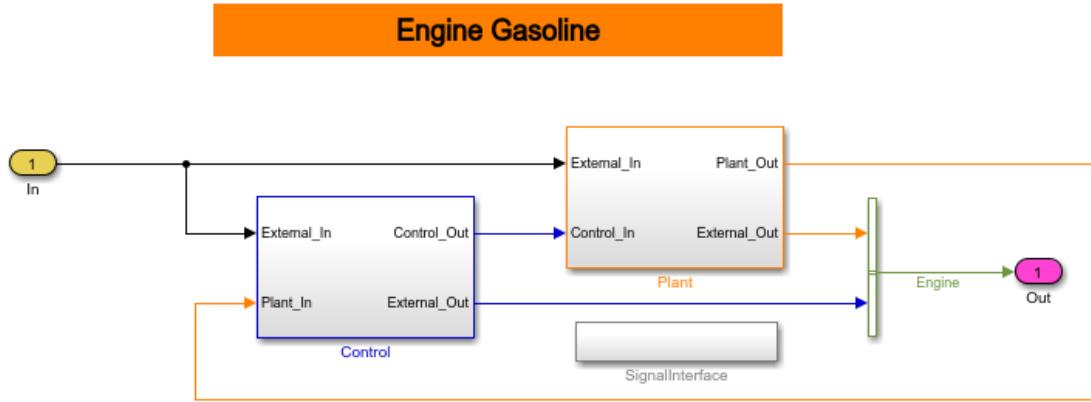
- Engine
- Drivetrain
- Vehicle Dynamics
- Electrical System
- Environment

Each ASM module contains models of different complexity. For example, the Engine module in the Vehicle Dynamics demo model contains a less complex Engine model than the Engine module in the Gasoline Engine demo model. Due to the standardized structure of the ASM modules, you can easily exchange it for a more or less complex model, depending on your requirements.

## ASM module without submodules

The module contains:

- A Plant block and/or a Control block.
- A SignalInterface block.



**Plant block** The Plant block contains the plant model of the ASM module. Refer to [Plant Subsystem](#) on page 20.

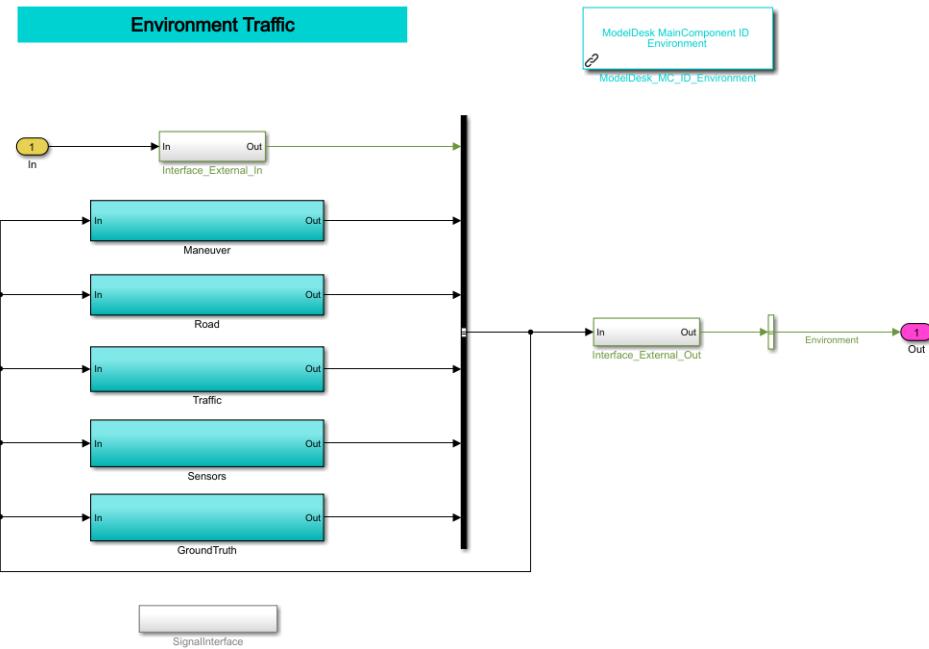
**Control block** The Control block contains the controller model of the ASM module. Refer to [Control Subsystem](#) on page 22.

**SignalInterface** The SignalInterface block contains the interface for communication with the ModelDesk plotting feature.

#### ASM module with submodules

The module contains:

- ASM submodule(s) and/or ASM task submodule(s).
- External\_In and External\_Out interfaces.
- A SignalInterface block.



**ASM submodule** An ASM submodule contains different functional subsystems that are not ASM modules on their own.

Refer to [ASM Submodule](#) on page 24.

**ASM task submodule** An ASM task submodule is a submodule that you can oversample in simulation.

Refer to [ASM Task Submodule](#) on page 25.

**Interfaces** Every ASM submodule contains External\_In and External\_Out interfaces for communication with the other ASM modules in the model.

**SignalInterface** The SignalInterface block contains the interface for communication with the ModelDesk plotting feature.

## Plant Subsystem

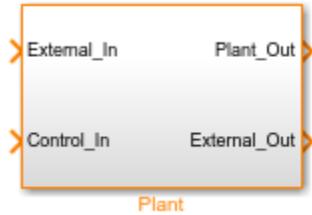
### Introduction

The Plant subsystem contains the plant model of the ASM module. It also contains the functionality to switch between a real component or a plant model.

All Plant subsystems are structured in a similar way.

**Plant block**

The Plant block looks like this:



**Imports** The block has the following imports:

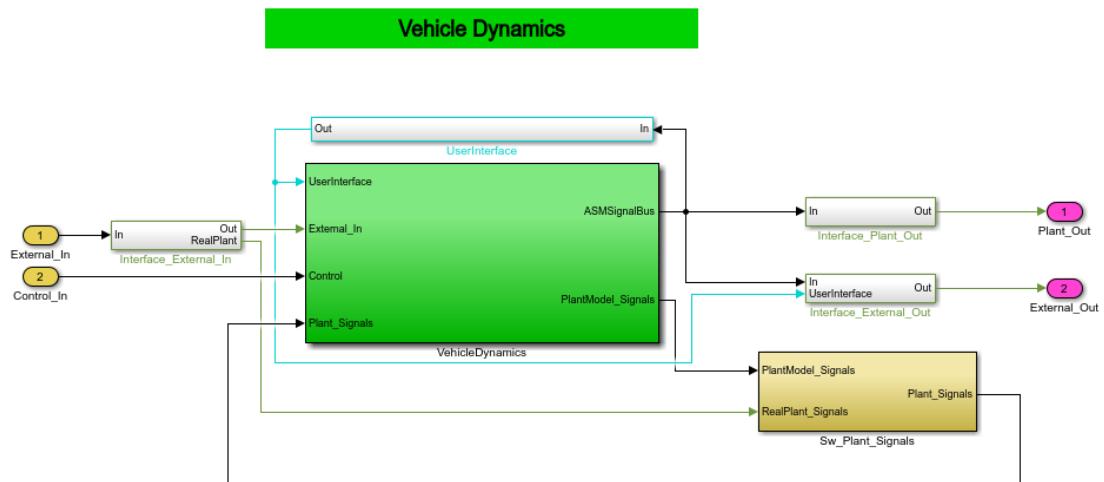
Name	Unit	Description
Control_In	[]	To connect signals from the Control block.
External_In	[]	To connect signals from other ASM modules.

**Outports** The block has the following outports:

Name	Unit	Description
External_Out	[]	Provides signals to other ASM modules.
Plant_Out	[]	Provides signals to the Control block.

**Structure**

The following illustration shows an example of the structure inside the Plant block:



**Interface\_External\_In** This interface provides signals from external ASM modules and signals from external (real) plant components (via I/O).

**Interface\_Plant\_Out** This interface provides signals for the Control subsystem of the same ASM module.

**Interface\_External\_Out** This interface provides plant signals for other ASM modules.

**<Model> block** The model block contains the functional part of the plant model.

**Sw\_Plant\_Signals** (optional) The switch lets you select between signals from a real plant component and the plant model.

**UserInterface** This interface is a central access point for different functions and settings inside the model. Refer to [User Interface](#) on page 30.

## Control Subsystem

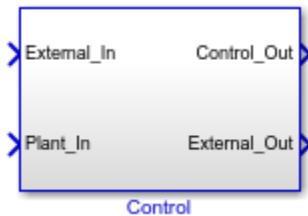
### Introduction

The Control subsystem contains the controller model of the ASM module. It also contains the functionality to switch between a real ECU or a control model.

All Control subsystems are structured in a similar way.

### Control block

The Control block looks like this:



**Imports** The block has the following imports:

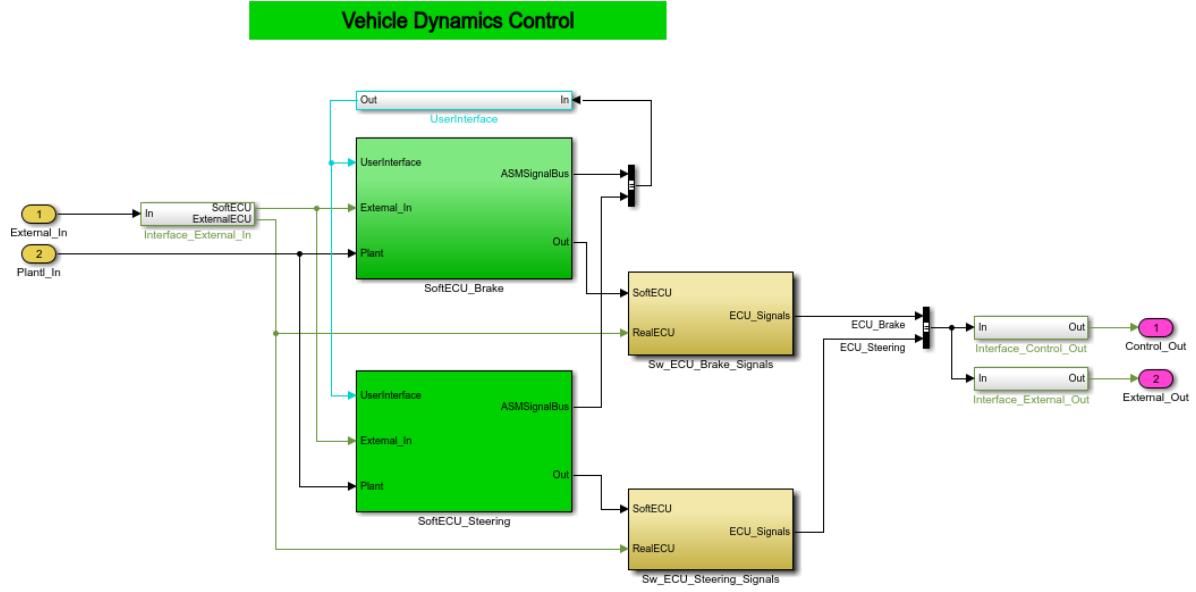
Name	Unit	Description
External_In	[]	To connect signals from other ASM modules.
Plant_In	[]	To connect signals from the Plant block.

**Outputs** The block has the following outputs:

Name	Unit	Description
Control_Out	[]	Provides signals to the Plant block.
External_Out	[]	Provides signals to other ASM modules.

## Structure

The following illustration shows an example of the structure inside the **Control** block:



**Interface\_External\_In** This interface provides signals from external ASM modules or real ECU signals via I/O.

**Interface\_Control\_Out** This interface provides control signals to the Plant subsystem of this ASM module.

**Interface\_External\_Out** This interface provides controller signals for the other ASM modules of the model.

**<SoftECU\_XX> block** The block contains the functional part of the controller model.

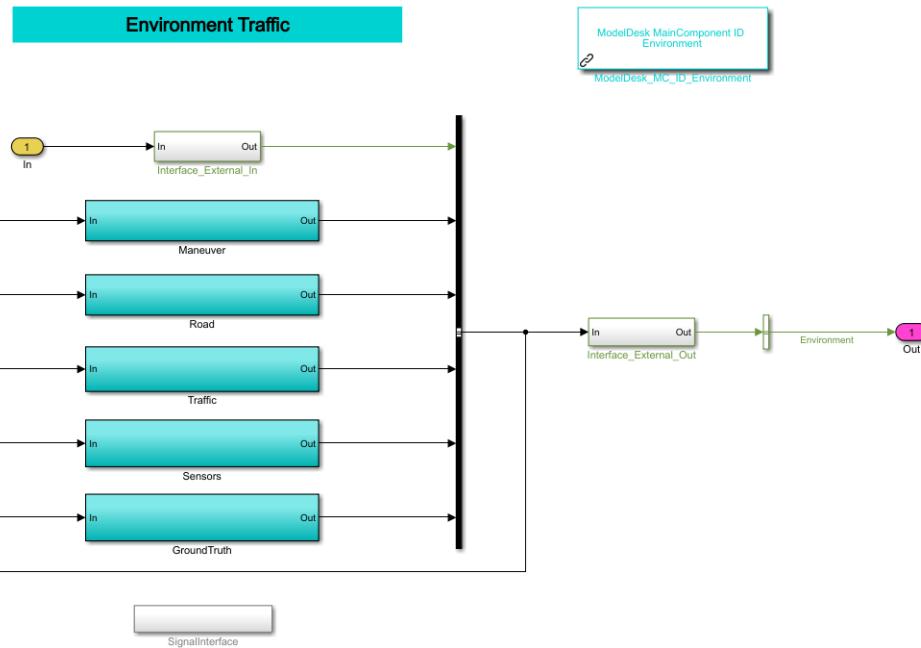
**Sw\_ECU\_Signals** (optional) This switch lets you select between signals from a real ECU and the soft ECU model.

**UserInterface** This interface is a central access point for different functions and settings inside the model. Refer to [User Interface](#) on page 30.

## ASM Submodule

### Introduction

An ASM module can contain further ASM submodules.



### Function

ASM submodules are used if the ASM module contains different functional subsystems.

These functional subsystems cannot form ASM modules on their own.

### Structure

An ASM submodule contains:

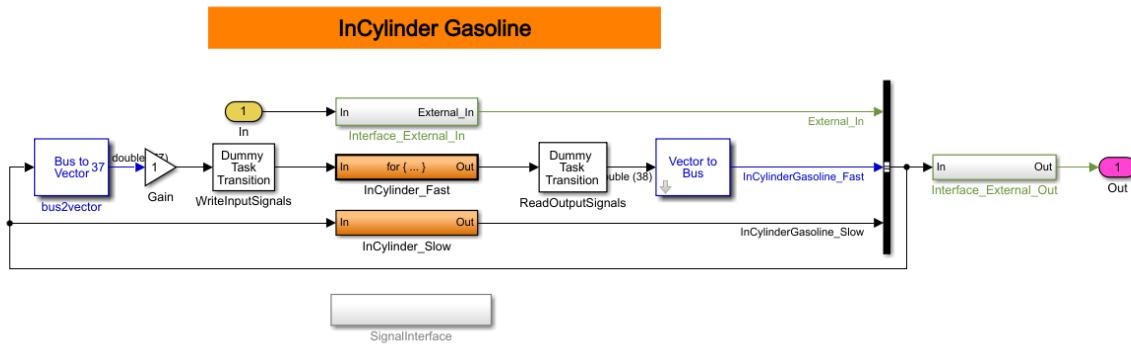
- A Plant and/or a Control block.

## ASM Task Submodule

### Introduction

An ASM module can contain ASM task submodules.

An ASM task submodule can be oversampled in simulation.



### Function

ASM task submodules are used if a functional subsystem of the ASM module is to be simulated with a different sample rate than the rest of the model.

### Structure

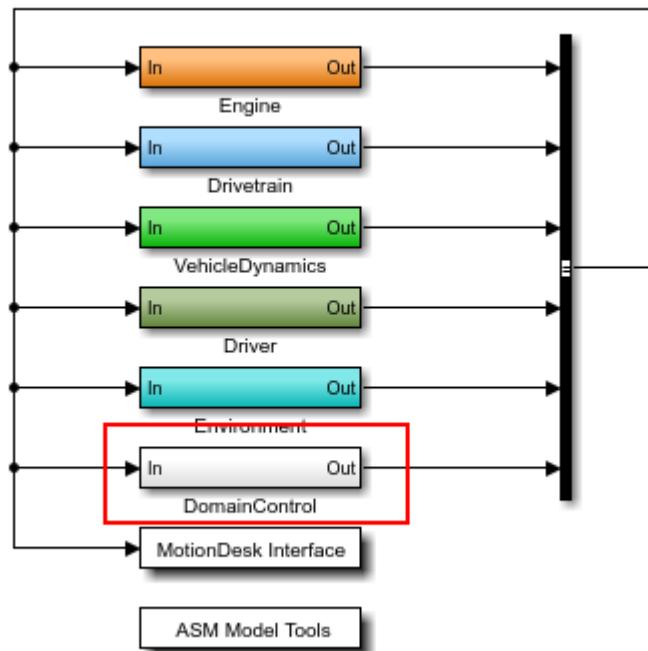
An ASM task submodule contains:

- A Plant and/or a Control block.

## Domain Control

### Introduction

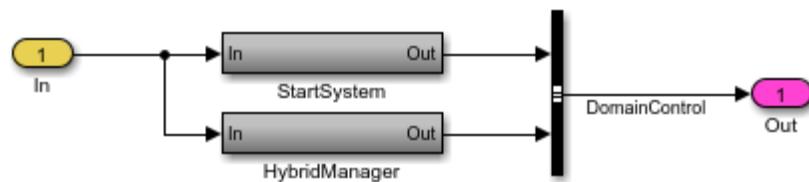
For common control functionalities, such as *hybrid manager* or *driver assistance*, it is also possible that several ASM modules are grouped in one block of the ASM model. This is done in the DomainControl block.



### Structure

The common control functions do not have the typical control loop structure used in the other ASM modules.

In this example, the DomainControl block contains the StartSystem and DriverAssistance ASM modules.

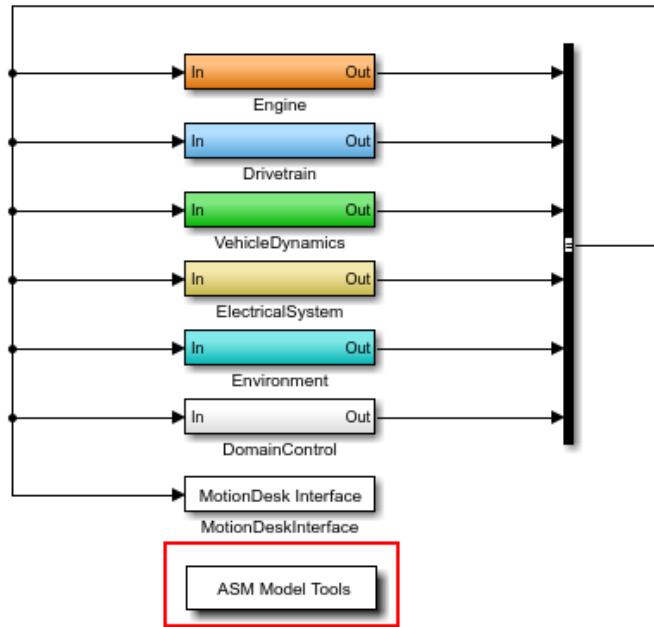


This grouping has no functional effect but is only done for better presentation in the model.

# ASM Model Tools

## Introduction

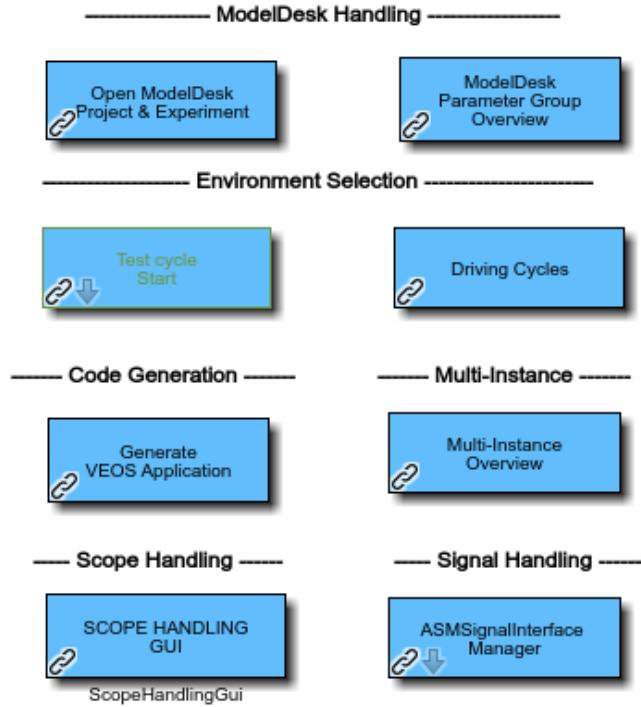
The ASM Model Tools subsystem contains different common functionalities for working with the ASM models.



## Structure

When you open the ASM Model Tools subsystem, it contains different blocks.

The following illustration shows an example.



## Functionalities

The ASM Model Tools can contain the following functionalities:

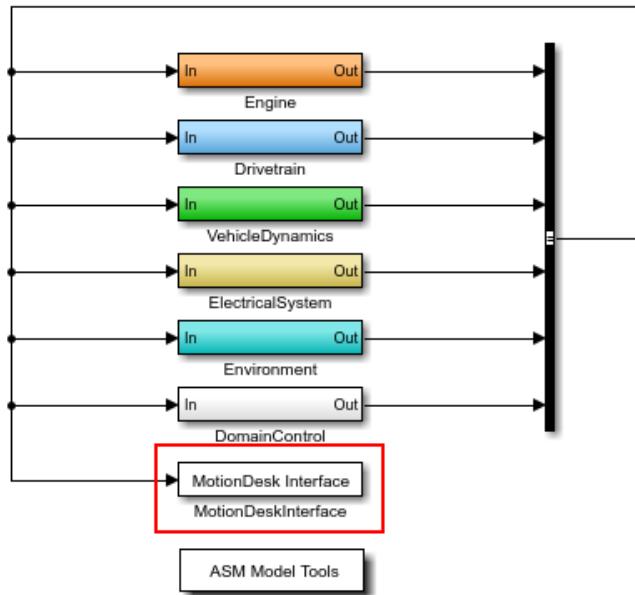
- Project and experiment handling
  - Open MotionDesk project and experiment: [Open Experiment](#) on page 193
  - Open ModelDesk project and experiment: [Open Experiment](#) on page 193
- ModelDesk parameter group:
  - ModelDesk Parameter Group Overview: [ModelDesk Parameter Group Overview](#) on page 166
- Environment selection:
  - Maneuver control: [Select Maneuver](#) on page 186
  - Driving cycles: [Test Cycle](#) on page 184
- Tire handling:
  - Replace tire model with TMEasy tire model: [Replace Tire TMEasy](#) on page 195
  - Replace tire model with Magic Formula tire model: [Replace Tire MagicFormula](#) on page 196
- Code generation:
  - Generate VEOS application: [Generate VEOS](#) on page 196
- Multi-instance:
  - Open multi-instance overview: [Multi-Instance Overview](#) on page 165

- Developer/operator handling:
  - Open developer library version: [Activate Developer Version on page 194](#)
  - Open operator library version: [Activate Operator Version on page 194](#)
- Scope handling:
  - Open the scope handling user interface: [Scope Handling User Interface on page 192](#)
- Signal handling:
  - ASM SignalInterface Manager: [ASMSignalInterface Manager on page 261](#)

## MotionDesk Interface

### Introduction

To animate a simulation in MotionDesk, the model must contain the MotionDeskInterface subsystem.



For information on animation in MotionDesk, refer to [Simulation and Visualization \(MotionDesk Basics\)](#).

### Function

The MotionDeskInterface block provides the interface to MotionDesk for animation. Signals required for animation are imported to the MotionDesk Interface from all ASM modules in the model. This subsystem calculates the motion data and transfers it to MotionDesk.

**Related topics****Basics**

[Simulation and Visualization \(MotionDesk Basics\)](#)

## User Interface

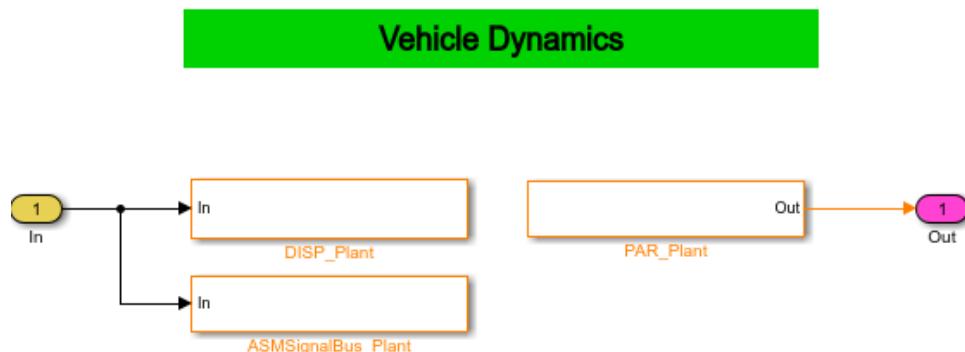
**Basics**

The UserInterface subsystem is used as a central access point inside the model.

**Structure**

The UserInterface subsystem is located inside every Plant and Control subsystem.

Depending on the model, the UserInterface contains different subsystems, but always has a similar structure.



It can contain the following subsystems:

- DISP: Contains different plotters for visualization during simulation.
- PAR: Contains the variant switches and parameters for running the model in different modes.
- TEST: Provides access to simulation results.
- MEAS: Provides access to steady-state measurement signals in the simulation model.

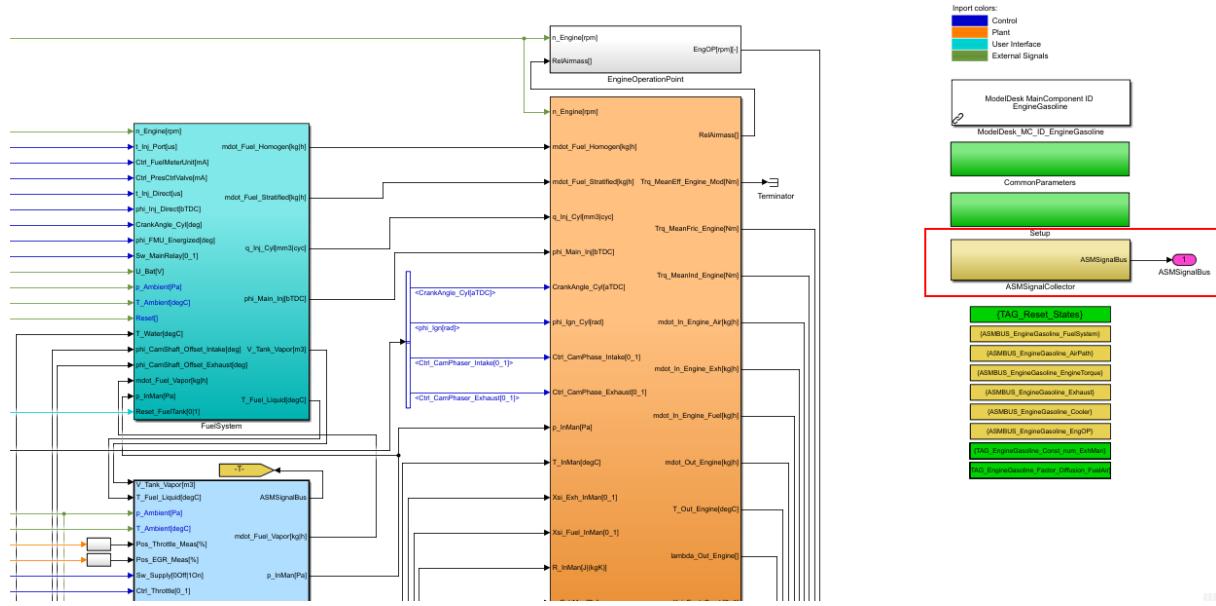
---

<b>Functions</b>	<p>Some functionalities of the UserInterface are:</p> <ul style="list-style-type: none"><li>▪ Accessing signals from the simulation model for display reasons.</li><li>▪ Setting parameters (for example, switching between manual and automatic transmission).</li><li>▪ Accessing signals from the simulation model for postprocessing.</li><li>▪ For ASM Engine Models: Evaluation of the simulation results by ASM Engine Testbench or with the measurement interface.</li><li>▪ Providing access to ControlDesk.</li><li>▪ Providing the ASMSignalBus for access to the SignalInterface.</li></ul>
------------------	---

## ASMSignalBus

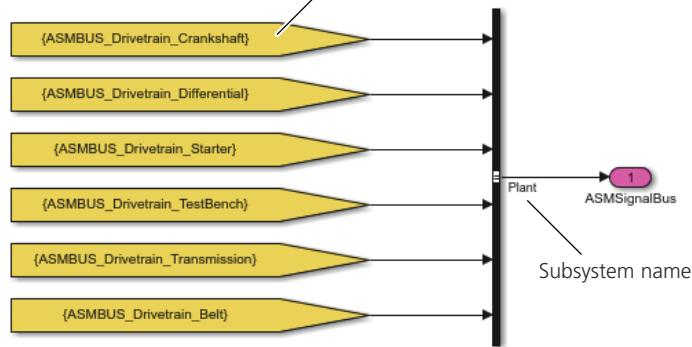
---

<b>Introduction</b>	<p>The ASMSignalBus is used inside each ASM module. It contains signals of ASM components.</p>
<b>Description</b>	<p>The ASMSignalBus is used in the Plant and Control subsystems to route signals to the UserInterface of the same module, for instance, to allow ControlDesk access to the signals.</p> <p>It is also used to route selected signals from the Plant subsystem to other ASM modules.</p> <p>The signals are accessible via the ASMSignalCollector blocks.</p> <p>The ASMSignalBus is used to provide signals in the SignalInterface block for the ModelDesk plotting feature.</p>
<b>ASMSignalCollector</b>	<p>The following illustration shows the Simulink block structure of the ASM Gasoline Engine model as an example. On the right, there is an ASMSignalCollector subsystem which collects the ASMSignalBus from each subsystem.</p>

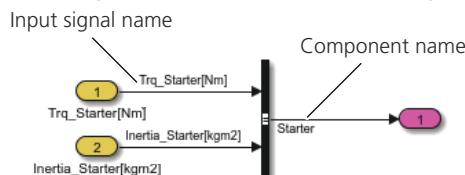


In an ASMSignalCollector block, all signals from the individual library blocks are collected by Goto/From connections and passed to a signal bus.

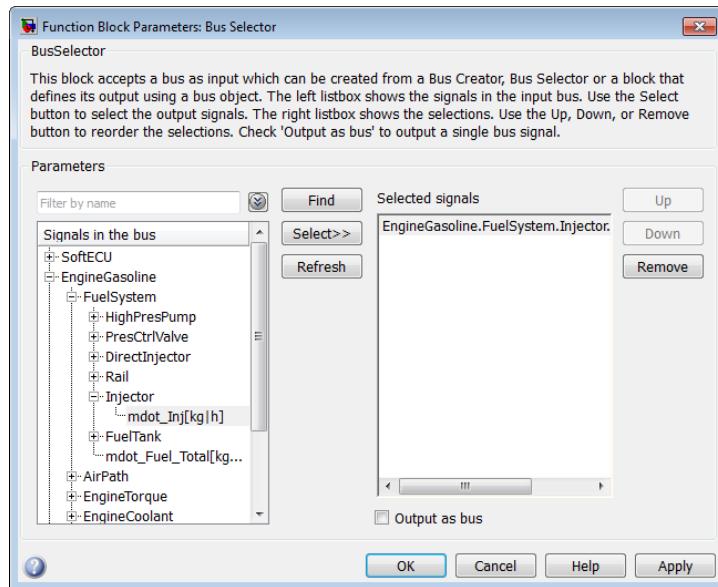
From blocks from every library block



Every library block also has an ASMSignalCollector that collects all the signals for the signal collector blocks on the top layer and forms a signal bus. The following illustration shows how these signal buses are created.



All signals in the ASM signal collector have signal labels and can easily be identified and chosen via Simulink bus selector blocks. As an example, the following illustration shows how the signal bus is grouped in the ASM Gasoline Engine model and how you can select the injected fuel mass flow:



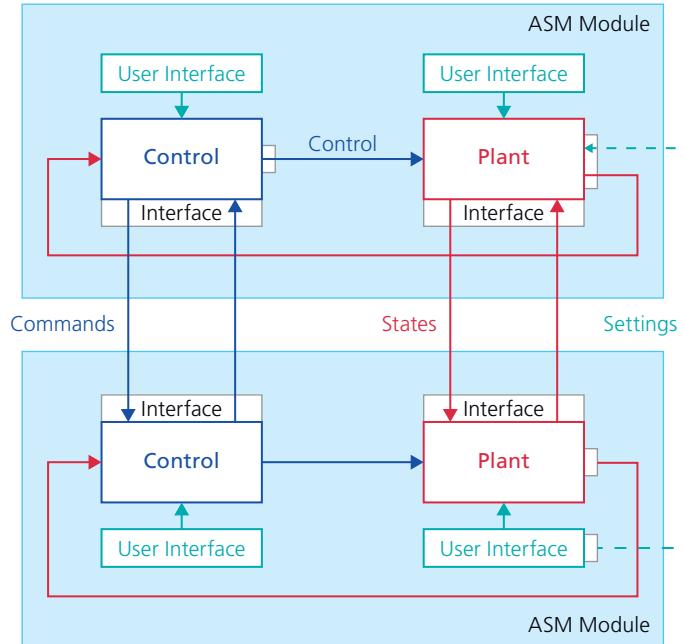
You can use the ASMSignalCollector signal bus to display offline results in the UserInterface of each ASM module.

## Communication Signals

### Introduction

For the communication between and inside the ASM modules, different types of signals are used.

The following illustration shows how the communication between different ASM modules works:



## Signals

There are signals of the following types:

**States** State and further signals of the Plant model. In some cases, the Control model can also have state signals.

**Controls** Control signals (manipulated variables) from the Control model to the Plant model, usually submodel-internal signals.

**Commands** Request signals from the Control model to other controllers.

**Settings** Signals and parameters from the UserInterface of the module.

**Animation** Animation signals for the MotionDesk Interface.

# Model Initialization

## Where to go from here

## Information in this section

### [Overview of Model Initialization](#)..... 35

Initializing an ASM model is done by a `go.m` file inside your project folder. This function manages the initialization and the variant handling of your ASM simulation model.

### [Model Initialization with go.m File](#)..... 36

With the `go.m` file you can define a simulation mode, a variant, the desired platform and a driving cycle (only engine models).

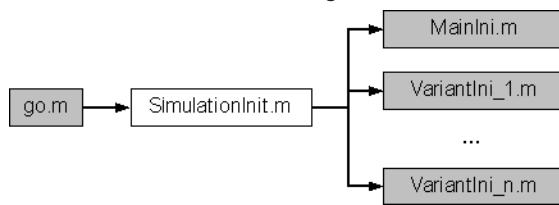
### [Handling of Main and Variant-Depending Initialization Files](#)..... 38

You can setup a project folder in order to deal with the `go.m` file.

## Overview of Model Initialization

### Overview

Initializing an ASM model is done by a `go.m` file inside your project folder. This function manages the initialization and the variant handling of your ASM simulation model. The following illustration shows the process:



The boxes to the right indicate functions which are part of your project folder. The box in the middle stands for the generic `SimulationInit.m` file, which is part of your ASM installation. When you call a `go.m` file, the generic `SimulationInit.m` file is executed. This file calls the `MainIni.m` file and, if necessary, different `VariantIni.m` files.

For more information, refer to:

- [Model Initialization with go.m File on page 36](#)
- [Handling of Main and Variant-Depending Initialization Files on page 38](#)

## Model Initialization with go.m File

### Introduction

With the `go.m` file you can define a simulation mode, a variant, the desired platform and, for engine models, a driving cycle.

### Simulation mode

The simulation mode defines the simulation environment your model should be simulated in. The following simulation modes are available.

Simulation Mode	Description
PC	Simulink simulation on PC
HIL	Real-time simulation on real-time hardware with an ECU connected
CPT	Real-time simulation on real-time hardware without an ECU connected

The standard simulation mode can be changed in the following command line of the `go.m` file

```
DefSimMode = 'PC';
```

You can call your `go.m` file with a simulation mode, for example,

```
go
go PC
go ('simmode','PC')
```

### Variant

Using the variant, you can define which initialization should be loaded to the MATLAB workspace. You can easily add further variants by attaching other variant structures, for example,

```
godata.variants.engine = { ...
    'diesel4cyl12l',...
    'diesel4cyl12ladvturbo'};
```

```
godata.variants.drivetrain = { ...
    'DrivetrainFWD',...
    'DrivetrainRWD',...
    'DrivetrainAWD'};
```

Append own variants in the same way. Make sure that you name a new variant list as a MATLAB structure variable with `godata.variants.` as prefix.

You can now call your `go.m` file with a variant name, for example,

```
<CodeBlock>go ('engine','diesel4cyl12ladvturbo')</CodeBlock>
```

```
go ('PC','engine','diesel4cyl12ladvturbo')
go ('simmode','PC','engine','diesel4cyl12ladvturbo')
```

**Note**

If you call a `go.m` file without variant arguments, the default variants are initialized. They are always on the first position of each variant definition list.

**Platform**

With the `platform` parameter, you can specify for which platform the model is prepared after it has been opened by the `go` file. The available platforms are defined in the `go` file.

```
DefPlatform = {'RTI',...
    'SCALEXIO',...
    'VEOS'};
```

You can now call your `go.m` file with a platform, for example,

```
go ('platform','VEOS')
go ('PC', 'platform','VEOS')
go ('simmode','PC', 'platform','VEOS')
```

**Note**

If you call a `go.m` file without platform argument, the default platform is set. It is always on the first position of the platform definition list.

The following changes are applied to the model:

**RTI** The '`asm_set_rti`' function is executed which adopts the system target file according to the currently active RTI platform.

More compiler settings are applied according to the settings in the `go.m` file.

**SCALEXIO** No changes are applied to the model.

Changes are done in ConfigurationDesk before the build. Nevertheless you should select this platform when working with SCALEXIO to avoid switching back and forth between platforms including interruption by user interfaces.

**VEOS** The '`asm_set_veos`' function is executed. It adopts all the settings needed for VEOS code generation.

**Driving cycles****Note**

The `drivingcycle` argument is only available in engine projects.

With the `drivingcycles` parameter, you can specify the driving cycle to be loaded by the `go.m` file. The available test cycles are defined in the `go.m` file:

```

drivingcycles = {...  

    'Ftp_75',...  

    'AC',...  

    'ESC',...  

    'ETC',...  

    'EUDC',...  

    'EUDC_with_6gear',...  

    'EUDC_with_gear',...  

    'FTP_Transient',...  

    'Ftp_75_short',...  

    'HIGHWAY',...  

    'JP_JE05',...  

    'JP_JE05_with_Gear',...  

    'Jap_10_15',...  

    'SC03',...  

    'US06',...  

    'WHTC',...  

    'ffe_city',...  

    'JP_JC08',...  

    'WLTC_Class1',...  

    'WLTC_Class2',...  

    'WLTC_Class3',...  

};

```

You can now call your `go.m` file with a test cycle name, for example,

```

go ('drivingcycle', 'AC')
go ('PC', 'drivingcycle', 'AC')
go ('simmode', 'PC', 'drivingcycle', 'AC')

```

#### Note

If you call a `go.m` file without `drivingcycle` argument, the default test cycle is set. It is always on the first position of the `drivingcycles` definition list.

## Handling of Main and Variant-Depending Initialization Files

### Introduction

You can setup a project folder in order to deal with the `go.m` file. The initialization of your project can be split in several folders under your project folder. ASM distinguishes between two kinds of initialization folders:

- Main initialization folder
- Variant initialization folder

For information on how to insert new variants into a `go.m` file, refer to [Model Initialization with go.m File](#) on page 36.

**Main initialization folder**

The `go.m` file executes always the contents of the main initialization folder defined in the `MainIni.m` file. This file holds all initialization files inside the main initialization folder that should be executed during model initialization. The following list is an exemplary list of initialization files inside a `MainIni.m`:

```
c = 1;
MainIniFiles{c} = 'IO_ini.m';
c = c+1;
MainIniFiles{c} = 'Cpt_IO_ini.m';
c = c+1;
MainIniFiles{c} = 'Cpt_MDL_ini.m';
c = c+1;
MainIniFiles{c} = 'sim_ini.m';
```

**Variant initialization folder**

You define the variant used within the `go.m` file. This file defines which variant initialization folder is executed. Only the selected variant is initialized. The `VariantIni.m` file is therefore executed which is part of the variant initialization folder and holds the initialization files that should be executed during model initialization. The following list shows an exemplary list of initialization files inside a `VariantIni.m`:

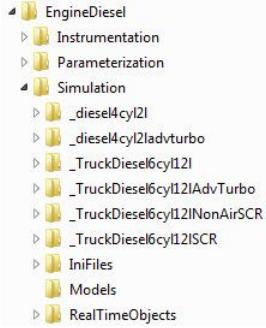
```
c = 1;
VariantIniFiles{c} = 'asm_Const_ini.m';
c = c+1;
VariantIniFiles{c} = 'asm_SoftECU_ini.m';
c = c+1;
VariantIniFiles{c} = 'asm_Engine_ini.m';
c = c+1;
VariantIniFiles{c} = 'asm_Drivetrain_ini.m';
c = c+1;
VariantIniFiles{c} = 'asm_Environment_ini.m';
c = c+1;
VariantIniFiles{c} = 'asm_VehicleDynamics_ini.m';
c = c+1;
VariantIniFiles{c} = 'changes.m';
```

**Note**

A variant initialization folder has an underscore character as a prefix to indicate this folder as a variant. It is not possible to use further underscores in the variant folder name.

## Example

The following illustration shows an example project folder.



The `_diesel4cyl2l`, `_diesel4cyl2ladvturbo`, `_TruckDiesel6cyl12l`, `_TruckDiesel6cyl12lAdvTurbo`, `_TruckDiesel6cyl12lNonAirSCR` and `_TruckDiesel6cyl12lSCR` folders contain the initialization variants each with an individual `VariantIni.m`. The `IniFiles` folder contains one `MainIni.m` which is executed during every initialization process regardless of the selected variant.

# General Information

## Where to go from here

## Information in this section

<a href="#">Folder Structure</a> .....	42
The folder structure of ASM can be divided into two parts: an installation folder and the project folders.	
<a href="#">Variable Structure</a> .....	47
ASM models are initialized via the MATLAB workspace. A special variable structure has been introduced for this purpose.	
<a href="#">Model Modifications</a> .....	52
When you work with the ASM models, you should consider some aspects to avoid malfunctions.	
<a href="#">Multi-Instance</a> .....	53
The multi-instance feature enables you to use an ASM block of an ASM Library several times in a model.	
<a href="#">Dimension Parameters</a> .....	57
ASM engine models contain parameters which influence signal dimensions.	
<a href="#">Operator Version</a> .....	61
The operator version is a model variant specifically designed for Simulink simulation.	
<a href="#">Numerical Aspects of ASMs</a> .....	63
ASM offers two techniques to overcome problems with calculation in real-time simulation.	

# Folder Structure

## Where to go from here

## Information in this section

### Basics on Folder Structure.....42

The folder structure of ASM can be divided into two parts: an installation folder and the project folders.

### ASM Installation Folder.....43

The ASM installation contains different folders.

### ASM Project Folder.....43

Gives you an overview of the project folder of ASM Engine and ASM VehicleDynamics projects.

# Basics on Folder Structure

## Basics

The folder structure of ASM can be divided into two parts: an installation folder which remains in your ASM installation and the project folders that can be copied to different places on your computer or network. You can therefore handle different simulation projects with the same ASM version.

This documentation explains the folder structure seen from an installation root called `DSPACE_ROOT`.

## Related topics

## Basics

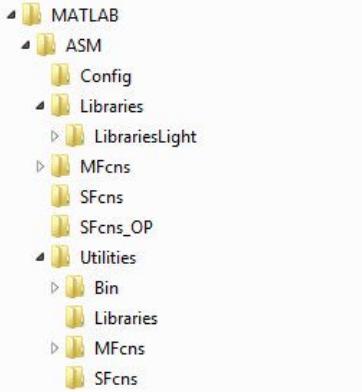
### ASM Installation Folder.....43

### ASM Project Folder.....43

## ASM Installation Folder

### Description

The following illustration gives you an overview of the folders in your ASM installation.



ASM is installed in <RCP\_HIL\_InstallationPath>\MATLAB with the following subfolders:

Folder	Description
Libraries	Contains Simulink libraries for the different models
MFcn	Contains MATLAB M-functions
SFcn	Contains MATLAB S-functions
SFcn_OP	Contains MATLAB S-functions which are related to the Operator version of ASM. This folder is optional and depends on the installed ASM products.
Utilities	Contains libraries and functions that are essential for all simulation models

Beneath the **Utilities** folder, the following subfolder are installed:

Folder	Description
Bin	Contains binary files
Libraries	Contains libraries for the different models
MFcn	Contains MATLAB M-functions
SFcn	Contains MATLAB S-functions

## ASM Project Folder

### Introduction

The structure of the project folder depends on the ASM model.

Here, you get an overview of the project folder of the following projects:

- ASM Engine, see [ASM Engine project](#) on page 44
- ASM VehicleDynamics, see [ASMVehicle Dynamics project](#) on page 45

#### Project folder location

The ASM project folder can be copied to every folder on your computer or network. ASM initializes the correct path information in the starting procedure.

##### Note

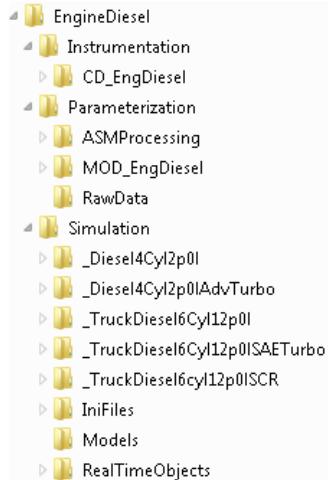
Do not copy your ASM project in deep paths.

The maximum allowed depth, where the root folder of the ASM example project starts, is 175 characters. If you have a customized project with adapted file names, for example, ModelDesk parameter files, the allowed path length up to the root folder might be even shorter. If the path is too long, the copy process will shorten it without further notice.

If you want to make a backup of your project, for example, on a server, it is recommended to create ZIP files.

#### ASM Engine project

The following illustration shows an example overview of the folders in an ASM Engine project folder.



The example is valid for the following:

- ASM Diesel Engine
- ASM Gasoline Engine
- ASM Diesel Engine InCylinder
- ASM Gasoline Engine InCylinder

You can find an example of an ASM project folder in  
`<RCP_HIL_InstallationPath>\Demos\ASM\EngineDiesel` in your dSPACE installation. It contains the following subfolders:

Folder	Description
Instrumentation	Contains ControlDesk projects and experiments. Refer to the related documentation for detailed information.
Parameterization	Contains all files related to the parameterization. See below.
Simulation	Contains all files belonging to the simulation model. See below.

**Parameterization folder** The folder can have the following additional folders:

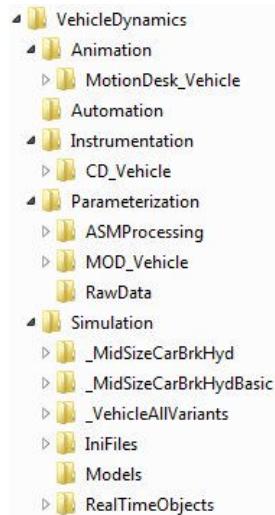
Folder	Description
ASMOptimizer	Contains all files related to the ASM Optimizer. This folder is optional.
ASMProcessing	Contains the ASM Engine Testbench related files.
MOD_<EngineType>	Contains all the ModelDesk related files.

**Simulation folder** The folder contains three or more additional folders:

Folder	Description
_<VARIANTNAME>	Contains initialization files for the simulation model depending on the selected variant.
IniFiles	Contains initialization files for the simulation model and test cycle definition.
Models	Contains the simulation model.
RealTimeObjects	Contains ZIP files with precompiled real-time objects for dSPACE simulation platforms.

## ASMVehicle Dynamics project

The following illustration gives you an example overview of the folders in an ASM Vehicle Dynamics project folder.



The following ASM projects have a folder structure similar to ASM Vehicle Dynamics:

- ASM Trailer
- ASM Truck
- ASM Traffic

You can find the ASM project folder in `<RCP_HIL_InstallationPath>\Demos\ASM\VehicleDynamics` in your dSPACE installation, for example. It contains the following subfolders:

Folder	Description
Animation	Contains MotionDesk projects and experiments. Refer to the related documentation for detailed information.
Automation	Contains demo scripts for ModelDesk automation.
Instrumentation	Contains ControlDesk projects and experiments. Refer to the related documentation for detailed information.
Parameterization	Contains all files related to the parameterization. See below.
Simulation	Contains all files belonging to the simulation model. See below.

**Parameterization folder** The folder can have the following additional folders:

Folder	Description
ASMProcessing	Contains the ASM Road Converter files to create road models from measured road data.
MOD_<EngineType>	Contains all the ModelDesk related files.

**Simulation folder** The folder contains the following additional folders:

Folder	Description
_<VARIANTNAME>	Contains initialization files for the simulation model depending on the selected variant.
IniFiles	Contains initialization files for the simulation model, maneuver and road definition.
Models	Contains the simulation model.
RealTimeObjects	Contains ZIP files with precompiled real-time objects for dSPACE simulation platforms.

# Variable Structure

## Where to go from here

## Information in this section

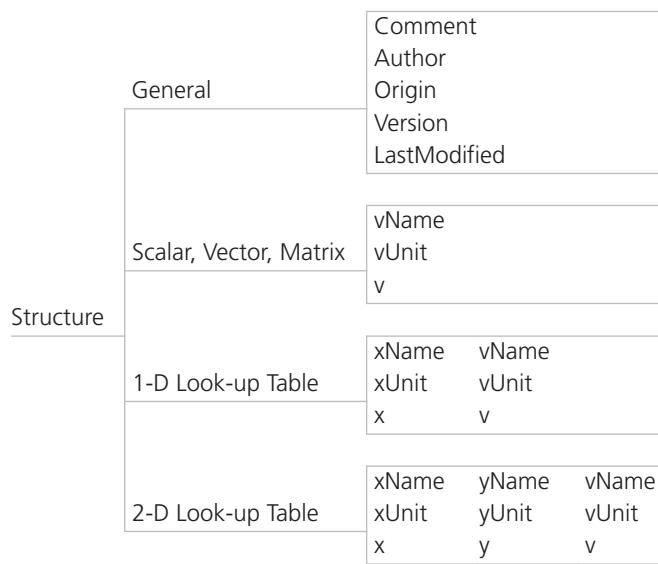
<a href="#">Basics on Variable Structure</a> .....	47
ASM models are initialized via the MATLAB workspace. A special variable structure has been introduced for this purpose.	
<a href="#">General Structure of Variables</a> .....	48
The general part of this definition is contained in every kind of variable.	
<a href="#">Scalar, Vector, Matrix</a> .....	49
Shows the structure of a scalar.	
<a href="#">1-D Look-up Table</a> .....	49
Shows the structure of a 1-D look-up table.	
<a href="#">2-D Look-up Table</a> .....	50
Shows the structure of a 2-D look-up table.	

## Basics on Variable Structure

### Variable structure

ASM models are initialized via the MATLAB workspace. A special variable structure has been introduced for this purpose. There is a general structure and different approaches for scalar, vector, matrix, 1-D look-up table, and 2-D look-up table values.

All the parameters for the initialization of ASM models are stored in MATLAB structure variables. The following illustration shows the variable structure:



The general part of this definition is contained in every type of variable. The other parts vary according to the variable.

## Related topics

### Basics

1-D Look-up Table.....	49
2-D Look-up Table.....	50
General Structure of Variables.....	48
Scalar, Vector, Matrix.....	49

## General Structure of Variables

### Description

The general part of this definition is contained in every kind of variable. The following table shows the general structure.

Structure	Description
Comment	Contains explanations on the variable
Author	Contains the name of the editor
Origin	Contains information about the parameter origin, for example, ECU software
Version	Contains ASM version information
LastModified	Contains date of last revision

**Related topics****Basics**

1-D Look-up Table.....	.49
2-D Look-up Table.....	.50
Scalar, Vector, Matrix.....	.49

## Scalar, Vector, Matrix

**Description**

The following table shows the structure of a scalar:

Structure	Description
vName	Name of scalar parameter
vUnit	Unit of scalar parameter
v	Scalar parameter

**Example**

```
MDL.EngineDiesel.Const.Const_R_Air = struct...
'Comment', 'Gas constant of air',...
'Author', 'dSPACE',...
'Origin', '',...
'Version', '1',...
'LastModified', '01.01.2015 00:00',...
'veName', 'Const_R_Air',...
'veUnit', '[J/(kgK)]',...
've', 287);
```

**Related topics****Basics**

General Structure of Variables.....	.48
Variable Structure.....	.47

## 1-D Look-up Table

**Description**

The following table shows the structure of a 1-D look-up table:

Structure	Description
xName	Name of input vector
xUnit	Unit of input vector
x	Input vector

Structure	Description
vName	Name of output vector
vUnit	Unit of output vector
v	Output vector

**Example**

```





```

**Related topics****Basics**

General Structure of Variables.....	48
Variable Structure.....	47

## 2-D Look-up Table

**Description**

The following table shows the structure of a 2-D look-up table:

Structure	Description
xName	Name of first input vector

Structure	Description
xUnit	Unit of first input vector
x	First input vector
yName	Name of second input vector
yUnit	Unit of second input vector
y	Second input vector
vName	Name of output vector
vUnit	Unit of output vector
v	Output vector

**Example**

```





```

**Related topics****Basics**

General Structure of Variables.....	48
Variable Structure.....	47

# Model Modifications

## Model Modifications

---

<b>Introduction</b>	If you make any modifications to the ASM models, note some points to avoid errors.
---------------------	--

<b>Broken links</b>	<p>Do not break or disable the link between an ASM block and its library.</p> <p>The following problems can occur when a library link is disabled or broken:</p> <ul style="list-style-type: none"><li>▪ The block might not be migrated correctly. Automatic migration is possible only for ASM blocks with an enabled library link.</li><li>▪ When you switch between the developer and operator version, broken links can lead to errors, especially in the case of missing licenses, i.e., if you only have an operator or a developer license.</li><li>▪ In case of migration, the parameterization via ModelDesk is possible only for blocks with an active library link. If the path of a block/variable has been changed in a newer version, for example, ModelDesk cannot access it and this can lead to errors.</li><li>▪ Support for requests which include ASM blocks with broken or disabled library links is difficult and limited.</li></ul> <p>If you require proprietary developments or additions, you are recommended to implement them outside the respective ASM library block. This way, the correct behavior of the ASM blocks can be maintained.</p>
<b>Library modification</b>	<p>The ASM libraries and the included blocks provided with the installation must not be modified. Otherwise, data loss can occur during the installation of patches, for example.</p>
<b>Multi-instance support</b>	<p>When you copy ASM blocks, keep in mind that not all ASM blocks support the multi-instance feature. For more information, refer to <a href="#">Multi-Instance</a> on page 53.</p>
<b>Custom libraries</b>	<p>It is not recommended to encapsulate ASM blocks in your own custom libraries. This can lead to license problems during code generation. Furthermore, these encapsulated ASM blocks can also lead to errors during the migration process.</p>

# Multi-Instance

## Where to go from here

## Information in this section

### [Multi-Instance](#).....53

The multi-instance feature enables you to to use an ASM block of an ASM Library several times in a model.

### [ASM Blocks Not Supporting the Multi-Instance Feature](#).....55

The multi-instance feature is not supported by all the ASM blocks.

# Multi-Instance

## Introduction

The multi-instance feature enables you to to use and parameterize an ASM block of an ASM Library or a ModelDesk Custom Library several times in a model.

## Multi-instance feature

An ASM block of an ASM Library or a ModelDesk Custom Library can be used several times in a model. ModelDesk provides the multi-instance feature to be able to access each of these blocks. To get an explicit description of duplicated blocks, each ASM and Custom Library block has its own instance ID and instance name. To handle and show this block in ModelDesk, the instance ID must be different for duplicated blocks of the same type and is used together with the instance name to handle and show this block in ModelDesk.

**Note**

- Duplicated blocks can only be used in the same main component subsystem (for example, Soft ECU, Engine, Vehicle Dynamics and Environment subsystems of the ASMVehicleDynamics model) as the original source block.
- Some ASM blocks (for example, road, maneuver) do not support the multi-instance feature (see [ASM Blocks Not Supporting the Multi-Instance Feature](#) on page 55).  
When you copy ASM blocks, keep in mind that not all ASM blocks support multi-instances.
- In general it is possible that duplicated blocks can have the same mask parameters. In this case note that the parameterization with ModelDesk of a model in a Simulink simulation is not clearly defined for the same mask parameters. ModelDesk provides an own parameter page for each of the duplicated blocks. So each block instance has got an own parameter file which may have different parameter values. If these parameter files are downloaded to the MATLAB workspace, the same mask parameter is assigned twice with different values.  
The parameter download to a real-time hardware is not affected by double mask parameters.

**Different ASM blocks belong to one ModelDesk parameter page** In some cases, different ASM blocks are parameterized with the same ModelDesk parameter page. For example, the TORQUE\_CONVERTER and LOCKUP\_CLUTCH blocks are parameterized with the Torque Converter parameter page.

To define which ASM blocks should belong to which parameter page, the `instance name` and `instance ID` parameters are used. ASM blocks that should be parameterized with the same ModelDesk parameter page must have the same `instance ID` and same `instance name`.

These settings are unimportant if one parameter page parameterizes only one ASM block.

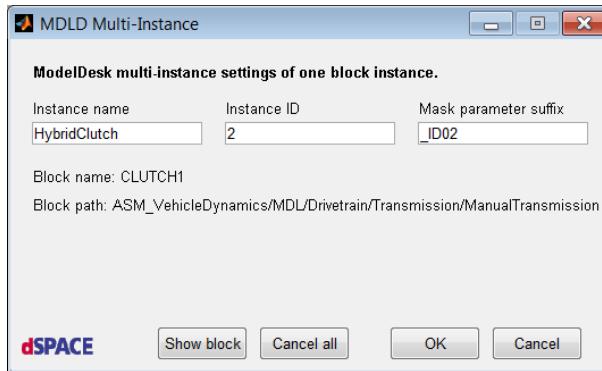
**Overview of multi-instances**

To change the multi-instance parameters without copying the ASM block or to get an overview of all multi-instance blocks used in the model, refer to [asm\\_multiinstance](#) on page 201.

You can also get an overview via the MultiInstance Overview block in your model. Refer to [Multi-Instance Overview](#) on page 165.

**Functionality**

If an ASM or Custom Library block is copied, the following dialog opens to set the multi-instance block parameters.



The dialog contains the following elements:

**Instance name** Lets you specify the instance name string that is displayed in ModelDesk.

**Instance ID** Lets you specify the instance ID to define the unique block instance. The instance ID is displayed in ModelDesk.

**Block name** Displays the Simulink block name.

**Block path** Displays the model path to the block.

**Show block** Opens the Simulink model subsystem of the block.

**Cancel all** Closes all multi-instance dialogs.

**OK** Closes the dialog and sets the multi-instance parameters.

**Cancel** Closes the dialog without setting multi-instance parameters specified in the dialog. Note that the instance ID is set automatically to hide blocks of the same type with the same ID anyway.

**Related topics****Basics**

[Parameterizing ASM Blocks of the Same Type \(ModelDesk Parameterizing\)](#)

## ASM Blocks Not Supporting the Multi-Instance Feature

**Introduction**

The multi-instance feature is not supported by all the ASM blocks. This topic provides a list of these ASM blocks.

### Unsupported ASM blocks

The following ASM blocks do not support the multi-instance feature:

- BASIC\_ROADS
- BATTERY\_CELL
- BATTERY\_THERMAL
- CONTINUOUS\_VALVE\_CONTROL
- DRIVETRAIN\_VARIANT\_SWITCHES
- FELLOW\_MOVEMENT
- FELLOW\_PARAMETERS
- LANESENSOR\_ENABLE
- MANEUVER\_SCHEDULER
- MANEUVER\_SELECT\_SWITCH
- MANEUVER\_START
- MANEUVER\_STATE
- MANEUVER\_STOP
- MODEL\_DESK\_WRITE\_ACCESS
- POSITION\_ERRORS
- RESET
- ROAD
- ROAD\_SELECT\_SWITCH
- SENSOR\_PARAMETERS
- SWITCHES\_DRIVETRAINBASIC
- TRAFFIC\_OBJECTS
- TRAFFIC\_SCHEDULER
- v\_ROAD\_REF
- VEHICLE\_DYNAMICS\_VARIANT\_SWITCHES

---

### Related topics

#### Basics

Multi-Instance.....	53
---------------------	----

# Dimension Parameters

## Dimension Parameters

### Introduction

ASM Engine models contain parameters which influence signal dimensions. As a general rule, the dimension parameters define the number of component parts (e.g., cylinder, manifolds) or events (e.g., injections) which are reflected in the model by the sizes of vectors and matrices. This number is fixed and cannot be changed during the simulation and after code generation. These parameters have been introduced, firstly to avoid variable signal dimensions, and secondly, for reasons of calculation performance. For dimension parameters, an upper limit can be applied due to the fixed size of depending parameter vectors and matrices. For some of the dimension parameters, there exists a second parameter that describes the used range up to the dimension parameter.

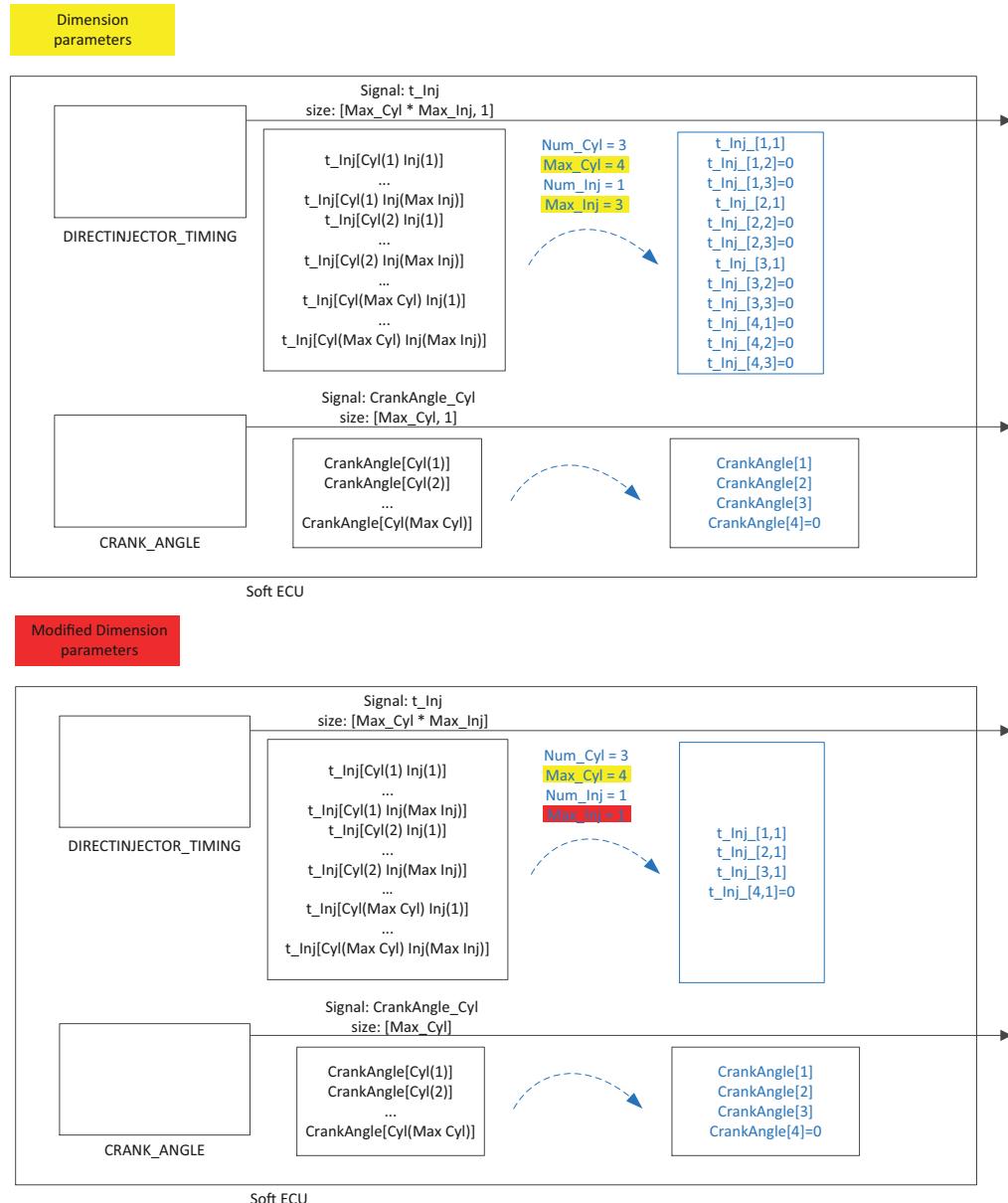
### Example

An example for a dimension parameter is shown below:

Signal dimension in current compiled code (adaption requires new code generation for online simulation)		
Limit for cylinder selective signals, maximal number is 20 <input type="text" value="8"/> [-]	Limit for direct injection selective signals, maximal number is 8 <input type="text" value="4"/> [-]	Limit for port injection selective signals, maximal number is 4 <input type="text" value="1"/> [-]
Number of rail systems <input type="text" value="1"/> [-]	Number of exhaust systems <input type="text" value="1"/> [-]	Number of cells for the SCR Catalyst <input type="text" value="3"/> [-]

In this example, the maximal number of cylinders is set to 8. Hence, the actual number of cylinders can be modified in the range of 1 to 8.

The modification of dimension parameters can lead to inconsistencies in signal dimensions of the model. When you change one of the dimension parameters, you have to execute the ModelDesk Processing to ensure that all derived parameters (e.g., injection signal mapping) are consistent. The following illustration shows the dependency of these parameters, for clarification:

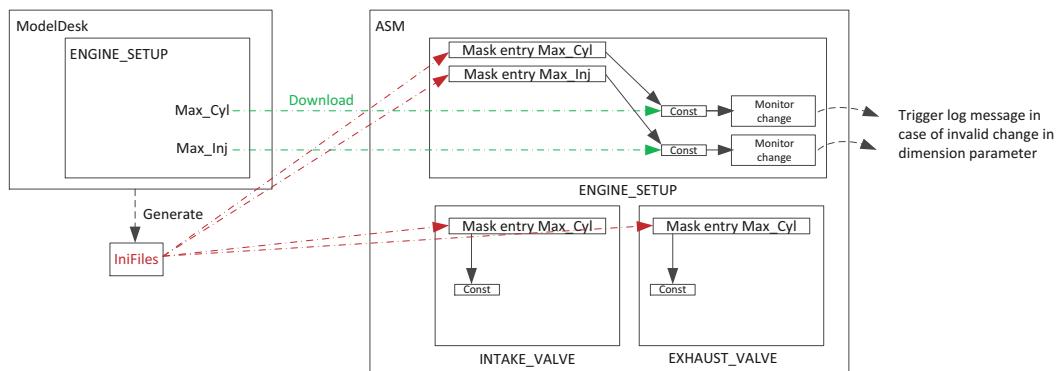


## Exceptions

The following exceptions apply to dimension parameters:

**Exception 1** Dimension parameters are used several times in the model by different ASM blocks but occur only once in ModelDesk on the Engine Setup parameter page. Therefore, it is required that all occurrences of a dimension parameter have the same mask parameter. The number of cylinders is used, for example, by the ENGINE\_SETUP, INTAKE\_VALVE, EXHAUST\_VALVE, SINGLEZONE\_CYLINDER, DIRECTINJECTOR blocks and others. MDL.<EngineType>.Setup.Const\_max\_num\_Cyl is the default mask value. As the parameters are used to specify the width of signal they cannot be distributed by From Goto connections as the constants within the COMMON\_ENGINE\_PARAMETER block.

**Exception 2** The ENGINE\_SETUP parameter page contains the dimension parameters. A check mechanism is incorporated in the block. It monitors the value of these parameters that is set during compile process and the actual values of the parameters. If a discrepancy is identified (e.g., because the dimension parameter has been changed after ModelDesk download), an error is issued in the Message Viewer (e.g., in ModelDesk or ControlDesk) to inform about the modification of a parameter and a potential malfunction of the model. For information on how to solve the error, refer to [Troubleshooting with Dimension Parameters Error Messages](#) on page 285.



The following table shows the dimension parameters with the related used range and their upper limit.

Dimension Parameter	Related range parameter used	Upper limit
Const_max_num_Cyl	Const_num_Cyl	20
Const_max_num_Inj_Direct	Const_num_Inj_Direct	8
Const_max_num_Inj_Port	Const_num_Inj_Port	4
Const_max_num_Ign	Const_num_Ign	1
Const_max_num_Events <sup>1)</sup>	-	-
Const_num_InMan	-	2
Const_num_ExhMan	-	2
Const_num_Rail	-	2
Const_max_num_Rail	-	2

Dimension Parameter	Related range parameter used	Upper limit
Const_num_pcv	–	2
Const_num_SCR_Cell	–	2
Const_num_Pumps	–	2
Const_num_ExhSys	–	2
Const_num_DPF	–	2
Const_num_DOC	–	2
Const_num_Throttle	–	2
Const_num_AdBluePump	–	2
Const_num_AirNonRetValve	–	2
Const_num_AirRegValve	–	2
Const_n_Cell_SCR	–	2
Const_num_PumpHose	–	2
Const_num_Intercooler	–	2
Const_num_EGR	–	2
Const_num_Comp	–	2
Const_num_Turb	–	2
Const_num_Sign_EngOP	–	–

<sup>1)</sup> Mask parameter of the APU\_Event which can correspond to ignition or injection signals

#### Note

Due to potential problems on signal dimensions caused by an improper modification of the dimension parameters, these parameters were excluded from ModelDesk download up to dSPACE Release 2015-A and were initialized to MATLAB workspace from an M file. For example, in ASM Diesel Engine this initialization file was `asm_enginediesel_ini` located in `<ProjectFolder> \Simulation\IniFiles`.

# Operator Version

## Operator Version

### Introduction

Some ASM models come with two versions:

- A developer version
- An operator version

The operator version of an ASM simulation package is a model variant specifically designed for Simulink® simulation. Typical applications are closed-loop tests of controller functions with model-in-the-loop (MIL) and/or software-in-the-loop (SIL), and also vehicle dynamics tests and parameter studies.

#### Note

The operator version is designed for Simulink simulation in normal and accelerator mode only. Real-time simulation on dSPACE platform is not possible.

The operator version is designed for single-tasking mode only. Simulation with multi-tasking mode in Simulink will fail.

### Availability

The ASM operator versions are available for some ASM blocksets, such as vehicle dynamics, chassis and engine applications.

### Activation

You can activate the operator version via the `ActivateOperatorVersion` block. Refer to [Activate Operator Version](#) on page 194.

#### Note

To ensure a correct switching between the developer and operator version never disable or break links of ASM blocks. Blocks with disabled or broken links to an ASM library are not replaced correctly. This can lead to malfunction, especially in the case of missing licenses. It is also not guaranteed that ModelDesk can handle such blocks correctly.

### Functionality

The model offers the same functionality, simulation quality and parameterization options as the standard simulation package (developer version). Models with the operator version are compatible with models with the developer version. You can parameterize models with operator version in ModelDesk. You can exchange the data sets (model parameters, road descriptions, maneuver descriptions) between

the models with operator and developer versions to reuse them supporting a seamless development process.

---

## Differences

The fundamental difference between the developer and the operator version is the way the library components are implemented in the operator version: They are encapsulated in separate systems to ensure good performance during Simulink simulation. The systems are accessible in the model so that their input/output behavior can be studied.

# Numerical Aspects of ASMs

## Where to go from here

## Information in this section

### [Basics on Numerical Aspects](#)..... 63

ASM offers two techniques to overcome problems with calculation in real-time simulation.

### [Local Subsystem Oversampling](#)..... 63

To overcome stability problems in real-time simulation, ASM uses a Simulink For Iterator subsystem for local subsystem oversampling.

### [Semi-Implicit Euler Stabilization](#)..... 65

To solve the model's stiff differential equations, a sophisticated integration method must be used to avoid the problem of numerical instability.

## Basics on Numerical Aspects

### Basics

Simulation of physical systems sometimes involves stiff differential equations. Unfortunately, this is a problem in real-time simulation, because you have a fixed simulation step size to calculate your simulation code in. A lot of common techniques for dealing with stiff equation systems are therefore not suitable for real-time simulation.

ASM offers two techniques which help to overcome this gap:

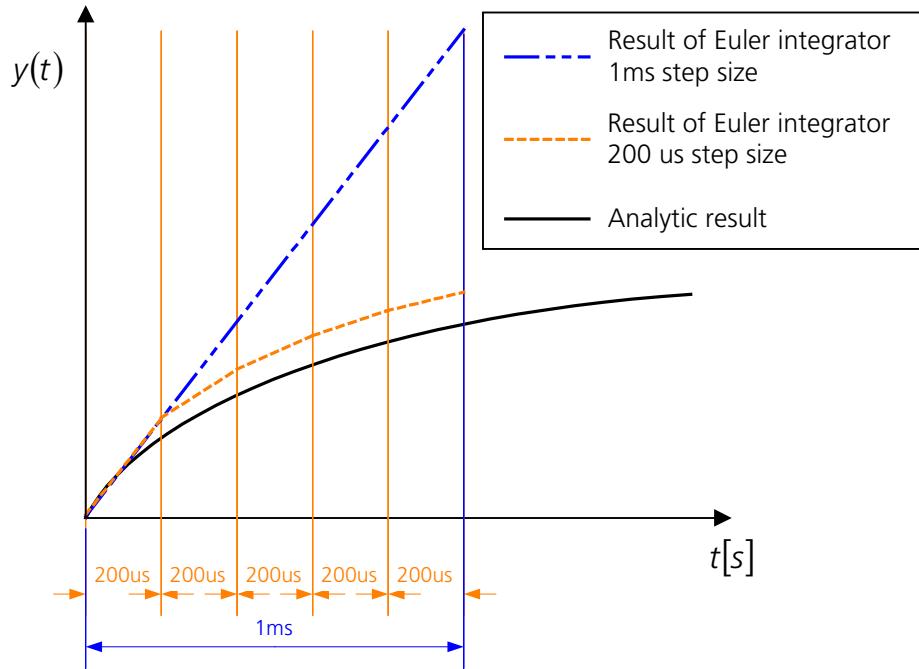
- [Local Subsystem Oversampling](#) on page 63
- [Semi-Implicit Euler Stabilization](#) on page 65

## Local Subsystem Oversampling

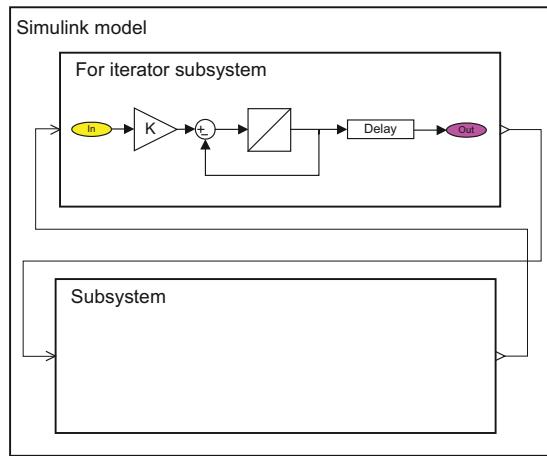
### Description

To overcome stability problems in real-time simulation, ASM uses a Simulink For Iterator subsystem for local subsystem oversampling. The idea is to evaluate subsystems with highly complex differential equations n-times during one major simulation step. This leads to a local oversampled subsystem, which is comparable to a lower step size for the entire simulation model.

The following illustration shows the effect of a subsystem oversampling for a step response of a first-order delay element:



The lowest (black) graph shows the step response of a first-order delay element. The highest (blue) graph shows the result of a forward Euler integrator with a sample time of 1 ms, the orange line shows the result for a sample time of 200  $\mu$ s. Normally, only a small part of a simulation model has to be integrated with such a small sample time. Therefore, this part can be in a Simulink subsystem which is used as a For Iterator subsystem. The next illustrations presents the basic idea:

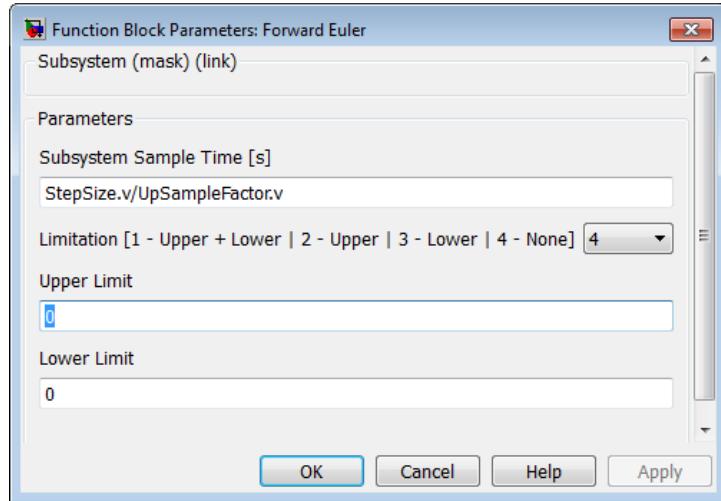


In this example the For Iterator subsystem contains a first-order delay element. For a major simulation step, the inputs of the For Iterator subsystem are set and not changed until the next major simulation step occurs. In the meantime, the For Iterator subsystem is evaluated n-times, which means the Euler integrator

can be used with a minor sample time. This enhances the accuracy of the integration algorithm for the stiff subsystem.

For Iterator subsystems need some adaptations compared with normal subsystems:

- For iterator subsystems need to have an output port for the ASMSignalBus because Goto/From connections cannot be used if they cross the For Iterator subsystem borders. For the same reason, common parameters have to be imported via signal lines.
- Simulink Integrator blocks must be replaced by an S-function integrator which accepts the minor sample times. The integration method is forward Euler. The following illustration shows the mask of this S-function.



- To ensure precise simulation results, the output signals from a For Iterator subsystem have to be delayed. This is necessary for connecting the output signals from the For Iterator subsystem with the major Simulink system. The Delay block has to be set to the number of iterations minus one to synchronize the For Iterator subsystem with the major Simulink system.

## Semi-Implicit Euler Stabilization

### Description

To solve the model's stiff differential equations, a sophisticated integration method must be used to avoid the problem of numerical instability. On the other hand, this method should not be computationally expensive. The Euler method was selected for these reasons. There are two Euler methods, forward and backward.

For the differential equation

$$\dot{x} = f(x, t)$$

where  $h$  is the simulation step size, the *forward* method reads:

$$x_{n+1} = x_n + hf(x_n, t_n)$$

and for the *backward* Euler integration

$$x_{n+1} = x_n + hf(x_{n+1}, t_{n+1})$$

holds.

The backward Euler integration is absolutely stable in the left half plane of the *s*-domain. The forward Euler, however, has limited stability. The backward Euler was therefore used to solve the stiff differential equations. However, the backward method cannot be solved explicitly. It can only be calculated iteratively, which increases the volume of computation. The function  $f(x_{n+1}, t_{n+1})$  is therefore replaced by its first two terms from Taylor series expansion. Then the function can be approximated as follows:

$$f(x_{n+1}, t_{n+1}) \cong f(x_n, t_n) + \frac{\partial f}{\partial x}(x_{n+1} - x_n) + \frac{\partial f}{\partial t}(t_{n+1} - t_n)$$

Moreover, if  $f$  is only a function of  $x$ , the following equation is obtained:

$$f(x_{n+1}, t_{n+1}) \cong f(x_n, t_n) + \frac{\partial f}{\partial x}(x_{n+1} - x_n)$$

Substituting in the backward form:

$$x_{n+1} = x_n + h \left( f(x_n, t_n) + \frac{\partial f}{\partial x}(x_{n+1} - x_n) \right)$$

and solving for  $x_{n+1}$ , yields:

$$x_{n+1} = x_n + h \left( 1 - h \frac{\partial f}{\partial x} \right)^{-1} f(x_n, t_n)$$

# Migrating ASM Models

## Where to go from here

## Information in this section

<a href="#">Basics on Model Migration</a> .....	68
The migration process updates your project including, for example, initializing new parameters and establishing new connections to block inputs or outputs.	
<a href="#">Former Versions of ASM Blocks</a> .....	69
During further development of ASM blocks a former version of a block is created in some cases.	
<a href="#">Cases Which Require Migration</a> .....	71
For some changes to ASM blocks, migration is required.	
<a href="#">Steps Performed During a Migration</a> .....	72
If you choose to migrate a model, some steps are executed.	
<a href="#">Migration Variants</a> .....	73
During migration pre- and postmigrate variants are initialized.	
<a href="#">Project Consistency</a> .....	75
The aim is to remove all pre-update and post-update initialization files in order to keep all parameter information only in the ModelDesk project.	
<a href="#">How to Migrate Blocks of Custom Component Libraries</a> .....	75
To use the multi-instance feature for blocks of custom component libraries, a migration is necessary.	

## Information in other sections

<a href="#">Automotive Simulation Models (ASM) (New Features and Migration)</a> 
---

## Basics on Model Migration

### Migration from earlier Releases

ASM provides the migration of models from earlier Releases. There are two different cases of migration depending on the Release you are migrating from:

**Automatic migration** Projects from dSPACE Release 2017-A and later can be migrated automatically.

They are migrated automatically when you open the related Simulink model in MATLAB.

**Migration from earlier Releases** Projects from Releases earlier than dSPACE Release 2017-A cannot be directly migrated but a migration to an intermediate Release is required:

1. Migrate the project to dSPACE Release 2017-A or later.
2. Migrate the intermediate version to the current Release.

#### Note

You can migrate ASM blocks only to later Releases. The blocks are not downward compatible, i.e., you cannot run the simulation of a migrated model with an older ASM version.

### Model migration

The migration process updates your project including, for example, initializing new parameters and establishing new connections to block inputs or outputs. If not stated otherwise, the new versions of the ASM blocks are used after the migration. The migration process guarantees similar model behavior as with the old ASM version. Nevertheless, for some new features manual changes might be necessary. During migration, parameter and model changes are applied, so that an "update diagram" can be performed without error. More detailed information on the changes in the model containing ASM blocks is given below.

#### Note

In case of extensive changes on a block, the old version of a block can be found in *Former Versions*. If the changes in the blocks conflict with the model behavior in the project, you can also replace the new blocks by the old ones, see the former version subsystems in the relevant ASM component library. If former version blocks are used in the model, it might be required that the parameterization is adapted manually. For more information on former versions, refer to [Former Versions of ASM Blocks](#) on page 69.

After migration, an update diagram should be executed without an error. The model can be simulated, or code can be generated without any further adaptions. However, the following warning may appear in the MATLAB Command Window:

**Warning:** In instantiating linked block '<BlockPath>' for parameter '<BlockParameterName>'.

You can ignore this warning, because it does not affect the functionality of your Simulink model. The following chapters give an overview which new features require a migration and which steps are performed during a migration of a Simulink model with ASM blocks. The executed steps depend on the used ASM blocks.

For further information on new features and changes on the ASM blocksets (from several releases) refer to [Changes to all ASM Products](#) on page 295 (for general changes) and to the *New Features and Migration History* in each blockset reference document (for changes on specific blocksets).

## Related topics

### Basics

Former Versions of ASM Blocks.....	69
Migrating a Project and Its Experiments (ModelDesk Project and Experiment Management  <td></td>	

### References

Changes to all ASM Products.....	295
----------------------------------	-----

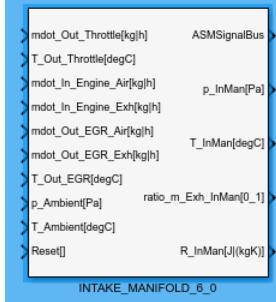
## Former Versions of ASM Blocks

### Introduction

In the further development of ASM library blocks, changes in the simulation behavior can occur. To avoid differences in the simulation results compared to the previous version, the blocks are migrated. If consistent model modifications cannot preserve the simulation behavior, a *former version* of this block is created.

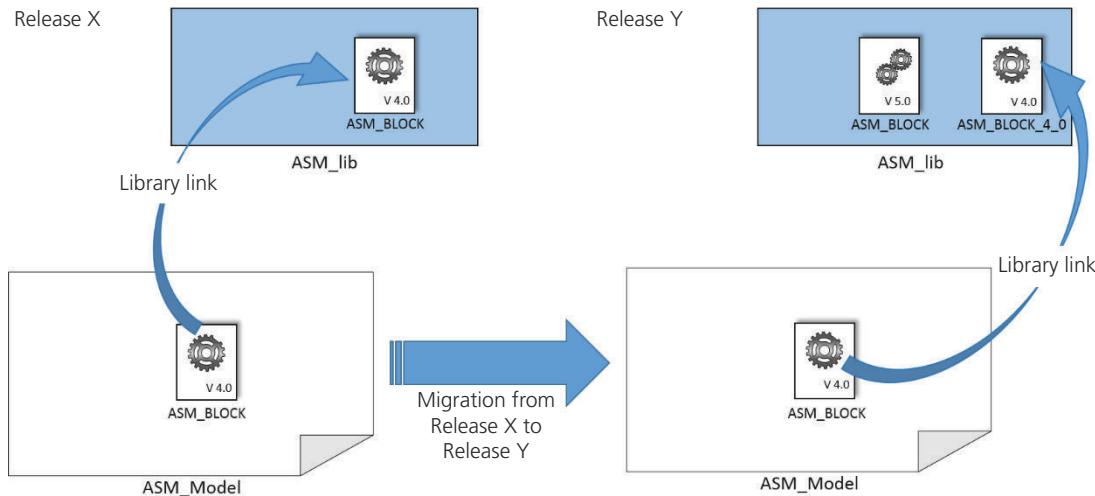
In a former version, the content of the block is kept unchanged. The development of the block then takes place in a new block. Consequently, the ASM library contains two versions of the block, a modified version and the former version. In the library, the former version blocks are placed in separate subsystems. Furthermore, the block names of the former version blocks in the ASM libraries are extended by the ASM version number with which the block has been created.

See the following example of a former version block:



### Migration process

The following illustration displays the migration to a former version.



The migration takes place for the model name `ASM_Model` from Release X to Release Y. In Release X, there was an ASM block with the name `ASM_BLOCK` and the ASM version 4.0. In Release Y, the block is developed to version 5.0. However, the simulation behavior of the block has changed in such a way that the behavior of the block can no longer be reproduced when migrating to Release Y. Therefore, the ASM library in Release Y contains two blocks: the enhanced block `ASM_BLOCK` in version 5.0 and `ASM_BLOCK_4_0`, which is the former version and corresponds to the block `ASM_BLOCK` with the version 4.0.

When migrating the ASM model `ASM_Model`, the library link of the affected ASM block is changed to the former version, i.e., from `ASM_BLOCK` to `ASM_BLOCK_4_0`. This ensures that the simulation behavior of the ASM model is not changed after the migration. You can manually replace the block in the model with the current version afterwards, to benefit from new features and the further development. However, this might lead to a modified simulation behavior and test results.

## Cases Which Require Migration

<b>Introduction</b>	The following changes to ASM blocks may occur. For some of them, migration is required.
<b>New import</b>	A new import is connected to a default value. The default value is set so that the model behavior is similar to that before migration. To get the full advantage of new features, it might be necessary to connect the import with another signal.
<b>New outport</b>	A new outport is grounded or connected with other (new) imports.
<b>New parameter</b>	For a new parameter, a “pre-update” variant ( <code>_asmmigratepre</code> ) is created. Default values are set for the new parameter in this variant.
<b>Renamed parameter</b>	For a renamed parameter, a “post-update” variant ( <code>_asmmigratepost</code> ) is added. This variant includes the following equation:  <code>&lt;new parameter name&gt; = &lt;old parameter name&gt;</code>  The initialization values are transferred to the new parameter. The old parameter is not deleted and still exists in the workspace.
<b>Resized parameter</b>	For a resized parameter, a “post-update” variant ( <code>_asmmigratepost</code> ) is added. The postmigrate function makes sure that the size of the map after the initialization is consistent to the new size even if the Ini files have been generated with an old ModelDesk version. There are only maps whose size are increased. For example, the size of a map was increased from [20 x 20] to [30 x 30]. Then the postmigrate function checks the size of the map, and if it is not equal to the expected new size, it is increased by extrapolation:  <pre>Map_extrapolated [ 1:20,1:20 ] == Map_before_extrapolation</pre> The initialization values are kept.
<b>Removed parameter</b>	This case can be ignored as unused workspace parameters do not affect the model.
<b>Only changes inside of the library block</b>	If there are just changes inside of a library block, no additional migration is necessary.

## Steps Performed During a Migration

<b>Migration steps</b>	If you open an ASM model of a previous version, you have to confirm the start of the migration process. If you choose to migrate the model, the steps described below are executed. If you reject to migrate the model, it is opened without any changes. You can view the model, but it might not be possible to perform an update diagram or a simulation. The model can contain bad links, unconnected ports and undefined parameters.
------------------------	---

The following steps are performed during migration:

1. A backup of the model file is automatically created. This backup is in the same folder as the original model. If a migration over several Releases is done, only the original model is saved. Intermediate versions are not saved. Nevertheless, it is recommended to create a backup of the whole project before starting the migration process.
2. After the model is opened, the new ASM blocks are automatically linked to the model, because the new ASM libraries are in the installation.
3. At first, the model is parsed to find all ASM blocks. Then the model is modified (the inports and outports are connected) according to the blocks that are found in it.

### Note

Blocks with links to an ASM library that have been disabled or broken are not migrated. This can lead to malfunction, especially in the case of Operator version, because the block might contain references to incorrect S-functions. It is also not guaranteed that ModelDesk can handle such blocks correctly.

To ensure that a model can be migrated never disable or break links of ASM blocks.

Further the pre- and postmigration variants (`_asmmigratepre`; `_asmmigratepost`) are created. Inside these folders, files (`_asmmigratepre/lnifiles/go_addonpre.m`, `_asmmigratepost/lnifiles/go_addonpost.m`) are created. These contain the calls for these variants. If a `go.m` file is found, the variants are also automatically inserted into this `go.m` file after a backup (`go.m.bak`) was created. If you use files other than `go.m` to start the model, you must add the new variants manually. The pre-update variant (which contains default values for new parameters) must be initialized before any other variant. The post-update variant (which resizes and renames parameters) must be executed after all other variants. If your model already contains pre- and postmigrate variants from a previous migration, always place the new premigrate variant before the other premigrate variants and the new postmigrate variant after the other postmigrate variant.

4. To keep the changes in the migrated model, it has to be saved manually. After being saved, the model cannot be used with any previous ASM versions.

- Finally, the migration recalls the initialization process, including the generated pre- and postmigrate variant. Existing ModelDesk Plotting blocks are updated. In seldom cases, the recall of the initialization cannot be performed. Then a hyperlink similar to the following is created:

```
*** !!! Run >> asm_tool_rehash !!!
*** !!! Run >> go again !!!
*** !!! Run >> Update ModelDesk Plotting Block!!!
```

Click the hyperlink to perform the required steps of `asm_tool_rehash.m`, `go.m` and the ModelDesk update.

- After this an “update diagram” can be performed without error. The simulation can be started or code can be generated.

### Parameterizing in ModelDesk

If you parameterize your models using ModelDesk, note the following points:

- If ASM and ModelDesk are used, it is recommended to migrate the ASM model before you open the ModelDesk project with the new ModelDesk version.
- You must generate new real-time code after model migration and before opening ModelDesk. This only applies in ModelDesk projects which use the real-time model.
- When a ModelDesk project created with a previous Release is opened in ModelDesk, the project is migrated automatically.

### Related topics

#### Basics

[Migrating a Project and Its Experiments \(ModelDesk Project and Experiment Management\)](#)

## Migration Variants

### Basic concept

When an ASM project is migrated, it can happen that new parameters are introduced. A premigrate variant is therefore added. This variant has to be initialized before all other ASM parameters.

If the sizes of maps are changed, new parameters are calculated from old parameters, or parameters are renamed, a postmigration variant is initialized. This variant has to be called after all other ASM parameters.

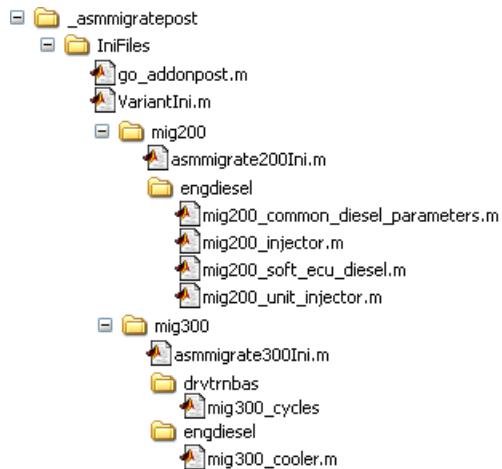
Premigration and postmigration variants have the same structure. The following topic gives an overview.

**Variant structure**

The following illustration shows an example of a simple postmigration variant structure.

**Note**

The structure was introduced with ASM 2.3. Since ASM 2.4 the file and folder names are shorter to avoid issues with long paths. If the model was already migrated with an older ASM version, it can contain migration variants with a different structure. A separate variant (`_asmmigratexxxpre` and `_asmmigratexxxpost`) was generated for every model. It is possible to use a mixture of old and new variants.



The generated initialization files are first sorted by model version (in this case `mig200` and `mig300`) and then by libraries (in this example `engdiesel` and `drvtrnbas`).

The call to `VariantIni.m` is added to the `go` file during migration. If you are using another file, you have to copy the contents of `go_addonpre.m` and `go_addonpost.m` to your initialization file.

`VariantIni.m` calls the `asmmigratexxxIni.m` file. This file calls the real initialization files (for example, `mig200_common_dieselparameters`, `mig200_injector`) which contain the information about the parameter adaptations.

The first time you migrate an ASM model, the whole structure (including `VariantIni.m` and `go_adonxxx.m`) is created. When you migrate a model which was already migrated to a previous ASM version, the new initialization files and the new `asmmigratexxxIni.m` are created. The calls to the new `asmmigratexxxIni.m` are also added to the existing `VariantIni.m`.

## Project Consistency

### Introduction

The following steps are recommended to keep the initialization process of the model consistent. The aim is to remove all pre-update and post-update initialization files in order to keep all parameter information only in the ModelDesk project.

### Workflow

The pre-update and post-update variants are initialization files used to introduce new parameters or change parameter attributes, such as their names. If no parameters were introduced or no operations were performed on the parameters during the migration, the pre-update and post-update variants are not required. For more information on the premigrate and postmigrate variants, refer to [Steps Performed During a Migration](#) on page 72.

After migrating the ModelDesk project you must generate new initialization files for all parameter sets. Afterwards, the pre-update and post-update variants can be removed and the backup go file (go.m.bak) can be restored.

## How to Migrate Blocks of Custom Component Libraries

### Introduction

To use the multi-instance feature for blocks of custom component libraries, a migration is necessary.

### Basics

As of dSPACE Release 2018-B, blocks of a ModelDesk custom library support the multi-instance feature. Refer to [Multi-Instance](#) on page 53.

To use this feature, the custom library and the model that includes blocks of this library must be migrated.

#### Note

It is recommended to migrate the custom library and the model. Without migration, the blocks of this library are only supported without multi-instance features.

### Migrating process

You must migrate the custom library and the models that contain blocks of the library.

- To migrate the custom library, refer to [Part 1](#) on page 76.
- To migrate the model, refer to [Part 2](#) on page 76.

**Part 1**

**To migrate custom libraries**

- 1 Start ModelDesk but do not load a project.
- 2 Register the custom library. For instructions, refer to [How to Register a Custom Library \(ModelDesk Parameterizing\)](#).  
ModelDesk prepares the blocks in the library for using the multi-instance feature.

---

**Part 2**

**To migrate models containing blocks of a custom library**

- 1 In MATLAB, open the model that includes blocks of the custom library.
- 2 Open the MDLD\_Utils.lib library.
- 3 Place the Custom Component Interface/ Custom Component Migration block in the model.
- 4 Double-click the Custom Component Migration block.  
An overview opens that shows all the custom component blocks that are not migrated and included in the active Simulink system.
- 5 Click Migrate to start the migration.

---

**Result**

When the library and the blocks in the model are migrated, you can use the multi-instance feature. If the model is migrated successfully, the Custom Component Migration block in the model can be deleted.

---

**Related topics**

**Basics**

[Integrating and Using Custom Libraries \(ModelDesk Parameterizing\)](#)

# Tools

## Where to go from here

## Information in this section

<a href="#">ASM Engine Testbench</a> .....	78
ASM Engine Testbench is a tool for testing the parameterization. The simulation results can be compared to measurements.	
<a href="#">ASM RoadConverter</a> .....	93
ASM RoadConverter is a MATLAB-based tool to create segment-based ASM roads from measured road data.	
<a href="#">ASM Controller Adaption</a> .....	107
ASM Controller Adaption is a tool to tune a controller for a known plant by three different approaches.	
<a href="#">Adams2ASM Converter</a> .....	128
The Adams2ASM Converter lets you convert data from an Adams Car™ model for use with ModelDesk.	
<a href="#">ASM LabelInterface</a> .....	137
The ASM LabelInterface can convert buses to vectors and vice versa. Labels inside a bus can be changed.	
<a href="#">ASM Utils</a> .....	153
The ASM Utils library serves as a basis for common functionalities in ASM models.	

# ASM Engine Testbench

## Where to go from here

## Information in this section

[Basics of ASM Engine Testbench](#)..... 78

ASM Engine Testbench is a tool for testing the parameterization of engine models. The simulation results can be compared to measurements.

[User interface of ASM Engine Testbench](#)..... 79

The user interface of ASM Engine Testbench consists of four pages.

[Working with ASM Engine Testbench](#)..... 90

To work with an experiment in ASM Engine Testbench.

## Basics of ASM Engine Testbench

## Basic Concepts of ASM Engine Testbench

### Introduction

ASM Engine Testbench is a tool for testing the parameterization of engine models. You can use it to compare simulation results to measurements.

### Simulation results

ASM Engine Testbench compares simulation results to measurements. You can also compare different simulation results. The results of each simulation run are saved and listed in a table so that different simulation runs can be selected for plotting.

### Plotsets

You can define plotset for viewing different signals. Each plotset contains a set of plots that are shown when plotting is started.

### Plots

You can define different plots in which different simulation results are shown. You can make multiple settings for each plot.

### Stimulus data

Stimulus data defines how the model is stimulated in test bench simulation. Stimulus data is based on the operating point concept. Thus, you can select different values for each stimulus data for each operating point. ASM Engine

Testbench lets you define two types of stimulus data: MDL and CPT. If CPT is used, the values are written to a time dependent look-up table which changes the output values after each operating point. The CPT path must therefore contain the values .x and .v. If MDL is used, the values are written directly to the specified MDL variable (.v) on the workspace. If the stimulus data contains one MDL type, the simulation is started for each operating point separately. If you use CPT types only, one simulation is started in which all operating points are simulated.

## User interface of ASM Engine Testbench

### Introduction

The user interface of ASM Engine Testbench consists of four pages.

### Where to go from here

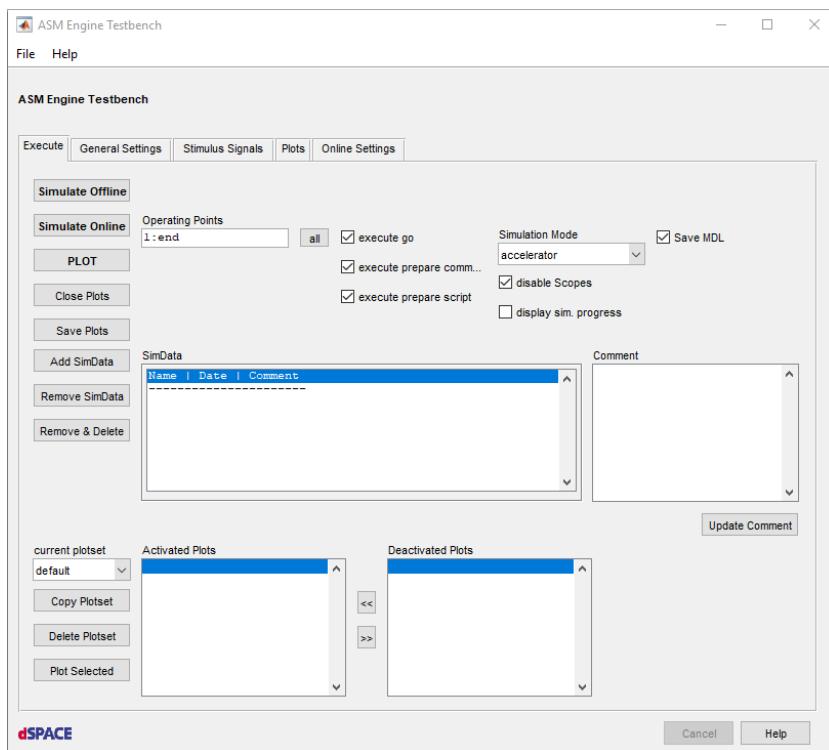
### Information in this section

<a href="#">Execute Page</a> .....	79
The test bench simulation can be started and the plots can be viewed.	
<a href="#">General Settings Page</a> .....	82
The general settings for the test bench simulation project can be made.	
<a href="#">Stimulus Signals Page</a> .....	83
Stimulus signals are used to operate the engine model in the desired state in test bench simulation.	
<a href="#">Plots Page</a> .....	85
The plots are defined on this page.	
<a href="#">Online Settings Page</a> .....	87
The settings for the online execution can be specified on this page.	

## Execute Page

### Description

The test bench simulation can be started and the plots can be viewed on the Execute page. Additionally, the results of the simulation runs and the plot sets can be organized and simulation settings can be made.



## Dialog settings

The page has the following dialog elements.

**Simulate offline** Starts the test bench simulation with the current settings in Simulink. When you click **Simulate offline**, a dialog opens to specify the name of the simulation run. When simulation has finished, the result plots are opened automatically.

**Simulate online** Starts the test bench simulation on the specified dSPACE Platform with the current settings (Refer to [Online Settings Page](#) on page 87). When you click **Simulate online**, a dialog opens to specify the name of the simulation run. When simulation has finished, the result plots are opened automatically.

**PLOT** Opens the result plots for the selected simulation run in the SimData table.

**Close Plots** Closes all result plots.

**Save Plots** Saves all open result plots. A folder must be specified where the plots should be saved. The plots are saved in the PNG and FIG file formats.

**Operating Points** Lets you specify the operating points that will be simulated in the next simulation run. For simulating all operating points either "1:end" or "100%" is valid. For simulating only a subset of operating points either a list of operating point numbers (for example, "1,2,3", "1:5", or "1:3,5:6") or a

percent specification (for example, "10%" for simulating the first 10% of all operating points) is allowed.

**Comment** Lets you specify a comment that will be shown in the SimData table to identify the simulation run.

**execute go** Indicates whether go should be executed before simulation to setup the MATLAB workspace for simulation.

**execute prepare cmd** Indicates whether the **prepare** command should be executed before simulation to setup the MATLAB workspace for simulation. The **prepare** command can be specified on the **General Settings** page.

**execute prepare script** Indicates whether the **prepare** script should be executed before simulation to setup the MATLAB workspace for simulation. The **prepare** script can be specified on the **General Settings** page.

**Simulation Mode** Lets you choose the simulation mode for the model (Accelerator or Normal, Accelerator is recommended for extensive simulations).

**disable Scopes** Indicates whether the scopes in the Simulink model should be disabled for simulation (recommended to achieve faster simulation).

**display sim. progress** Indicates whether a subsystem and two displays are inserted at the root level of the model. The displays show the current simulation time and the percentile progress.

**Save MDL** Indicates whether the MDL structure should be saved in the simulation results.

**SimData** Lists names, data and comments of the simulation results. In this table the simulation results can be organized. For plotting one or more rows can be selected.

**Comment** Shows the comment of the currently selected simulation result. The comment can be modified and then save by **Update Comment** button.

**Update Comment** Applies the text of the comment field to the currently selected simulation result.

**Add SimData** Adds SimData to the table. The simulation results can be selected in a file selection dialog.

**Remove SimData** Removes the selected SimData from the table but does not delete the simulation results file.

**Remove & Delete** Removes the selected SimData from the table and deletes the simulation results file.

**Activated Plots** Lists all active plots of the selected plotset. Only active plots will be plotted. You can switch a plot from active to inactive by using >> and <<.

**Deactivated Plots** Lists all inactive plots of the selected plotset.

**current plotset** Lets you choose the plotset.

**Copy Plotset** Copies the current plotset. You can specify a name for the new plotset.

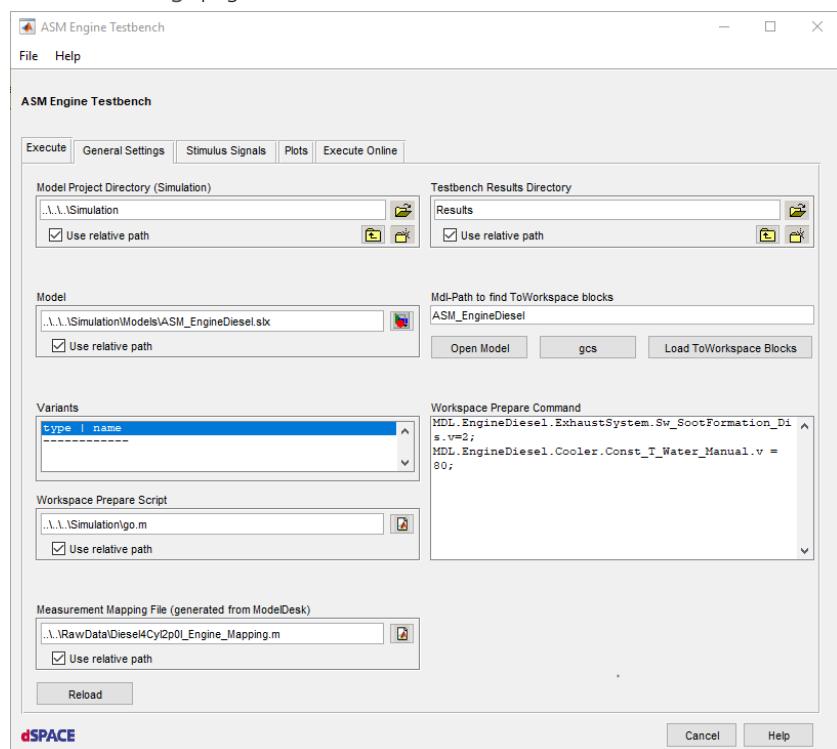
**Delete Plotset** Deletes the current plotset.

**Plot Selected** Plots only the signals selected in Active Plots.

## General Settings Page

### Description

The general settings for the test bench simulation project can be made on the General Settings page.



---

**Dialog settings**

The page has the following dialog elements.

**Model Project Directory** Lets you specify the **Simulation** folder of your model.

**Test bench Results Directory** Lets you specify the folder where the simulation results are saved.

**Model** Lets you specify the model that will be simulated.

**Mdl-Path to TestBlock** Lets you specify the path in the model to the test bench blocks. All To Workspace blocks that are contained in this system or in a subsystem are found in model parsing (Load Test bench Blocks).

**Open Model** Opens the model.

**gcs** Adds the path of the current open system of the model to the **Mdl-Path to TestBlock** edit field.

**Load Test bench Blocks** Parses the model and loads all test bench blocks in the model under the specified path to the User interface. You can select these blocks when you define plots.

**Variants** Lets you specify the variants called in **go**. You can define a type and the variant name and also handle multiple variants. Variants can be edited by using the context menu of the table.

**Workspace Prepare Script** Lets you specify a script which is called after **go** is executed. The script is executed in the simulation workspace. You can use it to change parameters, for example.

**Workspace Prepare Command** Lets you specify a command which is executed after **go** and the prepare script are executed. The command is executed in the simulation workspace. You can use it to change parameters, for example.

**Measurement Mapping File** Lets you specify a file which contains all available measurements. This file must normally be created by ModelDesk Processing.

**Reload** Loads the measurements to the User interface. You can select the measurements when you define stimulus signals and plots. Before stimulus data and plots are defined, this function must have been executed.

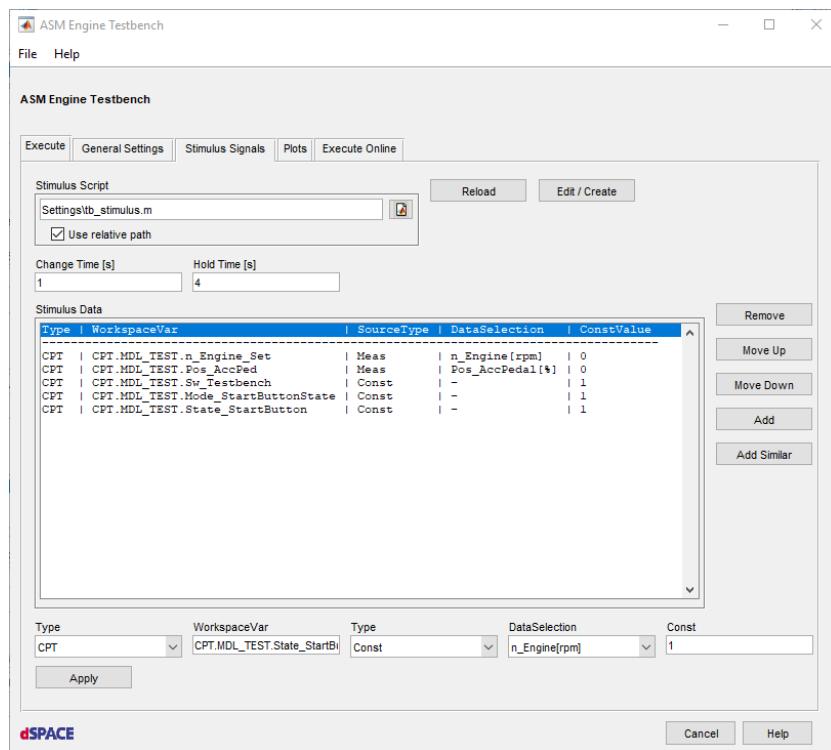
---

## Stimulus Signals Page

---

**Description**

Stimulus signals are used to operate the engine model in the desired state in test bench simulation, for example to hold the engine in the defined operating point for some time. The stimulus signals can be defined for the current experiment on the **Stimulus Signals** page. Stimulus signals can be defined by constant values, measurements or via a script.



## Dialog settings

The page has the following dialog elements.

**Stimulus Script** Lets you specify the M file where the stimulus script can be found. In this script stimulus signals can be defined. For detailed information on inputs and outputs, see the header of created M file.

**Reload** Loads the stimulus signals from the stimulus script to the User interface.

**Edit / Create** Opens the script in the editor. If the script does not exist, a new files is created.

**Change Time** Lets you specify the time that is used for changing between the operating points.

**Hold Time** Let you specify the time that is used for holding on operating point in a steady state.

**Stimulus Data** Lists the stimulus signals. To manipulate the list, use Remove, Move Up, Move Down, Add, Add Similar and Apply buttons. The table has the following fields.

Field	Description
Type	Type of the stimulus data (CPT or MDL)
WorkspaceVar	Path in the workspace where the value is written to
SourceType	Type of the data source of the stimulus data: ▪ Meas: Measurement

Field	Description
	<ul style="list-style-type: none"> <li>▪ Const: Constant value</li> <li>▪ Script: Script</li> </ul>
DataSelection	Selected data if measurement or stimulus script data is used.
ConstValue	Constant value if SourceType is Const.

**Remove** Deletes the selected entry of the Stimulus Data list.

**Move Up, Move Down** Moves the selected entry up or down in the Stimulus Data list.

**Add** Adds a new, empty entry in the Stimulus Data list.

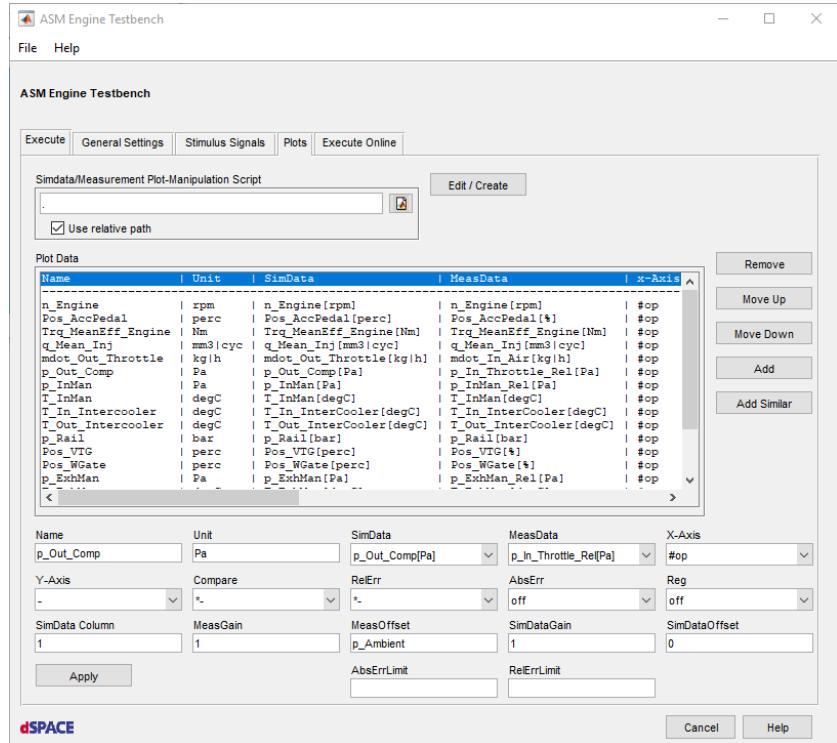
**Add Similar** Adds a new entry in the Stimulus Data list and copies the data of the selected entry to the new one.

**Apply** Applies the values specified in the edit fields and writes them to the Stimulus Data list.

## Plots Page

### Description

The plots are defined on the Plots page.



**Dialog settings**

The page has the following dialog elements.

**Simdata/Meas Manipulation Script** Lets you specify a script which manipulates the simulated and measurement data for plotting. This can be used for unit conversion, for example. For detailed information on inputs and outputs, see the header of the created M file.

**Edit/Create** Opens the script in the editor. If the script does not exist, a new file is created.

**Plot Data** Lists the plots which are defined. To manipulate the table data, use the Remove, Move Up, Move Down, Add, Add Similar and Apply buttons. The table has the following fields:

Field	Description
Name	Name of the plotted signal
Unit	Unit of the plotted signal
SimData	Simulated data used in the plot
MeasData	Measured data used in the plot
X-Axis	The x-axis of the plot. You can select measurement data or the number of the operating points.
Y-Axis	The y-axis of the plot. It must be specified only if a 3-D plot is desired.
Compare	Indicates whether and how the compare plot should be plotted
RelErr	Indicates whether and how the relative error plot should be plotted
AbsErr	Indicates whether and how the absolute error plot should be plotted
Compare	Indicates whether and how the regression plot should be plotted
SimData Column	Specifies the column of the plotted data in the ToWorkspace struct (in most cases 1) for multidimensional signals
MeasGain	Gain for the measurement data. It can be used for unit conversion, for example.
MeasOffset	Offset for the measurement data. It can be used for relative – absolute pressure correction, for example.
SimDataGain	Gain for the simulated data. It can be used for unit conversion, for example.

Field	Description
SimDataOffset	Offset for the simulated data. It can be used for relative – absolute pressure correction, for example.

**Remove** Deletes the selected entry of the Plot Data list.

**Move Up, Move Down** Moves the selected entry up or down in the Plot Data list.

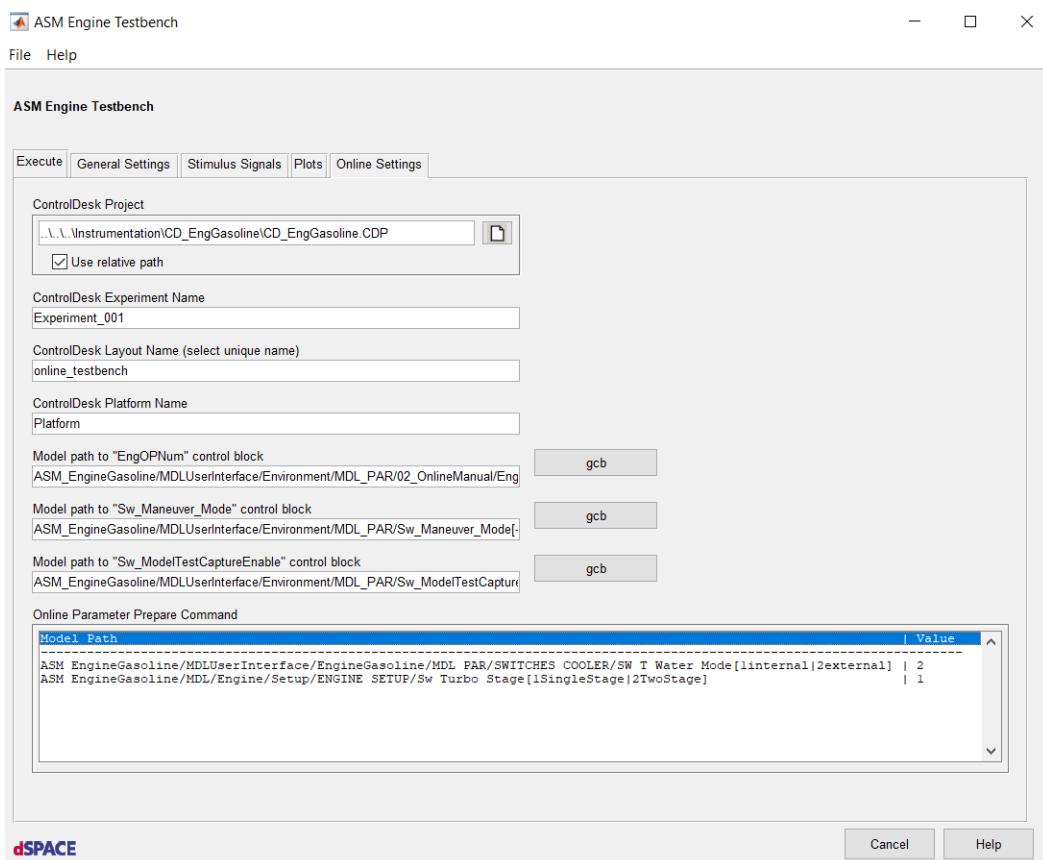
**Add** Adds a new, empty entry in the Plot Data list.

**Add Similar** Adds a new entry in the Plot Data list and copies the data of the selected entry to the new one.

**Apply** Applies the values specified in the edit fields and writes them to the Plot Data list.

## Online Settings Page

<b>Description</b>	The settings for the online simulation on a platform are specified on the Online Settings page. The online simulation is also started on this page in contrast to the Simulink simulation, which is started on the Execute page.
--------------------	--



## Dialog settings

The page has the following dialog elements.

**ControlDesk Project** Lets you specify the folder of the ControlDesk project.

**ControlDesk Experiment Name** Lets you specify the name of the ControlDesk experiment. The experiment must exist before you start the online simulation.

**ControlDesk Layout Name** Lets you specify a name for the new layout generated by the ASM Testbench. The name must be unique in the ControlDesk experiment.

**ControlDesk Platform Name** Lets you specify the platform on which the application is running. The platform has to be registered before you start the online simulation.

**Model path to "EngOPNum" control block** Lets you specify the path to the control block in the model.

**gcb** Lets you add the path of the selected block to corresponding edit field.

**Model path to "Sw\_Maneuver\_Mode" control block** Lets you specify the path to the control block of the maneuver mode switch in the model.

**Model path to "Sw\_ModelTestCaptureEnable" control block** Lets you specify the path to the control block of the enable switch for capturing the simulation results in the model.

**Online Parameter Prepare Command** Lets you define the exact model path of parameters set when the online simulation starts.

When you right-click in the field, the context menu presents you the following commands:

- Add: Create a new entry.
- Add GCB: Create a new entry with the path from the block selected in Simulink as default.
- Modify: Modify the selected entry.
- Remove: Remove the selected entry.

#### Note

Only constant blocks can be defined in this way.

The modified parameters are not reverted on the running platform. You have to do this manually.

#### Remarks

In order to validate the results of the application that runs on the online platform, *online simulation* lets you compare the results with the measurement data.

The specific engine operating points are set online via ControlDesk and the measurement variables are also read via ControlDesk.

#### Note

There is a difference between online and offline execution of the test bench simulation:

- In an offline simulation, i.e., a simulation via Simulink, you can select the control signals on the Stimulus page. The measurement data of these control signals is written to tables in a stimulus maneuver. The control signals correspond to the engine operating points that are selected on the execute page. The maneuver is started and the measurement signals are stored in the MATLAB workspace for each step in the table.
- In an online simulation, i.e., a simulation on the platform, you can select the engine operating points on the Execute page. Then, you start the online test bench on the Execute Online page. Now, each operating point is set via ControlDesk and the measurement signals are read via ControlDesk after a defined hold time.

This difference results from the non-existent synchronization and common time base between the online simulation on the platform and the test bench script.

# Working with ASM Engine Testbench

## Where to go from here

## Information in this section

<a href="#">How to Load an Experiment.....</a>	90
Before you can start working with the ASM Engine Testbench your must start the tool and load an experiment.	
<a href="#">How to View Simulation Results.....</a>	91
You can view the simulation results.	
<a href="#">How to Test an Engine Parameterization.....</a>	91
You can test your engine parameterization in a new simulation.	
<a href="#">How to Setup a New Test Bench Project.....</a>	92
You can setup a new project in the ASM Engine Testbench.	

## How to Load an Experiment

### Objective

Before you can start working with the ASM Engine Testbench your must start the tool and load an experiment.

### Method

#### To load an experiment

- 1 In MATLAB, change to the \Parameterization\ASMPProcessing\Eng\_Testbench folder of your ASM Engine project.
- 2 To start ASM Engine Testbench run the MATLAB file  
`go_asm_eng_testbench.m`
- 3 In the ASM Engine Testbench, click File - Open, change to the Settings folder and select the MAT file.

### Result

The experiment is loaded.

You can now view the simulation results and test your engine parameterization in a new simulation.

### Related topics

#### HowTos

<a href="#">How to Test an Engine Parameterization.....</a>	91
<a href="#">How to View Simulation Results.....</a>	91

## How to View Simulation Results

<b>Objective</b>	You can view the simulation results.		
<b>Precondition</b>	The experiment must be loaded, see <a href="#">How to Load an Experiment</a> on page 90.		
<b>Method</b>	<b>To view simulation results</b> <ol style="list-style-type: none"><li>1 In ASM Engine Testbench open the Execute page.</li><li>2 Select simulated data in the SimData table.</li><li>3 Select the desired plots in the current plotset by moving the plots from Deactivated Plots to Activated Plots or vice versa.</li><li>4 Click PLOT to start plotting.</li></ol>		
<b>Related topics</b>	<b>References</b>  <table><tr><td><a href="#">Execute Page</a>.....</td><td>79</td></tr></table>	<a href="#">Execute Page</a> .....	79
<a href="#">Execute Page</a> .....	79		

## How to Test an Engine Parameterization

<b>Objective</b>	You can test your engine parameterization in a new simulation.		
<b>Precondition</b>	The experiment must be loaded, see <a href="#">How to Load an Experiment</a> on page 90.		
<b>Method</b>	<b>To test an engine parameterization</b> <ol style="list-style-type: none"><li>1 On the Execute page, select the operating points that should be simulated in the Operating Points field.</li><li>2 Click SIMULATE to start the test bench simulation.</li></ol>		
<b>Related topics</b>	<b>References</b>  <table><tr><td><a href="#">Execute Page</a>.....</td><td>79</td></tr></table>	<a href="#">Execute Page</a> .....	79
<a href="#">Execute Page</a> .....	79		

## How to Setup a New Test Bench Project

**Objective**

You can setup a new project in the ASM Engine Testbench.

**Precondition**

ASM Engine Testbench must be started, see [How to Load an Experiment](#) on page 90.

**Method****To setup a new test bench project**

- 1 In ASM Engine Testbench, open the General Settings page.
- 2 Select your model directory.
- 3 Select your model.
- 4 Select the model path where the To Workspace blocks for test bench simulation can be found. Then load test bench blocks using Load Test bench Blocks.
- 5 Select your measurement file and click Reload to load measurements.
- 6 Open the Stimulus Signals page.
- 7 Add your stimulus signals for test bench simulation.
- 8 Open the Plots page.
- 9 Add or define your plots.
- 10 Open the Execute page.
- 11 Click SIMULATE to start simulation.

**Related topics****References**

Execute Page.....	79
General Settings Page.....	82
Plots Page.....	85
Stimulus Signals Page.....	83

# ASM RoadConverter

## Where to go from here

## Information in this section

<a href="#">Introduction to ASM RoadConverter.....</a>	93
ASM RoadConverter takes road measurement data and creates a MAT file to be imported to ModelDesk.	
<a href="#">User interface of ASM RoadConverter.....</a>	97
The user interface consists of a menu bar, a navigation area and some pages.	
<a href="#">Working with ASM RoadConverter.....</a>	100
To work with a project in ASM RoadConverter.	

# Introduction to ASM RoadConverter

## Where to go from here

## Information in this section

<a href="#">Basics on ASM RoadConverter.....</a>	93
ASM RoadConverter takes road measurement data and creates a MAT file to be imported to ModelDesk.	
<a href="#">Functional Description.....</a>	94
The road conversion is performed in two main steps – mapping and conversion.	
<a href="#">Folder Structure of ASM RoadConverter.....</a>	95
ASM RoadConverter stores its data in the project folder.	
<a href="#">Data Formats.....</a>	95
In ASM RoadConverter, different file formats are used.	

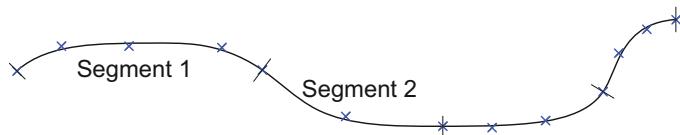
# Basics on ASM RoadConverter

## Introduction

ASM RoadConverter takes road measurement data and creates a MAT file to be imported to ModelDesk.

ASM RoadConverter fits splines to the road data. The start and end points of a segment match the specified raw data points exactly. The maximum deviation between the spline curve and the data points can be specified in the user interface.

The following illustration shows an example of raw data points and the resulting road curve.



## Related topics

### Basics

<a href="#">Functional Description</a>	94
--	----

## Functional Description

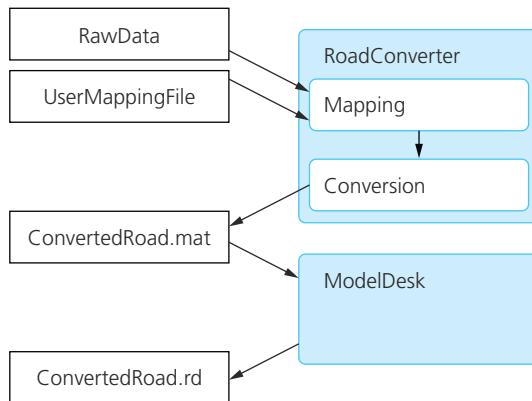
### Introduction

The road conversion is performed in two main steps – mapping and conversion.

### Road conversion

The mapping is performed to map the user road description to the RoadConverter internal format. Unit conversions and data analysis are performed here.

The conversion calculates the road segments and creates a converted road MAT file. This MAT file is imported to ModelDesk. ModelDesk stores the road in the ModelDesk `rd` format.



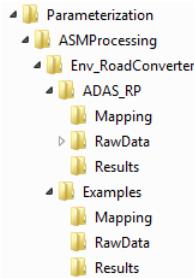
**Related topics****Basics**

Basics on ASM RoadConverter.....	93
----------------------------------	----

## Folder Structure of ASM RoadConverter

**Example projects**

The ASM VehicleDynamics demo projects contain example road conversion projects. The projects are located in <ASM project>\Parameterization\ASMPprocessing\Env\_RoadConverter.



The **Examples** folder contains two road conversion projects: one with road measurements in GPS format and one with xy coordinates. The **ADAS\_RP** folder contains an example of how to convert roads which are exported by ADAS RP.

Each road conversion project folder contains the **Mapping**, **RawData** and **Results** folders. Road measurement data is stored in the **RawData** folder. The raw data is usually available with user-specific names and is stored in M or MAT files. The user-mapping files are located in the **Mapping** folder. The mapping files are used to map the specific variable names to the required RoadConverter format.

## Data Formats

**Introduction**

In ASM RoadConverter, different file formats are used.

**RawData files**

The ASM RoadConverter can be used to convert roads from xy data or from GPS data. In both cases raw data is provided in MAT or M files. Data must be available in vectors of same length. The variable names in the RawData files can be chosen by the customer.

**xy data format** In case of xy data the following information is needed:

- x coordinate in [m]
- y coordinate in [m]
- z coordinate in [m] (optional, may be zero)
- Lateral slope angle (roll angle) in [deg] (optional, may be zero)

**GPS data format** In case of GPS data the following information is needed:

- Latitude in [deg]
- Longitude in [deg]
- z coordinate in [m] (optional, may be zero)
- Lateral slope angle (roll angle) in [deg] (optional, may be zero)

#### RoadConverter data format

The data is provided in the **RawData** folder in M files which define the variables or in MAT files in which the data is stored.

The ASM RoadConverter itself uses a matrix (**roaddata**) which contains the xy data or the GPS data. The mapping of raw data to this matrix is done in the mapping file.

**xy data** In case of xy data the mapping file must contain the information about the used data format and the data itself.

```
settings.method = 'xy';
..
roaddata(:,1) = <insert your mapping here> % x coordinate [m]
roaddata(:,2) = <insert your mapping here> % y coordinate [m]
roaddata(:,3) = <insert your mapping here> % z coordinate [m] (optional)
roaddata(:,4) = <insert your mapping here> % lateral slope angle [deg] (optional)
```

**GPS data** In case of GPS data the following information is needed in the mapping file.

```
settings.method = 'GPS';
..
roaddata(:,1) = <insert your mapping here> % Latitude [deg]
roaddata(:,2) = <insert your mapping here> % Longitude [deg]
roaddata(:,3) = <insert your mapping here> % z coordinate [m]
roaddata(:,4) = <insert your mapping here> % lateral slope angle [deg] (optional)
```

Decimation of data or unit conversions can be performed in the mapping file as well.

# User interface of ASM RoadConverter

## User Interface of ASM RoadConverter

### Introduction

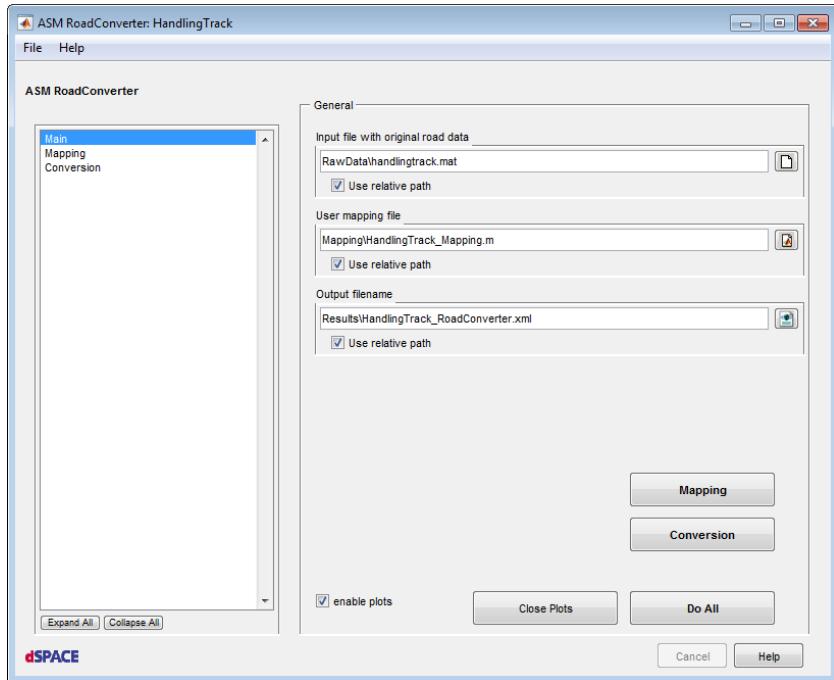
The user interface consists of a menu bar, a navigation area on the left and the pages on the right.

### Overview

The File entry in the menu bar can be used to save and load road conversion projects. The navigation area on the left can be used to switch between pages. After startup the Main page is active.

### Main page

The Main page is used to choose files and access the functionality via buttons.



**Input file with original road data** Lets you specify the file which contains the measured road data in customer format. To select a file, click the button to the right of the edit field. To open the file, right-click the button to open its context menu and choose Open.

**User mapping file** Lets you specify the mapping file which is used to map the raw data to the road converter format. To select a file, click the button to the

right of the edit field. To open the file, right-click the button to open its context menu and choose Open.

**Output file name** Lets you specify the file name of the MAT file to be created. To select a file, click the button to the right of the edit field. To open the file, right-click the button to open its context menu and choose Open.

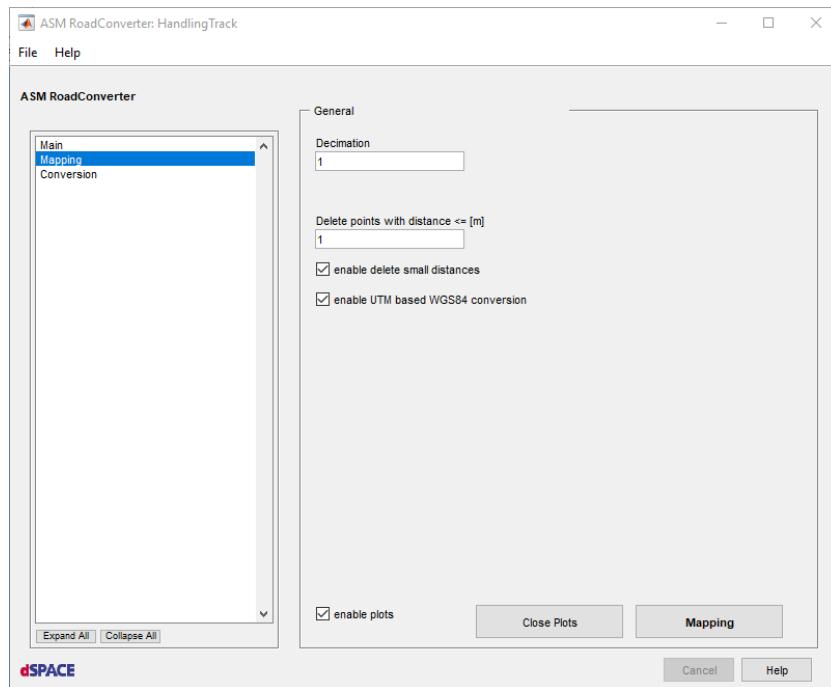
**Mapping** Starts the mapping process.

**Conversion** Creates the converted road.mat file to be imported in ModelDesk.

**Do All** Runs the complete conversion.

## Mapping page

During mapping, the road raw data is mapped to the road converter's internal format and a data analysis is performed. First the user mapping file is executed. The raw data is read and converted to the road converter's internal format. The user mapping file can also contain plots and printouts. Unit conversions are also performed during mapping. After user mapping, a general data analysis and some preprocessing steps are performed. All information is logged to the MATLAB Command Window.



The Mapping page contains main parameters for data preprocessing.

**Decimation** Lets you specify a value for decimation. Only every <x> measurement point is used.

**Delete points with distance <=[m]** Lets you specify a distance.

**Enable delete small distance** If selected, only data points with a distance larger than the value specified are used for conversion.

**Enable UTM based WGS84 conversion** Lets you specify whether to enable UTM-based WGS84 conversion of given GPS coordinates into x-,y- coordinates or enable *simplified spherical conversion*.

**Mapping** Starts the mapping process.

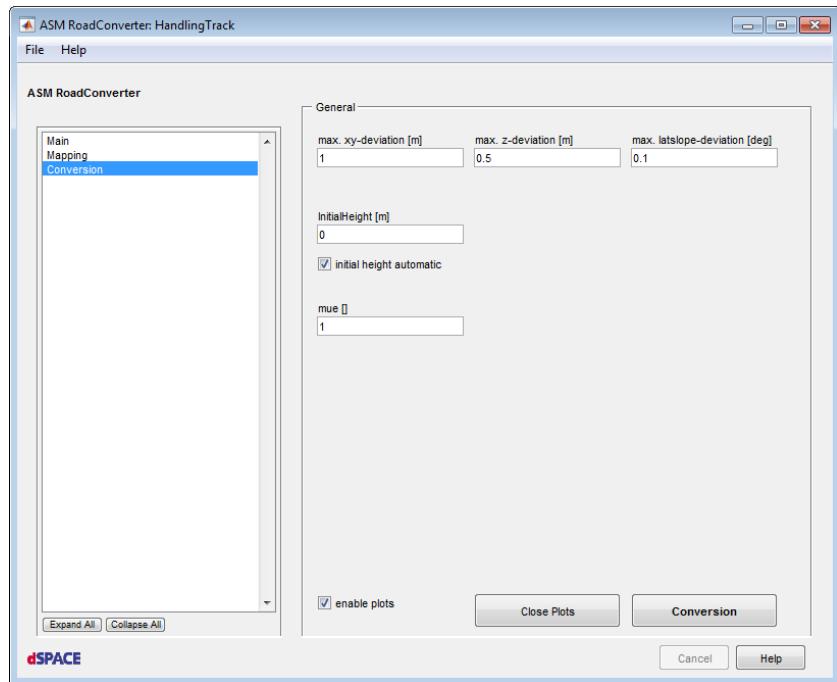
### Conversion page

During conversion, the ASM RoadConverter segments are calculated and the road MAT file to be imported to ModelDesk is created.

Conversion is performed in three steps:

1. The road segments are fitted to the xy geometry.
2. The road height profile is converted.
3. The road's lateral slope is converted.

All information is logged to the MATLAB Command Window.



The Conversion page provides access to conversion parameters.

**max. xy-deviation [m]** Lets you specify the accepted distance error between the measured data points and calculated road.

**max. z-deviation [m]** Lets you specify the accepted height error between the measured data and calculated road.

**max. latslope-deviation [deg]** Lets you specify the accepted lateral slope error between the measured data and calculated road.

**Initial height [m]** Lets you specify the initial height.

**initial height automatic** Indicates whether the initial height is calculated automatically. If it is selected, the initial height is calculated so that the road always has a positive height. If it is cleared, the value specified in Initial height [m] is used.

**mue []** Lets you specify the default value for  $\mu$  in the created road.

**Conversion** Starts the conversion.

## Working with ASM RoadConverter

### Where to go from here

### Information in this section

<a href="#">How to Start ASM RoadConverter</a> .....	101
ASM RoadConverter must be started within MATLAB.	
<a href="#">How to Manage Projects with ASM RoadConverter</a> .....	101
Working with ASM RoadConverter means editing and managing (conversion) projects.	
<a href="#">How to Run Existing RoadConverter Projects</a> .....	102
Running an existing road conversion project includes the following steps.	
<a href="#">How to Import a Road in ModelDesk</a> .....	102
The road XML file must be imported to the Pool in ModelDesk.	
<a href="#">How to Create a New Road Conversion Project</a> .....	104
To create new conversion projects, you should use an example road conversion project as a template.	
<a href="#">How to Convert Roads from ADAS RP</a> .....	105
To convert roads, which are exported from ADAS RP, use the example project.	
<a href="#">Tips &amp; Tricks</a> .....	106
Provides tips and tricks for work with road models.	

## How to Start ASM RoadConverter

<b>Objective</b>	ASM RoadConverter must be started within MATLAB.
<b>Method</b>	<b>To start ASM RoadConverter</b> <ol style="list-style-type: none"><li>1 Open MATLAB</li><li>2 Open the road conversion project folder, for example, &lt;work&gt;\Parameterization\ASMPprocessing\Env_RoadConverter\Examples</li><li>3 In the MATLAB Command Window, type go_roadconverter.</li></ol>

## How to Manage Projects with ASM RoadConverter

<b>Objective</b>	Working with ASM RoadConverter means editing and managing road conversion projects.		
<b>Project file</b>	A road conversion project file contains all the settings to convert a road. It can be opened with the ASM RoadConverter. Road conversion project files are located in the root of the road conversion project folder. For example, you can find the <code>HandlingTrack.mat</code> and <code>GPS_Example.mat</code> in <ASMPproject>\Parameterization\ASMPprocessing\Env_RoadConverter\Examples.		
<b>Method 1</b>	<b>To open a project</b> <ol style="list-style-type: none"><li>1 Select File – Open to load an existing project file.</li></ol>		
<b>Method 2</b>	<b>To save a project</b> <ol style="list-style-type: none"><li>1 Select File – Save or Save as to save the current project file.</li></ol>		
<b>Related topics</b>	<b>References</b>  <table><tr><td>User Interface of ASM RoadConverter.....</td><td>97</td></tr></table>	User Interface of ASM RoadConverter.....	97
User Interface of ASM RoadConverter.....	97		

## How to Run Existing RoadConverter Projects

**Objective** Running an existing road conversion project includes the following steps.

Method	To run existing RoadConverter projects
	<ol style="list-style-type: none"><li>1 In MATLAB, change to the road conversion project folder: <code>cd &lt;project_folder&gt;</code></li><li>2 To open the road converter, type <code>go_roadconverter</code> The RoadConverter opens.</li><li>3 To load a different road conversion project file, choose <b>File - Open</b> in the menu bar.</li><li>4 Click <b>Do All</b> on the Main page.</li><li>5 To optimize the conversion, open the other pages, change parameters, and rerun the conversion.</li></ol>

**Result** The conversion result is stored in a MAT file in the **Result** folder. You have to import this MAT file to the ModelDesk pool, [How to Import a Road in ModelDesk](#) on page 102.

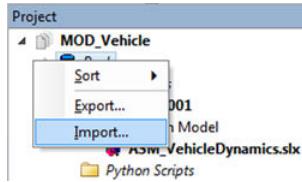
Related topics	References
	<p>User Interface of ASM RoadConverter.....97</p>

## How to Import a Road in ModelDesk

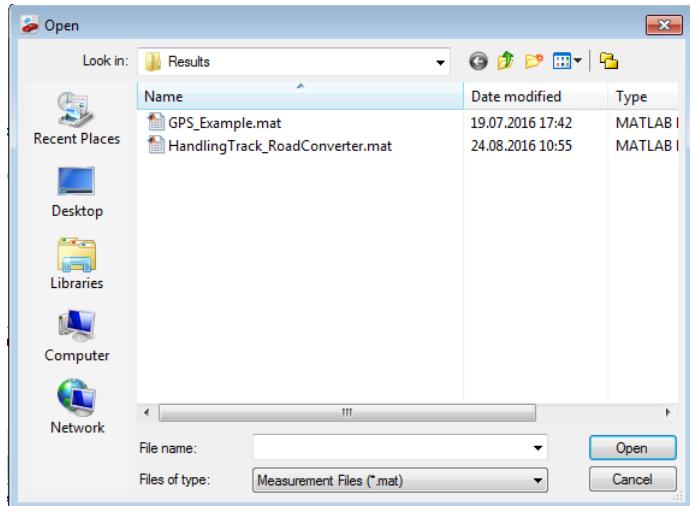
**Objective** The created road MAT file must be imported to the Pool in ModelDesk. Perform the following steps for road import.

**Method****To import a road in ModelDesk**

- 1 Open a ModelDesk project and experiment.
- 2 To import the MAT file to the Pool, choose Import from the context menu on the Pool node in the Project Navigator.



- 3 In the Open dialog, select the MAT file.



- 4 Confirm the file conversion dialog.
- 5 Save the project and experiment.

**Result**

The road file is converted to the .rd format and is imported to the ModelDesk Pool. You can now use the road in ModelDesk.

**Related topics****References**

[Import \(ModelDesk Basics\)](#)

## How to Create a New Road Conversion Project

### Objective

To create new road conversion projects, you should use an example road conversion project as a template.

#### Note

To convert GPS road data, use the `GPS_Example.mat` and `GPS_Example_Mapping.m` files.

Here, the process of road conversion is shown with a specific example.

### Method

#### To create a new road conversion project

- 1** Copy and rename the `..\Parameterization\ASMPProcessing\Env_RoadConverter\Examples` folder to `<Env_RoadConverter\MyRoad>`.
- 2** In MATLAB, browse to the new folder and open the road converter using `go_roadconverter`.
- 3** Provide the road measurement data in M or MAT format in the `RawData` folder.

As an example create a new M file

`<\Env_RoadConverter\MyRoads\RawData\MyRoad.m>` and copy the following artificial road data to this file.

```
Road_x = [0:200]; % x [m]
Road_y = sin(Road_x/15)*20; % y [m]
Road_z = (1-cos(Road_x/32))*5; % z [m]
```

The lateral slope is not used in this example.

- 4** In the ASM RoadConverter dialog on the Main page, select the new raw data file under Input file with original road data.
- 5** Copy and rename the example mapping script `\Mapping\HandlingTrack_Mapping.m` to `\Mapping\MyMapping.m`.
- 6** In User mapping file, select the `MyMapping.m` file.
- 7** Right-click the icon next to the edit field.



Click Open. The M file is opened in MATLAB.

- 8** Edit the file in MATLAB.  
Modify the variable names and unit conversions in the user mapping script according to your raw data. In this case, the mapping of original data area must be adapted to

```
% mapping of original data -----
orig.x = Road_x'; % x [m]
orig.y = Road_y'; % y [m]
orig.height = Road_z'; % z [m]
orig.q = zeros(size(orig.x)); % LatSlope [rad]
```

The lateral slope is set to zero in this example.

In the data reduction area, there is a code line that is not usable for this data. You must delete or comment this code line:

```
%roaddata = roaddata(1:3045,:);
```

**9** Enter a new result file name under Output filename.

**10** Save the conversion project.

**11** Run the conversion.

## Related topics

### HowTos

How to Start ASM RoadConverter.....	101
-------------------------------------	-----

## How to Convert Roads from ADAS RP

### Objective

To convert roads which are exported from ADAS RP, use the example project located in  
 <ASMPProject>\Parameterization\ASMPProcessing\Env\_RoadConverter\ADAS\_RP.

### Method

#### To convert roads from ADAS RP

- 1** In ADAS RP, create a route and export it using the RoadDataExport plug-in. The data is exported in several CSV files and one text file.
- 2** Copy the files to the ADAS\_RP\RawData folder of your road conversion project. You should use subfolders for sorting.
- 3** You have to convert the exported CSV files to MAT format, which can be used for road conversion. Run `asm_env_adasrp_parser.m` in MATLAB. In the Open dialog, select the exported text file.
- 4** In your road conversion project, select the text file or the converted MAT file. Perform the steps as described in [How to Import a Road in ModelDesk](#) on page 102.

## Tips & Tricks

---

**Introduction** This topic provides tips and tricks for work with road models.

---

**Closed roads** To create a closed road, it is recommended to convert one lap which has a gap between the first and last data points. After conversion, close the road in ModelDesk by using the **Close road** command from the context menu of the Segments pane in the ModelDesk's Road Generator.

---

**Decimation** The minimum segment length in the Road Generator is 1 m. It is recommended to use measurement data with distances greater than 1 m between data points. If the measurement data contains data with distances lower than 1 m, the points should be ignored during conversion. The **Delete points with distance <= <x> m** settings can be enabled and modified on the Mapping page of ASM RoadConverter.

---

**Related topics**

**References**

[Close Road \(ModelDesk Road Creation\)](#)

# ASM Controller Adaption

## Where to go from here

## Information in this section

Introduction to ASM Controller Adaption.....	107
Tuning Methods.....	119
Describes the different tuning methods.	

## Introduction to ASM Controller Adaption

## Where to go from here

## Information in this section

Basics on ASM Controller Adaption.....	107
ASM Controller Adaption is used to tune a controller.	
Plant Types.....	109
There are different plant types with different transfer functions and parameters.	
User interface of ASM Controller Adaption.....	111
The user interface of ASM Controller Adaption is started from MATLAB.	
Working with ASM Controller Adaption.....	114
The example illustrates how a cascade controller structure of a permanent magnet synchronous motor (PMSM) can be tuned with ASM Controller Adaption.	

## Basics on ASM Controller Adaption

### Introduction

The ASM Controller Adaption tool offers three different approaches for fine-tuning a controller for a known plant.

### Requirements

To use ASM Controller Adaption, you need the MATLAB Control System Toolbox.

**Overview**

The controller can be adapted according to the following approaches:

- Amplitude optimum
- Symmetrical optimum
- Ziegler-Nichols (step response method)

The controller can be designed for the following plant types:

- First-order lag element
- Second-order lag element
- First-order lag element with integrator
- First-order lag element with dead time

Not all plant types can be adapted with each tuning method. The following table gives an overview of the valid tuning approaches for the different plant types:

<b>Plant Type</b>	<b>Amplitude Optimum</b>	<b>Symmetrical Optimum</b>	<b>Ziegler-Nichols Approach</b>
First-order lag element	✓	–	–
Second-order lag element	✓	✓ <sup>1)</sup>	✓ <sup>1)</sup>
First-order lag element with integrator	✓ <sup>1)</sup>	✓	–
First-order lag element with dead time	✓ <sup>1)</sup>	✓ <sup>1)</sup>	✓ <sup>1)</sup>

<sup>1)</sup> Certain conditions have to be fulfilled to apply the method for the specific plant.

The controller tuning rules and the conditions to be fulfilled are described in [Tuning Methods](#) on page 119.

The controller is assumed to be a PI controller. This description refers to a controller with the following underlying equation in the Laplace domain:

$$G_C(s) = \frac{K_R}{s} \cdot \frac{T_R s + 1}{1}$$

where:

$K_R$  is the gain

$T_R$  is the time constant of the controller

**Related topics****Basics**

Amplitude Optimum.....	119
Symmetrical Optimum.....	122
Ziegler-Nichols Method.....	125

# Plant Types

## Introduction

ASM Controller Adaption can be used for various plant types with different transfer functions and parameters.

## Different plant types

You can use ASM Controller Adaption with the following plant types:

- First-order lag element
- Second-order lag element
  - Real second-order lag element
  - Composed second-order lag element
- First-order lag element with integrator
- First-order lag element with dead time

## First-order lag element

The transfer function of the first-order lag element in the Laplace domain is:

$$G(s) = \frac{K}{Ts + 1}$$

where:

$K$  is the gain

$T$  is the time constant of the plant

## Second-order lag element

There are two different ways to parameterize a second-order lag element. As transfer function parameters you can use either the parameters of a *real* second-order lag element or the parameters of a second-order lag element *composed* of two first-order lag elements.

**Real second-order lag element** The transfer function in the Laplace domain of a real second-order lag element has the following form:

$$G(s) = \frac{K}{T^2 s^2 + 2dTs + 1}$$

where:

$d$  is the damping of the plant

$K$  is the gain

$T$  is the time constant of the plant

**Composed second-order lag element** If the damping is equal to or larger than 1, the second-order lag element can be described by two first-order lag elements in series. This is the composed second-order lag element. For dampings lower than 1, the plant has complex conjugate poles.

The composed second-order lag element is determined by:

$$G(s) = \frac{K}{T_1 s + 1} \cdot \frac{1}{T_2 s + 1}$$

where:

$K$  is the gain

$T_1$  is the first-order time constant of the plant

$T_2$  is the second-order time constant of the plant

#### First-order lag element with integrator

The first-order lag element with integrator is composed by a first-order lag element and an integrator in series. The transfer function in the Laplace domain is:

$$G(s) = \frac{K}{T_1 s} \cdot \frac{1}{T_2 s + 1}$$

where:

$K$  is the gain

$T_1$  is the integral time constant

$T_2$  the first order time constant of the plant

#### First-order lag element with dead time

The first-order lag element with dead time is composed by a first-order lag element and a dead time in series. The transfer function in the Laplace domain is given as:

$$G(s) = \frac{K}{T_1 s + 1} \cdot e^{-sT_t}$$

where:

$K$  is the gain

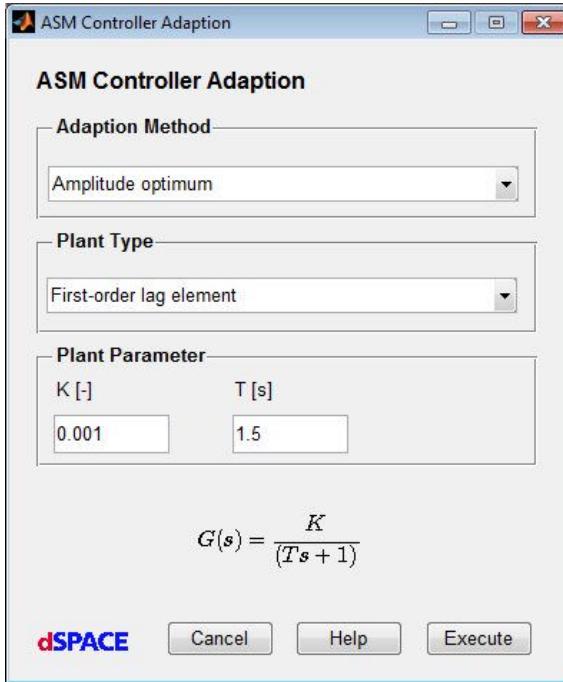
$T_1$  is the first-order time constant of the plant

$T_t$  is the dead time of the plant

## User interface of ASM Controller Adaption

### Starting the user interface

Enter `asm_controller_adaption` in the MATLAB Command Window.



### Parameters

**Adaption Method** Lets you select one of the following adaption methods:

- Amplitude optimum
- Symmetrical optimum
- Ziegler-Nichols method

**Plant Type** Lets you select one of the following plant types:

- First-order lag element
- Real second-order lag element
- Composed second-order lag element
- First-order lag element with integrator
- First-order lag element with dead time

**Plant Parameter** Lets you enter the plant parameters. Depending on the selected plant type, different plant parameters are available.

The following table shows the possible plant types and the parameters you have to enter for each.

Plant type	Parameters	Description
First-order lag element	K	Gain
	T	Time constant

Plant type	Parameters	Description
Real second-order lag element	K T d	Gain Time constant Damping coefficient
Composed second-order lag element	K T <sub>1</sub> T <sub>2</sub>	Gain First-order time constant Second-order time constant
First-order lag element with integrator	K T <sub>1</sub> T <sub>2</sub>	Gain Integral time constant First-order time constant
First-order lag element with dead time	K T <sub>1</sub> T <sub>t</sub>	Gain First-order time constant Dead time

**Execute** Adjusts the controller according to the selected options. The dialog remains open.

**Help** Opens dSPACE Help.

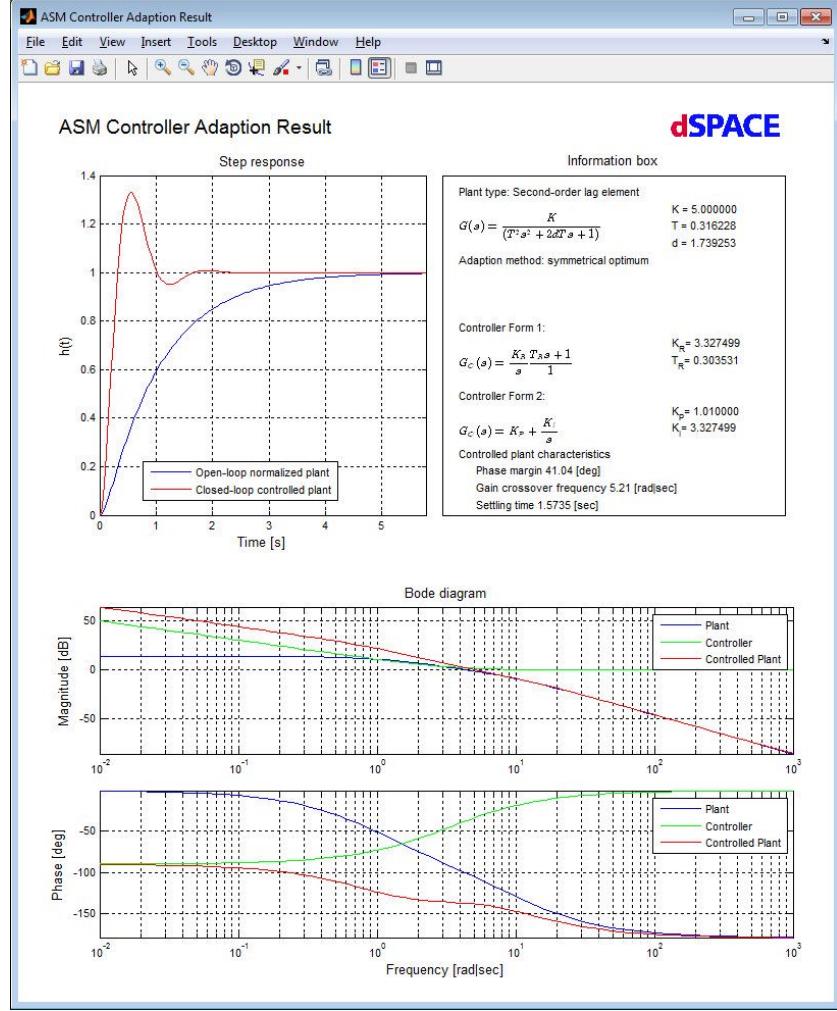
**Cancel** Closes the dialog without adjusting the controller.

---

#### Output report

The output report is generated automatically if the specific tuning conditions are fulfilled and no error occurs.

The following illustration shows an example report:



The top left of the ASM Controller Adaption Result window shows the step response of the closed-loop controlled plant and open-loop normalized plant (normalized by plant gain K).

The information box at the top right provides information about the plant transfer function, the tuning method, the controller parameters in two different forms, and the characteristics of the controlled plant. The box also informs you, whether warnings occurred while the controller was being tuned. The warning messages are displayed in detail in the MATLAB Command Window.

Tuning warning: The plant is low damped. Amount optimum method can fail and lead to instability.  
f<sub>t</sub> >>

The bottom half of the ASM Controller Adaption Result window shows a Bode plot with magnitude and phase of the transfer function.

## Working with ASM Controller Adaption

### Introduction

The following example illustrates how a cascade controller structure of a permanent magnet synchronous motor (PMSM) can be tuned with ASM Controller Adaption. A cascade controller structure has to be tuned from the innermost control loop to the outermost one. In this example, the controller structure consists of an inner current control loop and an outer speed control loop.

### Current control loop

The plant to be controlled by the current controller consists of the motor inductance (inductivity  $L$  and resistance  $R$ ). It can be described as a first-order lag element. The parameters are calculated by:

$$K = \frac{1}{R}$$

$$T = \frac{L}{R}$$

There are often further time constants (smaller than the time constant of the inductance) in the plant. Typically, you also have to take into account the dead time of the inverter, the delay of the sample and hold element for the current measurement, and possible measurement filter time constants. Therefore, the current control plant can be approximated as first-order lag element with dead time or as a composed second-order lag element, whereby the smaller time constants can be added up to one time constant ( $T_2$  or  $T_t$ ).

You can tune the current controller by using the amplitude optimum method for good reference behavior or the symmetrical optimum method for good disturbance behavior. The Ziegler-Nichols approach can also be applied. For a further explanation of current controller tuning, refer to [SCH09].

### Speed control loop

The speed control plant consists of the closed current control loop and the rotational mechanical part in series. The mechanical part with the inertia  $J$  and the friction  $F$  can be described as a first-order lag element or as a simple integrator. The first-order lag element is determined by:

$$K = \frac{1}{F}$$

$$T = \frac{J}{F}$$

If the mechanics are regarded as the integrator, the integral time constant is calculated by:

$$T = J$$

The closed current control loop is the second part of the speed control plant. To keep the effort required for the controller tuning low, the current control plant is regarded as a first-order lag element with one equivalent time constant  $T_{eq}$ .

$$G_{CurrentControlLoop}(s) = \frac{1}{T_{eq}s + 1}$$

To determine the equivalent time constant of the current control loop, the settling time of the current control loop can be taken into account. It can be multiplied by a factor between 1 and 2. A higher factor results in a more stable behavior and a smaller factor in a faster system response.

$$T_{eq} = 1 \dots 2 T_{SettlingTime\ Current\ Control\ Loop}$$

Further time constants, such as measurement delay times, or sample and hold dead times, can be added to the smaller time constant as well.

In addition, the motor constant  $K_M$ , which describes the relation between the torque-generating current  $i$  and the torque  $Trq$ , has to be considered:

$$K_M = \frac{i}{Trq} = \frac{[A]}{[Nm]}$$

Thus, the speed control plant can be described as a first-order lag element with integrator or a second-order lag element (composed of two first-order lag elements).

$$G_{SpeedControlPlant}(s) = K_M \cdot G_{CurrentControlLoop}(s) \cdot G_{Mechanic}(s)$$

It is recommended to use the symmetrical optimum approach for the controller tuning to achieve a good disturbance behavior of the closed-loop plant.

### Example

In this example, a PMSM with the following characteristics is tuned:

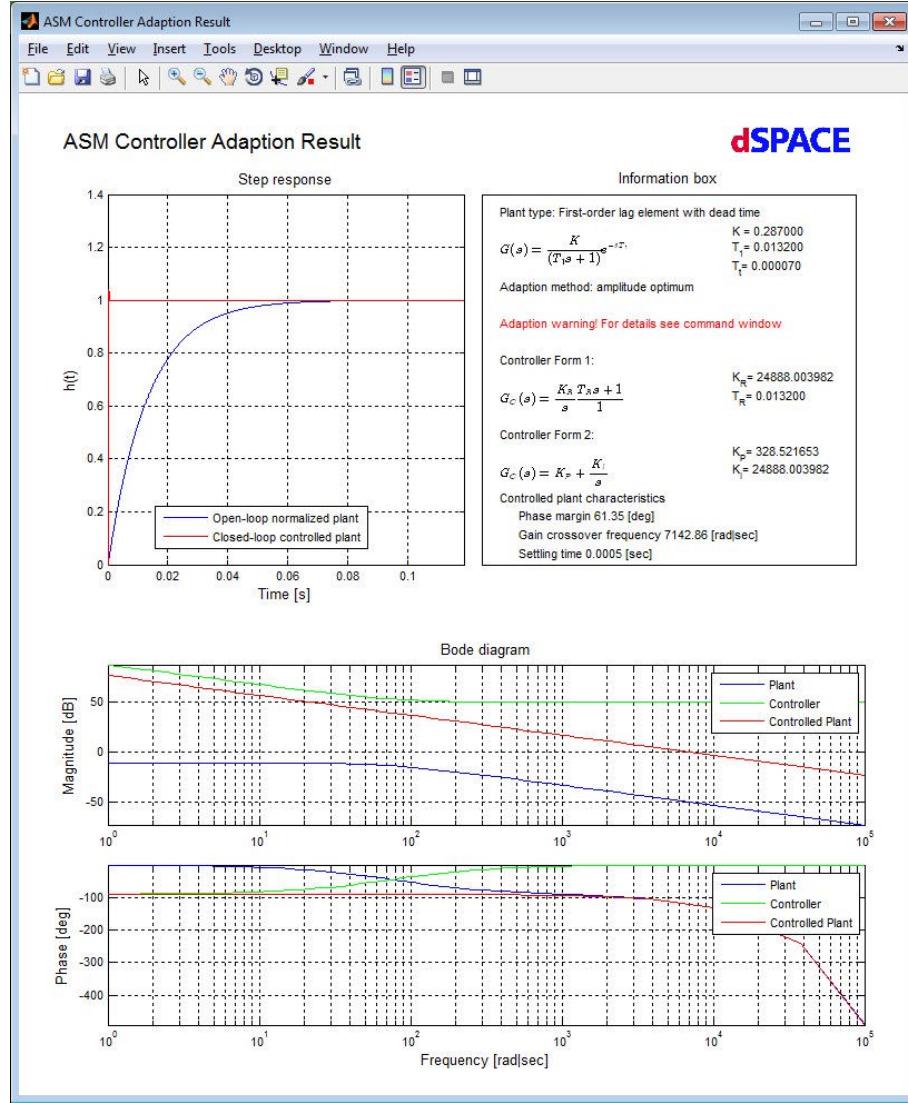
$$L = 45.83 \text{ mH}$$

$$R = 3.48 \Omega$$

Current measurement dead time:

$$T_t = 70 \mu s$$

The current controller is tuned with the amplitude optimum method. The following illustration shows the result report:



### Note

The report displays a tuning warning because the larger time constant is more than 4 times larger than the smaller time constant. According to [SCH09], it is recommended to apply the symmetrical optimum method. Nevertheless, the amplitude optimum method is used in this case. Choose the tuning method individually for each use case, depending on the desired behavior.

**Note**

The inner current control loop should be tested on the real plant. The controller parameters can be optimized with regard to the desired behavior. It might be more appropriate to determine the settling time (used for tuning the speed controller) with the step response of the real current plant.

The equivalent time constant of the closed-loop current control plant is calculated by the settling time of the current control closed-loop step response and a factor of 2 (can be any number between 1 and 2):

$$G_{CurrentControlLoop}(s) = \frac{1}{0.001s + 1}$$

The inertia is:

$$J = 0.0046 \text{ kg m}^2$$

The ratio between the torque-generating current and the torque is determined as:

$$K_M = 5.1 \frac{[A]}{[Nm]}$$

The delay time of the speed measurement is:

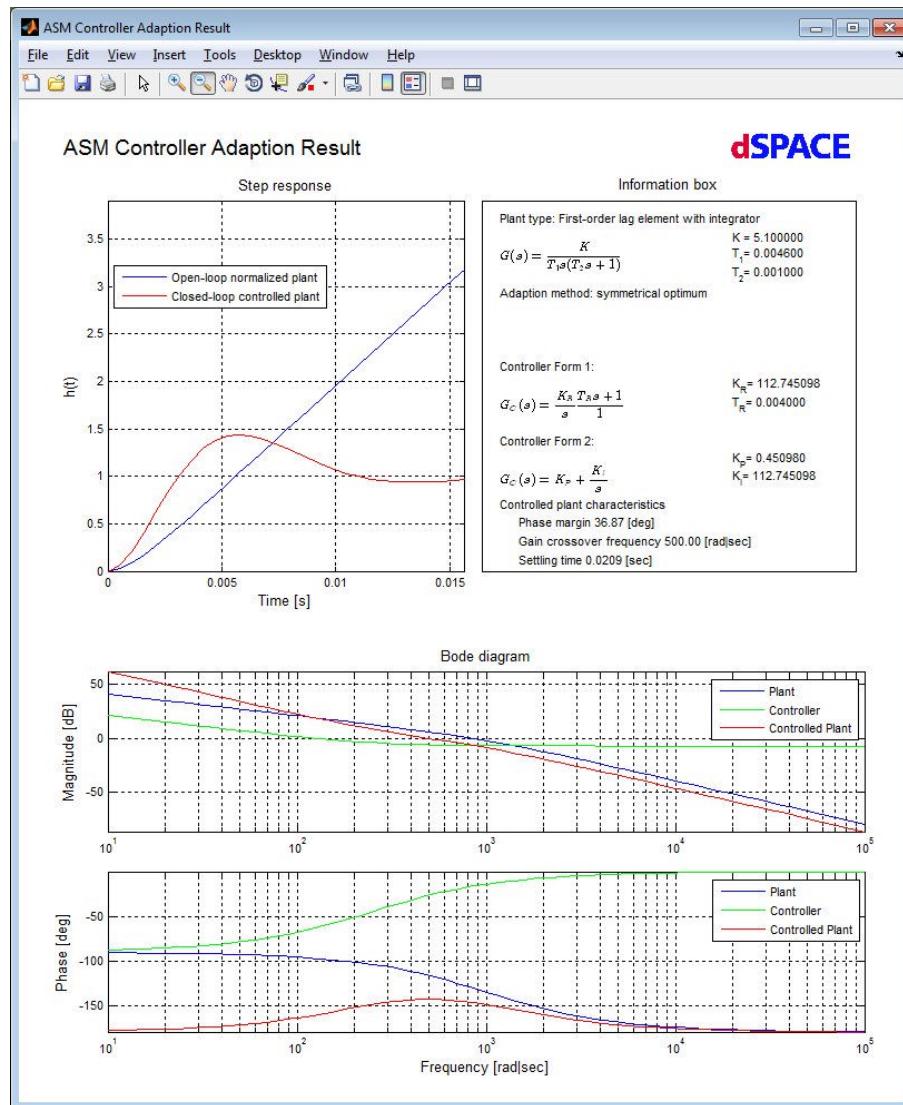
$$T_t = 70 \mu s$$

which can be added to the smaller time constant.

Thus, the simplified transfer function of the speed control can be written as:

$$G(s) = \frac{5.1}{0.0046s} \cdot \frac{1}{0.001s + 1}$$

To achieve good disturbance behavior, the symmetrical optimum method is used for tuning. Refer to the following illustration.



## Related topics

### Basics

Ziegler-Nichols Method.....	125
-----------------------------	-----

# Tuning Methods

## Where to go from here

## Information in this section

[Overview of Tuning Methods](#)..... 119

There are three different controller tuning methods to chose from.

[Amplitude Optimum](#)..... 119

The amplitude optimum tuning approach is available for several plant types.

[Symmetrical Optimum](#)..... 122

The symmetrical optimum tuning approach is available for several plant types.

[Ziegler-Nichols Method](#)..... 125

The Ziegler-Nichols tuning approach is available for a delayed first-order lag element or a second-order lag element under certain conditions

## Overview of Tuning Methods

### Methods

You can select one of three controller tuning options:

- Amplitude optimum
- Symmetrical optimum
- Ziegler-Nichols (step response method)

The tuning rules for each plant type are described in the following.

### Related topics

### Basics

[Amplitude Optimum](#)..... 119

[Symmetrical Optimum](#)..... 122

[Ziegler-Nichols Method](#)..... 125

## Amplitude Optimum

### Introduction

The amplitude optimum tuning approach is available for all plant types listed in [Plant Types](#) on page 109. This section describes the calculation rules for each plant type.

**First-order lag element**

Transfer function plant:

$$G(s) = \frac{K}{Ts + 1}$$

Transfer function controller:

$$G_C(s) = \frac{K_R}{s} \cdot \frac{Trs + 1}{1}$$

Calculation rule:

$$Tr = T$$

$$K_R = \frac{10}{2 \cdot T \cdot K}$$

**Tip**

$K_R$  can be chosen arbitrarily. The closed-loop system is stable for any  $K_R$ . For the amplitude optimum method,  $K_R$  is determined without an exact calculation rule.

**Second-order lag element**

Transfer function plant:

$$G(s) = \frac{K}{T^2 s^2 + 2dTs + 1}$$

**Note**

The damping of the plant has to be sufficient. If the damping  $d$  is smaller than 1, the amplitude optimum method might fail and cause instability [FOE94].

Transfer function controller:

$$G_C(s) = \frac{K_R}{s} \frac{Trs + 1}{1}$$

**Real second-order lag element** For a real second-order lag element, the controller parameters are determined by another tuning rule. Therefore, the plant parameters have to be converted into the following form [FOE94]:

$$G(s) = \frac{1}{a_2 s^2 + a_1 s + a_0}$$

The parameters are calculated by [FOE94]:

$$r_0 = a_0 \cdot \frac{a_1^2 - a_0 a_2}{a_1 a_2}$$

$$r_1 = a_1 \cdot \frac{a_1^2 - a_0 a_2}{a_1 a_2} - a_0$$

$$T_R = \frac{r_1}{r_0}$$

$$K_R = \frac{r_0}{2}$$

**Composed second-order lag element** If the transfer function is composed of two first-order lag elements in series and the larger time constant is 5 times larger than the smaller time constant, the controller is tuned according to the following rule.

Tuning condition [SCH09]:

$$T_1 > 4T_2$$

Calculation rule [FOE94]:

$$T_R = T_1$$

$$K_R = \frac{1}{2 \cdot K \cdot T_2}$$

#### Tip

If the larger time constant is more than 4 times larger than the smaller time constant, it is recommended to use the symmetrical optimum method instead. [SCH09]

### First-order lag element with integrator

Transfer function plant:

$$G(s) = \frac{K}{T_1 s} \cdot \frac{1}{T_2 s + 1}$$

Transfer function controller:

$$G_C(s) = \frac{K_R}{s} \cdot \frac{T_R s + 1}{1}$$

#### Note

The following tuning condition has to be fulfilled to apply the amplitude optimum method for a first-order lag element with integrator.

Tuning condition [SCH09]:

$$\frac{T_1}{K T_2} \gg 1$$

The plant parameters have to be converted into the following form [FOE94]:

$$G(s) = \frac{1}{a_2 s^2 + a_1 s + a_0}$$

The parameters are then calculated as follows [FOE94]:

$$r_0 = 2 \cdot 10^{-6} \text{ (chosen to avoid division through zero)}$$

$$r_1 = a_1 \cdot \frac{a_1^2 - a_0 a_2}{a_1 a_2} - a_0$$

$$T_R = \frac{r_1}{r_0}$$

$$K_R = \frac{r_0}{2}$$

The resulting controller is a P controller due to a neglectable I-part ( $10^{-6}$ ).

#### First-order lag element with dead time

Transfer function plant:

$$G(s) = \frac{K}{T_1 s + 1} \cdot e^{-s T_t}$$

Transfer function controller:

$$G_C(s) = \frac{K_R}{s} \cdot \frac{T_R s + 1}{1}$$

#### Note

The following tuning condition has to be fulfilled to apply the amplitude optimum method for a first-order lag element with dead time.

Tuning condition:

$$T_t < T_1$$

Tuning rule [SCH09]:

$$T_R = T_1$$

$$K_R = \frac{T_1}{2 \cdot K \cdot T_t}$$

#### Related topics

##### Basics

Symmetrical Optimum..... 122

## Symmetrical Optimum

#### Introduction

The symmetrical optimum tuning approach is available for all plant types listed in [Plant Types](#) on page 109, except for the first-order lag element. The following sections describe the calculation rules for each plant type.

**Second-order lag element**

Transfer function plant:

$$G(s) = \frac{K}{T^2 s^2 + 2dTs + 1}$$

**Note**

The damping of the plant has to be larger than 1. Otherwise, the plant will have complex conjugate poles and the symmetrical optimum approach cannot be applied. [FOE94]

Tuning condition:

$$d \geq 1$$

Therefore, the plant has to be composed of two first-order lag elements and the transfer function can be rewritten as:

$$G(s) = \frac{K}{T_1 s + 1} \cdot \frac{1}{T_2 s + 1}$$

Transfer function controller:

$$G_C(s) = \frac{K_R}{s} \cdot \frac{T_R s + 1}{1}$$

It is assumed that  $T_1$  is the larger time constant.

**Note**

If the larger time constant is less than 4 times larger than the smaller time constant, it is recommended to use the amplitude optimum method instead. [SCH09]

Tuning rule [SCH09]:

$$T_R = k_1 \cdot 4 \cdot T_2$$

$$K_R = k_2 \cdot \frac{T_1}{2 \cdot K \cdot T_2 \cdot T_R}$$

The correction factors  $k_1$  and  $k_2$  consider the ratio between both time constants. They are determined by [SCH09]:

$$k_1 = \frac{1 + \left(\frac{T_2}{T_1}\right)^2}{\left(1 + \frac{T_2}{T_1}\right)^3}$$

$$k_2 = 1 + \left(\frac{T_2}{T_1}\right)^2$$

**First-order lag element with integrator**

Transfer function plant:

$$G(s) = \frac{K}{T_1 s} \cdot \frac{1}{T_2 s + 1}$$

Transfer function controller:

$$G_C(s) = \frac{K_R}{s} \cdot \frac{T_R s + 1}{1}$$

The parameters are calculated by [SCH09]:

$$T_R = 4 \cdot T_2$$

$$K_R = \frac{T_1}{2 \cdot K \cdot T_2}$$

**First-order lag element with dead time**

Transfer function:

$$G(s) = \frac{K}{T_1 s + 1} \cdot e^{-sT_t}$$

Transfer function controller:

$$G_C(s) = \frac{K_R}{s} \cdot \frac{T_R s + 1}{1}$$

**Note**

The rising time has to be larger than the delay time. Otherwise, the symmetrical optimum method cannot be applied. [FOE94]

Tuning condition:

$$T_t < T_1$$

**Note**

If the larger time constant is less than 4 times larger than the smaller time constant, it is recommended to use the amplitude optimum method instead. [SCH09]

Calculation rule [SCH09]:

$$T_R = k_1 \cdot 4 \cdot T_t$$

$$K_R = k_2 \cdot \frac{T_1}{2 K T_t T_R}$$

The correction factors  $k_1$  and  $k_2$  consider the ratio between both time constants. They are determined by [SCH09]:

$$k_1 = \frac{1 + \left(\frac{T_t}{T_1}\right)^2}{\left(1 + \frac{T_t}{T_1}\right)^3}$$

$$k_2 = 1 + \left(\frac{T_t}{T_1}\right)^2$$

**Related topics****Basics**

Amplitude Optimum.....	119
------------------------	-----

## Ziegler-Nichols Method

**Introduction**

The Ziegler-Nichols tuning approach is available for a delayed first-order lag element or a second-order lag element under certain conditions. This section describes the calculation rules for both plant types. There are two methods to tune a controller with the Ziegler-Nichols approach. Here, only the step response method is used in accordance with [FOE94].

**Second-order lag element**

Transfer function plant:

$$G(s) = \frac{K}{T^2 s^2 + 2dTs + 1}$$

**Note**

The damping of the plant has to be larger than 1. Otherwise, the plant Ziegler-Nichols approach cannot be applied [FOE94].

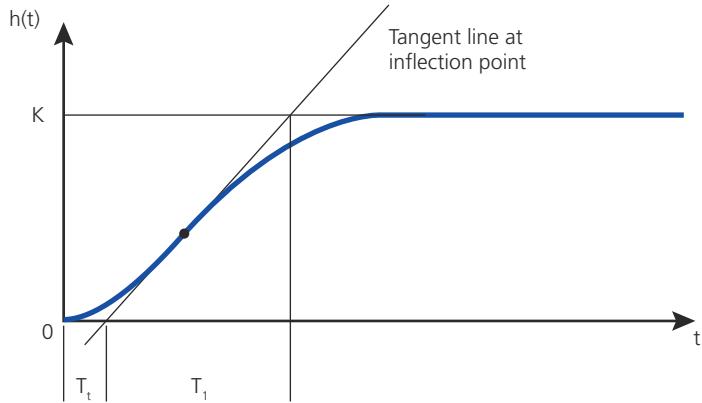
Tuning condition:

$$d \geq 1$$

Transfer function controller:

$$G_C(s) = \frac{K_R}{s} \cdot \frac{T_R s + 1}{1}$$

To tune the controller, the step response of the plant has to be recorded. A tangent at the inflection point of the step response is required to determine the rising time and the delay time for the controller tuning.

**Note**

The rising time has to be larger than the delay time. Otherwise, the Ziegler-Nichols approach cannot be applied [FOE94].

Tuning condition:

$$T_1 > T_t$$

Tuning rule [FOE94]:

$$T_R = 3.33 \cdot T_t$$

$$K_R = 0.9 \cdot \frac{T_1}{T_t \cdot K \cdot T_R}$$

**First-order lag element with dead time**

Transfer function:

$$G(s) = \frac{K}{T_1 s + 1} \cdot e^{-s T_t}$$

Transfer function controller:

$$G_C(s) = \frac{K_R}{s} \cdot \frac{T_R s + 1}{1}$$

**Note**

The rising time has to be larger than the delay time. Otherwise, the Ziegler-Nichols approach cannot be applied [FOE94].

The following tuning condition has to be fulfilled [FOE94]:

$$T_t < T_1$$

Tuning rule [FOE94]:

$$T_R = 3.33 \cdot T_t$$

$$K_R = 0.9 \cdot \frac{T_1}{T_t \cdot K \cdot T_R}$$

---

**Related topics****Basics**

Working with ASM Controller Adaption.....	114
---	-----

# Adams2ASM Converter

## Where to go from here

## Information in this section

[Basics on the Adams2ASM Converter](#)..... 128

The Adams2ASM Converter lets you convert data from an Adams Car™ model for use with ModelDesk.

[Parameters in Adams](#)..... 130

You can specify different suspension parameters.

[How to Use the Adams2ASM Converter](#)..... 135

To use the suspension data from an Adams Car project in ASM, you must perform the following steps.

## Basics on the Adams2ASM Converter

### Introduction

The Adams2ASM Converter lets you convert data from an Adams Car™ model for use with ModelDesk.

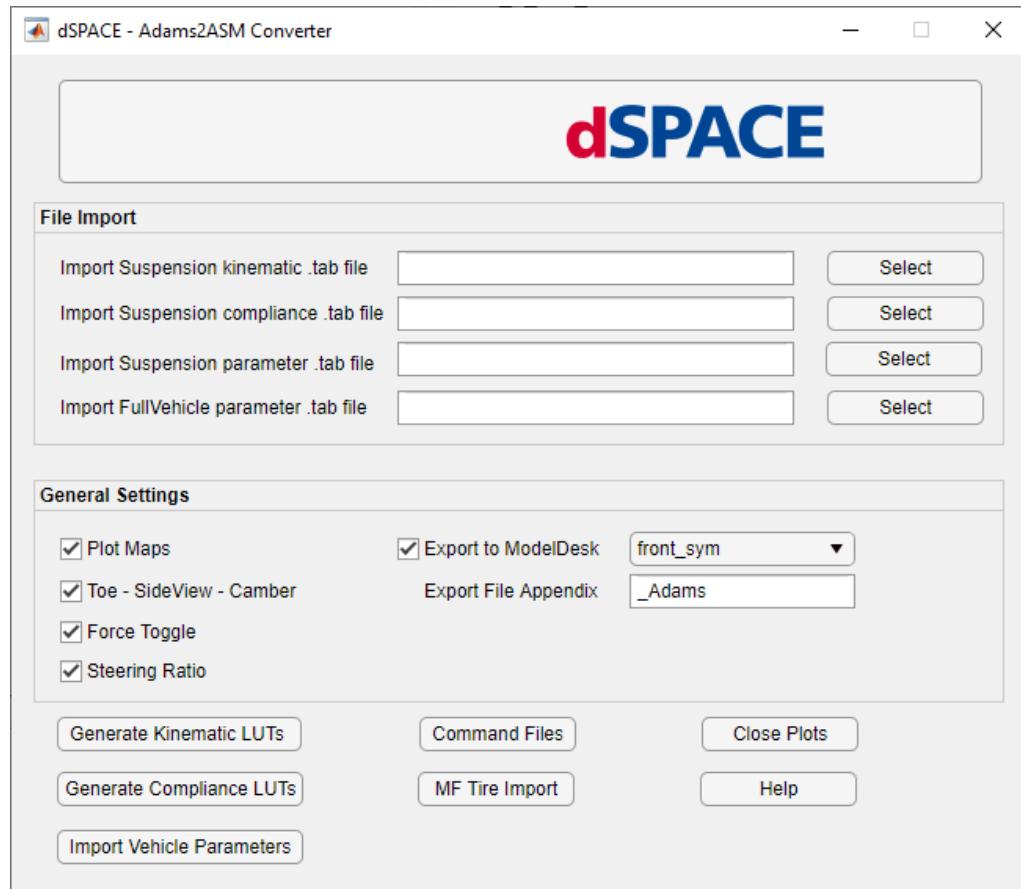
This way, you can use suspension kinematics, compliance and full vehicle data from an Adams Car project for parameterization in ModelDesk.

For an Adams model to be converted to an ASM model, you must specify some parameters in your Adams project. Refer to [Parameters in Adams](#) on page 130.

### Opening the Adams2ASM Converter

You can open the Adams2ASM Converter in MATLAB via `asm_vd_adams_import`.

## User interface



The user interface of the Adams2ASM Converter contains the following elements:

Element	Description
File import	To select the TAB files generated in Adams Car for import.
Import TAB file	
General Settings	
Plot Maps	To generate plots when exporting data.
Toe-Sideview-Camber	To select the toe-sideview-camber coordinate system. Otherwise, the cardan angles are used.
Force Toggle	To consider the ancillary spring. The normal force of the wheel is set as spring force. Also the ratio of wheel displacement ( $Displ\_Z\_Wheel[m]$ ) and spring displacement ( $Displ\_Spring[m]$ ) is 1.

Element	Description
Export to ModelDesk	To export the converted data directly to your ModelDesk project. Select the corresponding parameter type in the list.
	<p><b>Note</b></p> <p>Make sure to make the correct selection in the list and to open the corresponding parameter page in ModelDesk. Otherwise, export will stop with an error.</p>
Export File Appendix	To specify a name appendix for the export files.
Others	
Generate Kinematics LUTs	To start conversion of the kinematics data.
Generate Compliance LUTs	To start conversion of the compliance data.
Import Vehicle Parameter Command Files	To start conversation of the full vehicle data.
MF Tire Import	To open the user interface for import of Magic Formula parameters from a TIR file to the corresponding ModelDesk XML file. Refer to <a href="#">Importing TIR Files (ASM Vehicle Dynamics Addendum)</a> .
Close Plots	To close all plots that were opened during conversion.
Help	To open dSPACE Help on the Adams2ASM Converter.

## Parameters in Adams

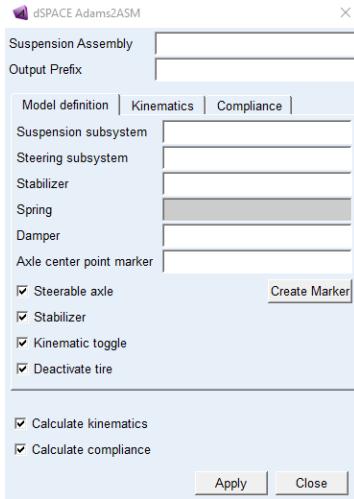
### Introduction

After importing the CMD files to your Adams working folder, the dSPACE tab is available in Adams Car. When you click the tab, you can select a *dependent* or an *independent* axle.

The following parameters are available:

- Dependent axle, refer to [Parameters for a dependent axle](#) on page 131.
- Independent axle, refer to [Parameters for an independent axle](#) on page 133.

The following illustration shows as an example the dialog for parameterization of a dependent axle.



#### Parameters for a dependent axle

You can specify the following parameters for a dependent axle:

##### General

Parameter	Description
Suspension Assembly	To select the suspension assembly to be analyzed.
Output Prefix	To define the name of the output file.
Calculate kinematics checkbox	To generate the results for the kinematics look-up tables. The results are stored in the Adams working folder as <code>Adams_LUT_kinematic.tab</code> .
Calculate compliance checkbox	To generate the results for the compliance look-up tables. The results are stored in the Adams working folder as <code>Adams_LUT_compliance.tab</code>
Apply	To apply the changes.
Close	To close the dialog.

##### Model definition tab

Parameter	Description
Suspension subsystem	To select the suspension subsystem.
Steering subsystem	To select the steering subsystem.
Stabilizer	To select a reference point that describes the displacement of the stabilizer.
Spring	To select the left spring of the suspension.
<b>Note</b>	
Do not use this parameter for leaf springs. In this case, you must select Force Toggle in the Adams2ASM Converter user interface.	
Damper	To select the left damper of the suspension.

Parameter	Description
Axle center point marker	To define the axle center point marker.
Steerable axle checkbox	To activate steerable suspension.
Stabilizer checkbox	To activate if a stabilizer is present.
Kinematic toggle checkbox	To automatically activate the kinematic mode in Adams for the subsystem. This is recommended. If there is a problem in the calculation of the imaginary steering axle, you must manually select the calculation mode.
Deactivate tire checkbox	To deactivate the tires so that the impressed forces and moments act on the wheel center. This is recommended.
Create marker	To create an axle center point marker. When you click the button, a dialog opens. Define the following parameters: <ul style="list-style-type: none"> <li>▪ Type: Single</li> <li>▪ Location dependency: Centered between coordinates</li> <li>▪ Centered between: Two coordinates</li> <li>▪ Coordinate reference #1: Left wheel center</li> <li>▪ Coordinate reference #2: Right wheel center</li> <li>▪ Orientation dependency: User-entered values</li> <li>▪ Orientation used: Euler Angles</li> <li>▪ Euler angles: 0,0,0</li> </ul>

#### Kinematics tab

Parameter	Unit	Description
Suspension parameters		
Number of Steps	[]	To specify the number of sampling points.
Bump Travel	[m]	To specify the wheel jounce (positive). <sup>1)</sup>
Alpha Angle Wheel	[deg]	To specify the torsion of the axle around its center point. <sup>1)</sup>
Steering parameters		
Number of Steps	[]	To specify the number of sampling points.
Steering Wheel Angle	[deg]	To specify the amplitude of the steering wheel. <sup>1)</sup>
Ratio Steering Wheel to PitmanArm	[deg/deg]	To specify the ratio of the steering wheel to the pitman arm.

<sup>1)</sup> Make sure that the displacement of the wheel and the steering rod is strictly monotonically increasing. Avoid that the suspension runs into a bump or makes a rebound stop.

#### Compliance tab

Parameter	Unit	Description
Number of steps	[]	To specify the number of sampling points.
Left Wheel		
Stimulus Force Fx	[N]	To specify the stimulus force in the x-direction at the wheel center.
Stimulus Force Fy	[N]	To specify the stimulus force in the y-direction at the wheel center.

Parameter	Unit	Description
Stimulus Torque Mx	[Nm]	To specify the stimulus torque in the x-direction at the wheel center.
Stimulus Torque Mz	[Nm]	To specify the stimulus torque in the z-direction at the wheel center.
<b>Axle Center</b>		
Stimulus Force Fx	[N]	To specify the stimulus force in [N] in the x-direction at the axle center.
Stimulus Force Fy	[N]	To specify the stimulus force in [N] in the y-direction at the axle center.
Stimulus Torque Mz	[Nm]	To specify the stimulus torque in [Nm] in the z-direction at the axle center.

#### Parameters for an independent axle

You can specify the following parameters for an independent axle:

##### General

Parameter	Description
Suspension Assembly	To select the suspension assembly to be analyzed.
Output Prefix	To define the name of the output file.
Calculate kinematics checkbox	To generate the results for the kinematics look-up tables. The results are stored in the Adams working folder as <b>Adams_LUT_kinematic.tab</b> .
Calculate compliance checkbox	To generate the results for the compliance look-up tables. The results are stored in the Adams working folder as <b>Adams_LUT_compliance.tab</b> .
Apply	To apply the changes.
Close	To close the dialog.

##### Model definition tab

Parameter	Description
Suspension subsystem	To select the suspension subsystem.
Steering subsystem	To select the steering subsystem.
Stabilizer	To select a reference point that describes the displacement of the stabilizer.
Spring	To select the left spring of the suspension.
<b>Note</b>	
Do not use this parameter for leaf springs. In this case, you must select Force Toggle in the Adams2ASM Converter user interface.	
Damper	To select the left damper of the suspension.
Steerable axle checkbox	To activate steerable suspension.
Stabilizer checkbox	To activate if a stabilizer is present.
Kinematic toggle checkbox	To automatically activate the kinematic mode in Adams for the subsystem. This is recommended. If there is a problem in calculating the imaginary steering axle, you must manually select the calculation mode.

Parameter	Description
Deactivate tire checkbox	To deactivate the tires so that the impressed forces and moments act on the wheel center. This is recommended.

**Kinematics tab**

Parameter	Unit	Description
Suspension parameters		
Number of Steps	[]	To specify the number of sampling points.
Bump Travel	[m]	To specify the wheel jounce (positive). <sup>1)</sup>
Rebound Travel	[m]	To specify the wheel rebound (negative). <sup>1)</sup>
Steering parameters		
Number of Steps	[]	To specify the number of sampling points.
Steering Rod displ.	[m]	To specify the amplitude of the steering rod displacement. <sup>1)</sup>

<sup>1)</sup> Make sure that the displacement of the wheel and the steering rod is strictly monotonically increasing. Avoid that the suspension runs into a bump or makes a rebound stop.

**Compliance tab**

Parameter	Unit	Description
Number of steps	[]	To specify the number of sampling points.
Stimulus Force Fx	[N]	To specify the stimulus force in the x-direction at the wheel center.
Stimulus Force Fy	[N]	To specify the stimulus force in the y-direction at the wheel center.
Stimulus Torque Mx	[Nm]	To specify the stimulus torque in the x-direction at the wheel center.
Stimulus Torque My	[Nm]	To specify the stimulus torque in the y-direction at the wheel center.
Stimulus Torque Mz	[Nm]	To specify the stimulus torque in the z-direction at the wheel center.

**Parameters for full vehicle**

You can specify the following parameters for the full vehicle:

**General**

Parameter	Description
Full Vehicle Assembly	To select the full vehicle assembly to be analyzed.
Output Prefix	To define the name of the output file.
Body subsystem	To select the body subsystem.
Powertrain subsystem	To select the powertrain subsystem.
Front tire subsystem	To select the front tire subsystem.
Rear tire subsystem	To select the rear tire subsystem.
Apply	To apply the changes.
Close	To close the dialog.

### Powertrain tab

Parameter	Description
Drivetrain Type	To specify the number of sampling points.
Front Differential (optional)	To select the front differential subsystem.
Center Differential (optional)	To select the center differential subsystem.
Rear Differential (optional)	To select the rear differential subsystem.

### Related topics

#### Basics

[Suspension \(ASM Vehicle Dynamics Addendum\)](#)

## How to Use the Adams2ASM Converter

### Objective

To use the suspension data from an Adams Car project in ASM, you must perform the following steps.

### Method

#### To use the Adams2ASM Converter

- 1 Call `asm_vd_adams_import` in the MATLAB Command Window.  
The Adams2ASM Converter opens.
- 2 Click Command Files to open the installation folder containing the following command files.
  - `dspace_adams2asm_converter_fullvehicle.cmd`
  - `dspace_adams2asm_converter_suspension.cmd`
  - `dspace_adams2asm_dbox.cmd`
  - `dspace_adams2asm_init.cmd`
- 3 Copy the files to your Adams working folder.
- 4 In Adams, execute the `dspace_adams2asm_init.cmd` file.  
In the Adams user interface, there is now a dSPACE tab.
- 5 Click the dSPACE tab and select the *independent* or *dependent* axle type or the full vehicle converter.  
A dialog opens.
- 6 Specify the suspension or full vehicle parameters and click **Apply**.  
Adams now provides:
  - Three TAB files in the Adams working folder for suspension conversion:
    - `Adams_LUT_parameter.tab`
    - `Adams_LUT_kinematic.tab`
    - `Adams_LUT_compliance.tab`

- One TAB file for full vehicle conversion:
    - **Adams\_LUT\_FullVehicle\_Parameter.tab**
- Copy the TAB files to your ASM working folder.
- 7** Open your ModelDesk project and experiment.
- 8** In the Adams2ASM Converter, select the TAB files and specify the settings for the export.  
Click Generate Kinematics LUTs, Generate Compliance LUTs or Import Vehicle Parameters.  
The Adams2ASM Converter starts the conversion.
- 

**Result**

The simulation data from Adams Car is converted and exported to your ModelDesk experiment. You can now use it for further parameterization in ModelDesk.

# ASM LabellInterface

## Where to go from here

## Information in this section

<a href="#">Basics of ASM LabellInterface Library</a> .....	137
ASM LabellInterface is a library to handle labels and buses in Simulink.	
<a href="#">Bus To Vector Block</a> .....	139
The Bus2Vector block converts a bus (or parts) to a vector.	
<a href="#">Change Labels Block</a> .....	151
The ChangeLabels block changes the labels of a bus.	

# Basics of ASM LabellInterface Library

## Where to go from here

## Information in this section

<a href="#">Basics on the ASM LabellInterface</a> .....	137
The ASM LabellInterface lets you handle labels and buses from within Simulink and lets you perform different actions.	
<a href="#">How to Open the ASM LabellInterface Library</a> .....	138
The ASM LabellInterface library must be opened in MATLAB.	

# Basics on the ASM LabellInterface

## Basics

LabellInterface is a library to handle labels and buses in Simulink. This is necessary if you want to route them to a block that does not support buses, for example, dSPACE Task Transition or IPC blocks. This toolbox makes it possible to change the bus structure into a flat muxed vector and vice versa.

The ASM LabellInterface lets you handle labels and buses from within Simulink and lets you perform the following actions:

- Route signals to blocks that do not support buses, such as the Simulink Unit Delay block or the IPC block contained in the dSPACE RTI-MP library.
- Change the bus structure into a flat vector and vice versa.
- Change labels.

- Make a bus structure visible or manipulable with ControlDesk or AutomationDesk.

**Note**

The ASM LabelInterface provides special features for the dSPACE RTI-MP IPC connections and SCALEXIO Multicore. Refer to the description of the IPC/ModelPort Block(s) in [Options Page](#) on page 141.

It is strongly recommended that you specify correct dimensions on the **Data Type and Dimensions** page. Otherwise, Simulink maps the dimensions dynamically, which might cause errors.

Wrong data types might cause errors if the bus contains different data types or if one of the subsequent blocks needs specific data types.

---

**Related topics****References**

<a href="#">Options Page</a> .....	141
------------------------------------	-----

## How to Open the ASM LabelInterface Library

---

**Objective**

The ASM LabelInterface library must be opened in MATLAB.

---

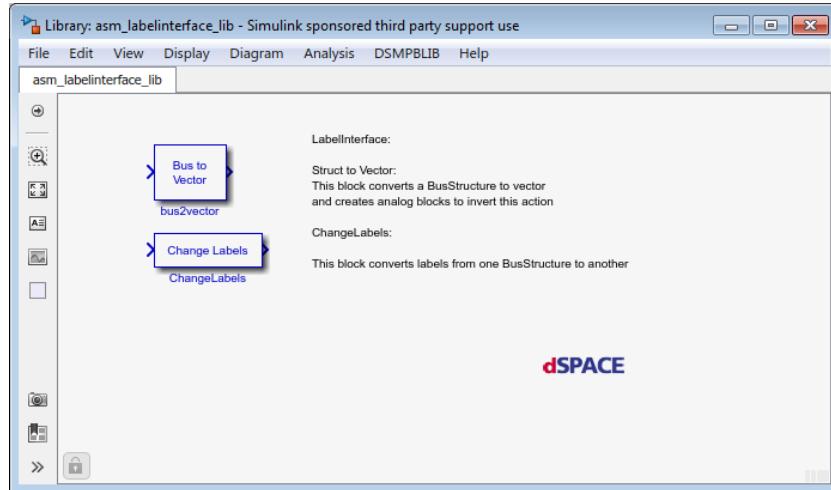
**Method****To open the ASM LabelInterface library**

- 1 In the MATLAB Command Window, type

```
asm_labelinterface_lib
```

**Result**

The library opens.

**Note**

You can also access the ASM LabellInterface via the Library Browser.

**Related topics****Basics**

[Basics on the ASM LabellInterface.....](#) 137

## Bus To Vector Block

**Where to go from here****Information in this section**

<a href="#">Description of Bus2Vector Block.....</a>	140
The Bus2Vector block converts a bus (or parts) to a vector.	
<a href="#">Options Page.....</a>	141
To select the blocks to be generated.	
<a href="#">Special Options Page.....</a>	142
To provide options for handling the size of vectors and TRC files.	
<a href="#">Signal Page.....</a>	146
To select some or all signals from the input bus structure.	

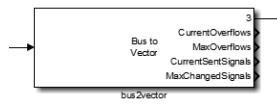
Datatypes & Dimensions Page.....	148
To set the dimensions and data types of the selected signals.	

## Description of Bus2Vector Block

### Basics

In some MATLAB releases, the UnitDelay and Memory blocks do not support Simulink buses. This also affects the IPC blocks of the dSPACE RTI-MP Blockset and certain S-functions. The Bus2Vector block (B2V) and the generated Vector2Bus block (V2B) are designed to support the routing of Simulink buses through Simulink blocks that do not allow buses.

The following shows the Bus2Vector block:



### Related topics

### References

Datatypes & Dimensions Page.....	148
Options Page.....	141
Signal Page.....	146
Special Options Page.....	142

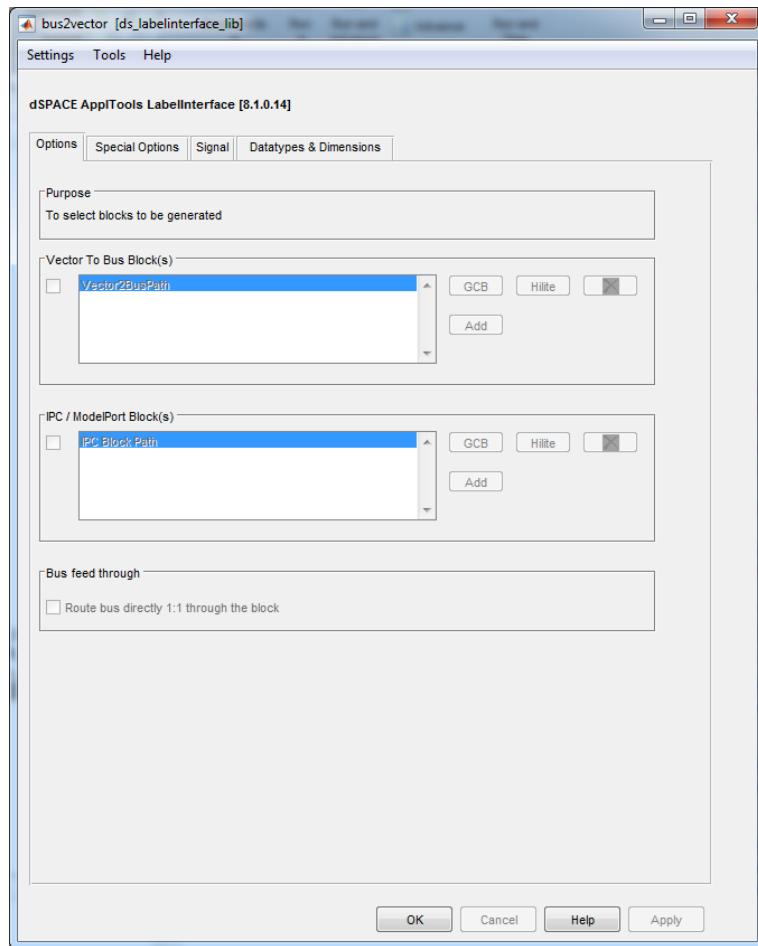
## Options Page

### Purpose

To select the blocks to be generated.

### Description

The Options page lets you make the general settings of the Bus2Vector block.



### Vector To Bus Block(s)

The checkbox lets you specify the blocks you want to generate. If the list is empty, a new block is created with a unique name.

**GCB** Adds the currently selected block to the list.

**Add** Adds a block of which you typed the full block path to the list.

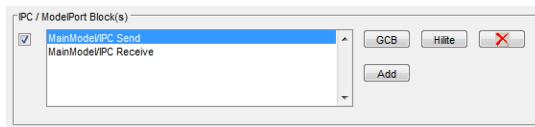
**X** Lets you remove the selected block from the list.

**IPC ModelPort Block(s)**

This checkbox lets you add blocks of the following types:

- IPC Send or IPC Receive blocks
- Model Import and Model Outport blocks contained in the dSPACE Model Port Block Library

The blocks are set to the correct data type and dimension during the generation process. You can add multiple blocks to the list.

**Bus feed-through**

When you select the Route bus directly 1:1 through the block checkbox, the LabelInterface blocks remain in the model and the bus feed-through is maintained as usual. Use this option if you do not want to manipulate the buses.

**Related topics****Basics**

Description of Bus2Vector Block..... 140

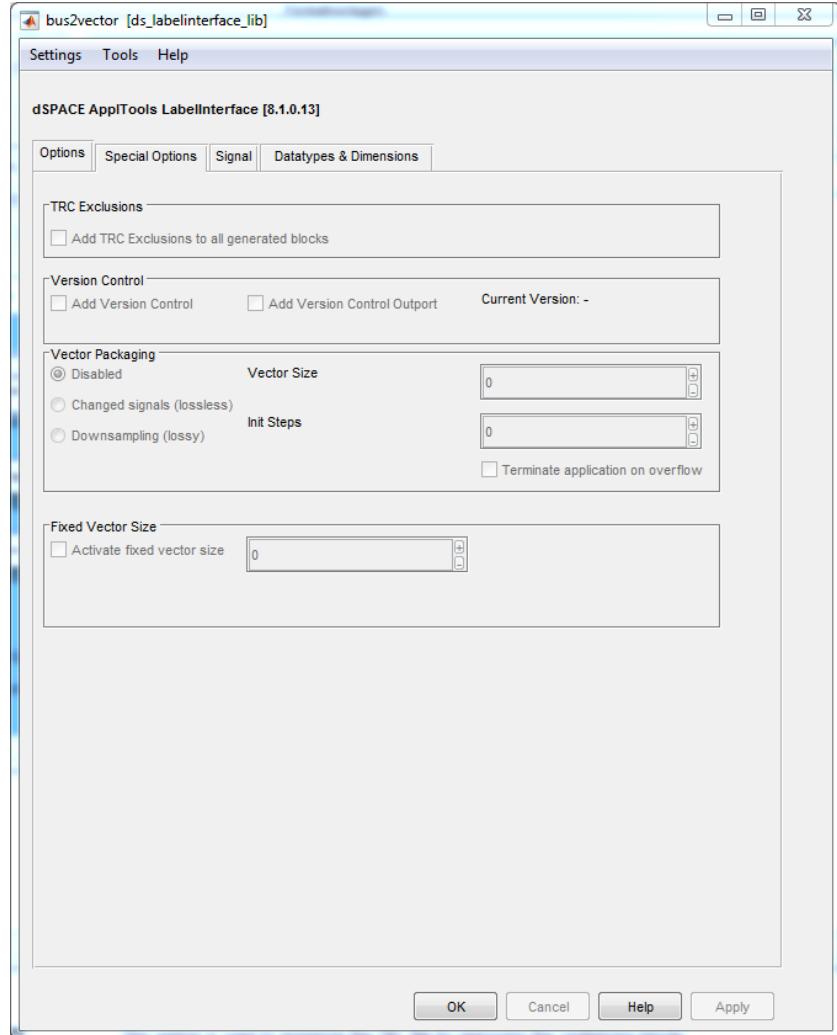
## Special Options Page

**Purpose**

To provide options for handling the size of vectors and TRC files.

## Description

The Special Options page provides a number of options for you to handle the size of vectors and TRC files:



## TRC Exclusions

Select the Add TRC Exclusions to all generated blocks checkbox to minimize the TRC file by removing the underlying blocks from it.

### Note

This is useful as the underlying blocks have very cryptic names and you might not want to use them most of the time.

**Version Control**

The **Version Control** mechanism lets you check the valid connection between two multicore or multiprocessor systems. The version is a simple number that increments with every change of the Bus to Vector block. If a compiled object of a node is combined with a different version of the compiled code of another node, the version check detects any deviations in the dimensions or in the order of signals, and displays deviations found on the interface.

**Note**

To see version deviations, add an appropriate instrument to your ControlDesk layout and connect it to the `dsLabelInterfaceVersionCompare/Out1` variable of the Vector to Bus block. *True* indicates that the versions are identical. However, if you select the TRC exclusions checkbox, this variable is not available, and therefore cannot be activated via ControlDesk.

**Add Version Control** Lets you enable version control, which will generate additional blocks within the LabelInterface.

**Add Version Control Outport** Lets you route the version control information to an outport to make it available within the model.

**Current Version** Displays the current version. It increases automatically each time you generate LabelInterface blocks.

**Vector Packaging****Note**

This feature is not available in the ASM version of the LabelInterface library.

**Disabled** Lets you deactivate vector packaging.

**Changed signals (lossless)** Lets you minimize the data transfer on interprocessor communication. This feature sends only the altered signals to the receiving node.

**Downsampling (lossy)** Lets you minimize the data transfer on interprocessor communication. The feature splits the signals into smaller data chunks (the chunk size is defined by the vector size). Only one chunk per simulation step is transferred. If a signal has been altered, and the altered data is not contained in the chunk transferred, the signal is not updated until the chunk containing this signal is transferred.

**Vector Size** Lets you specify the number of signals to minimize the data transfer. The block will use only the vector size you have specified. If you select the **Changed signals (lossless)** checkbox and several signals change within one

sampling step, this will be shown in the output signals that are added to the Bus To Vector block.

**Init Steps** Lets you specify the number of initialization steps to be ignored. This is particularly useful if a large number of signals have changed, which might cause overflows.

**Terminate application on overflow** If you select this checkbox, the run time application is terminated if an overflow occurs.

**CurrentOverflows** Number of changed signals that cannot be transferred due to an overflow.

**MaxOverflows** Maximum number of signals that could not be transferred since the application was last started.

**CurrentSentSignals** The number of signals transferred within the current sampling step.

**MaxChangedSignals** Maximum number of changed signals used in the current sampling step.

## Fixed Vector Size

### Note

This feature is not available in the ASM version of the LabellInterface library.

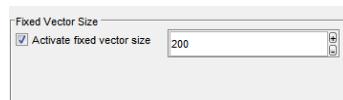
When you select this checkbox, an improved combination of different compiled nodes is used, which is particularly useful when working with variants in the Workflow Management (WFM). This option also lets you specify the number and order of all signals. Even when you remove a signal from the incoming bus, the signal is added to the vector, using a dummy value. You cannot remove signals or change the order of signals while this checkbox is selected.

When you select the Vector Packaging options, the related vector size parameter is ignored and the fixed vector size defined here is used.

### Note

You must select the SelectedSignals checkbox on the Signal page to activate the Fixed Vector Size option.

To make sure that the vector remains unchanged, you need to specify a maximum vector size.



## Related topics

### Basics

Description of Bus2Vector Block..... 140

## Signal Page

**Purpose**

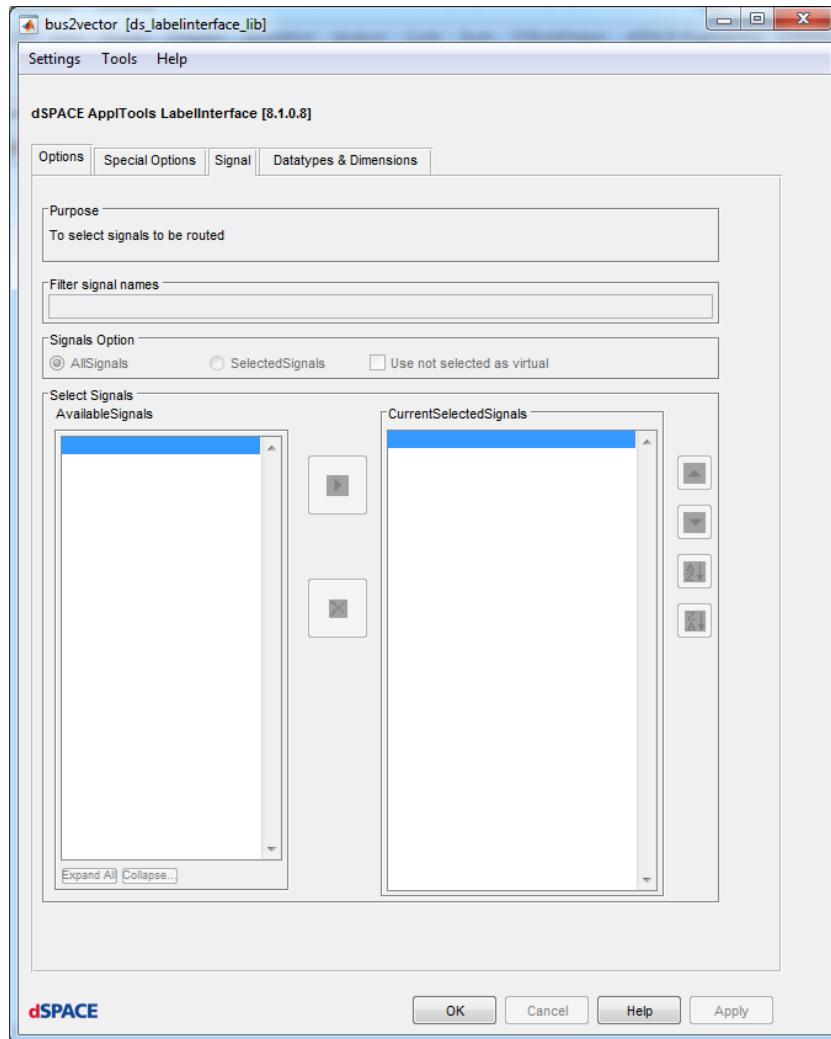
To select some or all signals from the input bus structure.

**Description**

The Signal page lets you select special signals from the bus without having to route all signals through the blocks.

**Tip**

To improve the performance of the IPC connection, you are recommended to minimize the number of signals that pass through the IPC block. See also [Datatypes & Dimensions Page](#) on page 148 for the Set all Boolean signals to “zipped” option.



---

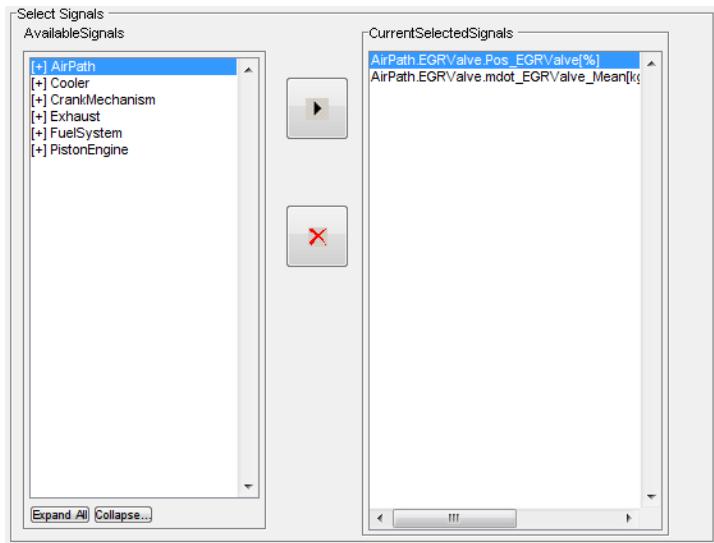
<b>Filter signal names</b>	Lets you filter the tree view of the bus structure. To be able to handle the signals, you must add them to the <b>CurrentSelectedSignals</b> list.
----------------------------	--

---

<b>Signals option</b>	The following options are available:
	<b>AllSignals</b> Lets you route all signals through the blocks.
	<b>SelectedSignals</b> Lets you copy the signals from the tree view to the list of selected signals.
	<b>Use not selected as virtual</b> Lets you create exactly the same bus as the input bus in the Vector to Bus block. However, only the selected signals are routed through the Bus to Vector block. All other signals are added to the Vector to Bus block with constant values (refer to <a href="#">Datatypes &amp; Dimensions Page</a> on page 148).

---

<b>Select signals</b>	Lets you add the signals to the <b>CurrentSelectedSignals</b> list. Only the selected signals are routed through the blocks.
-----------------------	--



**Available Signals** Lets you add the signals to the **CurrentSelectedSignals** list.

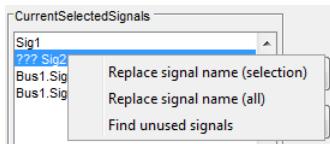
**CurrentSelectedSignals** Lets you view all signals that will be integrated when generating the LabellInterface.

**X** Lets you delete signals that are no longer required.

---

<b>Replace signal name (context menu)</b>	If the input vector was changed, the currently selected signals might become invalid, which is indicated by the ???-prefix. For some use cases it is important to keep the bus structure. If invalid signals exist, they are fed by a Constant block with zeros. Nevertheless a warning is issued during the generation of the B2V and V2B blocks.
---	--

A context menu opens and lets you replace an invalid signal name by a valid one:



**Replace signal name (selection)** Lets you replace the signals selected in the list. A dialog opens to let you select the signal to be replaced as well as the new signal. The new signal must be part of the input bus.

**Replace signal name (all)** Lets you replace all signals listed.

**Find unused signals** Lets you scan the model for signals selected in the list that are no longer used in the model after they have been passed through the Vector To Bus block.

#### Note

To use this option, you must first generate the blocks.

#### Related topics

#### Basics

Description of Bus2Vector Block..... 140

## Datatypes & Dimensions Page

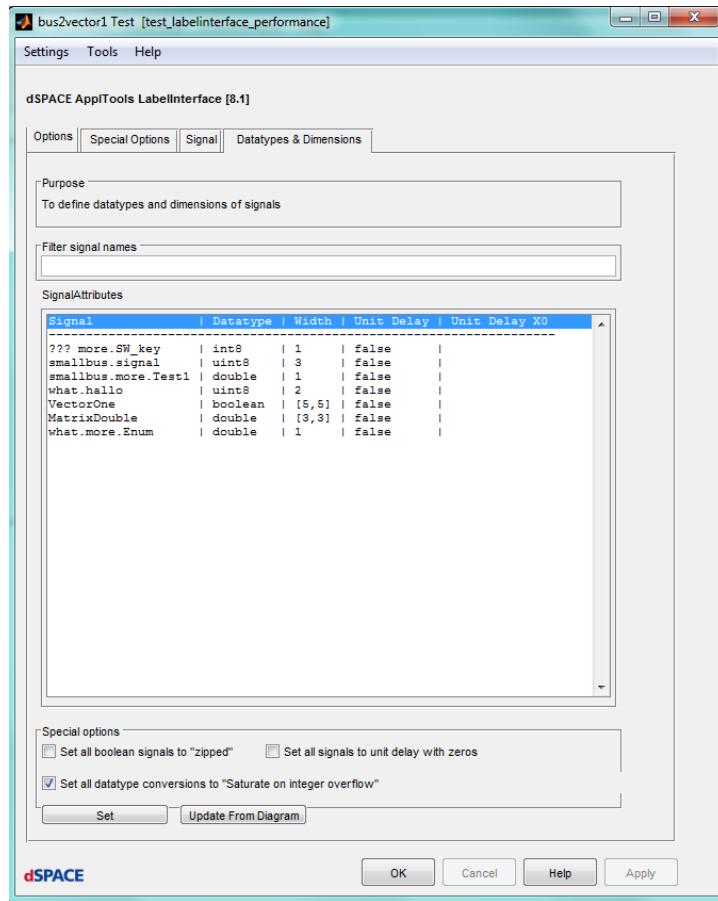
**Purpose** To set the dimensions and data types of the selected signals.

**Description** Because it is not possible to route signals of different data types through IPC connections or other blocks, the data type has to be converted to double and back to the original data type.

#### Note

To convert to a vector, you need to specify the exact dimension of each signal.

You must define correct data types and dimensions before disrupting the connections with the model to recreate the bus structure.

**Filter signal names**

Lets you filter for signal names. All signals matching the defined filter are selected in the SignalAttributes list.

**SignalAttributes**

Lets you specify the attributes for all signals at the same time. You can multiselect lines to set the signal attributes.

Signal	Datatype	Width	Unit Delay	Unit Delay X0
??? more.SW_key	auto	1	false	
smallbus.signal	uint8	3	true	1
smallbus.more.Test1	double	1	false	
what.hallo	uint8	2	true	1
VectorOne	boolean	[vectorOne, vectorOne]	false	
MatrixDouble	double	[3,3]	false	
what.more.Enum	Color	2	false	[Co

## Special options

The Special Options checkboxes let you select the following:

**Set all Boolean signals to “zipped”** Lets you zip all Boolean signals to a unit32 signal. This special feature of the LabellInterface library lets you minimize the number of signals routed through the blocks. To set signals to Boolean, you can either select signals individually or set all signals to Boolean by selecting Set all Boolean signals to “zipped”.

### Note

When the dimension of a Boolean signal is defined dynamically by a MATLAB workspace variable, the zip mechanism is deactivated. However, the zip mechanism only works reliably with the vector size that is defined during generation, because the mechanism requires exact dimensions. Because dimensions might change whenever a diagram is updated, you need to define exact dimensions to use the zip mechanism.

**Set all signals to unit delay with zeros** Lets you force a unit delay in each signal line. The default value is a zeros matrix which has the same dimensions as the signal.

### Tip

If a unit delay is activated and specified in the signals attributes table, this setting is applied as the master setting.

**Set all data type conversions to “Saturate on integer overflow”** Lets you change the settings of all data type conversions used inside the blocks. If you activate/deactivate this feature, it will affect the generated run-time code.

**Set** Lets you open the DataType and Dimensions Options dialog.

**Update From Diagram** Lets you use the data types and dimensions of the model.

### Note

To use the Update From Diagram checkbox, your model must be precompilable via the MATLAB **Update Diagram** command.

## Related topics

### Basics

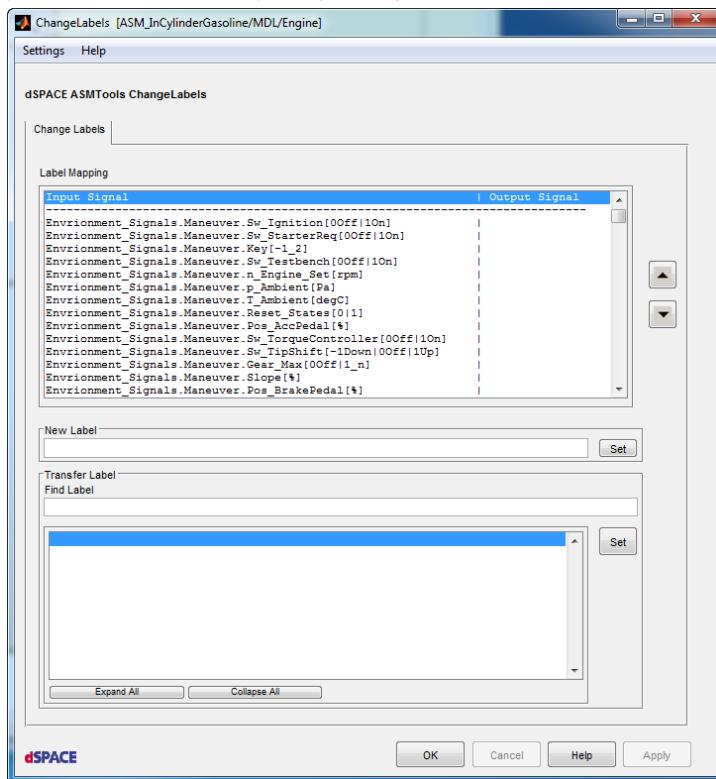
Description of Bus2Vector Block.....	140
--------------------------------------	-----

# Change Labels Block

## Description of ChangeLabels Block

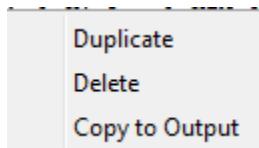
### Purpose

This block lets you change single label names in a bus. This feature can be particularly useful if you want to update a Simulink bus but the model parts affected by the update cannot adapt to the modifications involved. In this case, you can recreate a bus by using its original name.



### Label Mapping

This list shows the input labels on the left and the output labels on the right. With the up and down buttons on the right you can change the order of the output signals. Multiselection is possible.



The context menu on the Label Mapping list provides options to handle the signals list.

- **Duplicate:** Duplicates a label in the bus. This lets you use the signal *as is* and with a different name in the output bus if required.
- **Delete:** Deletes the signal from the output bus.
- **Copy to Output:** Uses the same label in the output bus.

---

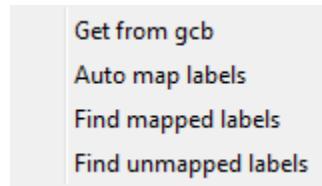
**New Label**

You can type a new label into the New Label edit field. When you click the Set button, the new label is assigned to the currently selected input signal in the Label Mapping list.

---

**Transfer Label**

Any existing bus can be loaded and the signal names in the bus structure can be used to define the output signal names of the Change Label block. This list displays all mappable labels. When you click the Set button next to the Select Label list, a highlighted label in the Select Label list is mapped to a selected input label in the Label Mapping list. Multiselection is not possible.



The context menu on the Select Label list provides the following options:

- **Get from gcb:** Import labels from the currently selected block (GCB) that is driven by a BusSelector block.
- **Auto map labels:** Automatically maps all selectable labels to output labels if
  - The full path of a selectable label is found in an input label or
  - The unique name of a selectable label matches the unique name of an input label.
- **Find mapped labels:** Finds and selects all mapped labels.
- **Find unmapped labels:** Finds and selects all unmapped labels.

**Find Label** Lets you find specific labels in the selectable labels list. Type a regular expression into the Find Label edit field and press the Enter key to find and highlight labels that match the regular expression string.

# ASM Utils

## Where to go from here

## Information in this section

Basics on the ASM Utils Library.....	153
The ASM Utils library serves as a basis for common functionalities in ASM models.	
For Iterator Subsystem.....	155
Transfer Functions.....	157
ModelDesk Interface.....	160
Unit Conversion.....	169
Matrix Operations.....	170
Miscellaneous.....	180
Extended Look-Up Table.....	188
Replace Scopes.....	192
Open Experiment.....	193
Operator.....	193
Block Replacement.....	195
Code Generation.....	196
Random Numbers.....	197

## Basics on the ASM Utils Library

### Introduction

The ASM Utils library serves as a basis for common functionalities in ASM models.

## How to Open the ASM Utils Library

### Objective

The ASM Utils library serves as a basis for common functionalities in ASM models. You can open the library in MATLAB.

**Method****To open the ASM Utils library**

- 1 In the MATLAB Command Windows, type `ASM_Utils.lib`.

**Result**

The ASM Utils library opens.



# For Iterator Subsystem

## Where to go from here

## Information in this section

### [Forward Euler](#)..... 155

The Forward Euler block performs the integration based on the forward Euler method.

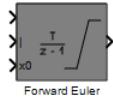
### [For Iterator Delay](#)..... 156

The ForIteratorDelay block applies a sufficient delay when transporting signals out of a Simulink for iterator subsystem.

## Forward Euler

### Introduction

The **Forward\_Euler** block performs the integration based on the forward Euler method. The block is used specifically for the local oversampling method as described in [Local Subsystem Oversampling](#) on page 63.



### Imports

The following table shows the imports:

Name	Unit	Description
In	[]	Integrand
Init	[]	Initial value
Reset	[0 1]	Resets the states: ▪ 0: No influence on the integrator ▪ 1: Reset of states

### Outports

The following table shows the outports:

Name	Unit	Description
Out	[]	Integration Result

**Parameters**

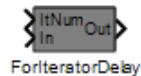
The following table shows the parameters:

Name	Unit	Description
Library for licensecheck	[]	Specifies the ASM library to be used for the license check of the underlying S-function. The <i>Inherited from Block</i> option can be used only if the block is used in an ASM block.
LimState	[1 2 3 4]	Limitation method: <ul style="list-style-type: none"> <li>▪ 1: Upper and lower</li> <li>▪ 2: Upper</li> <li>▪ 3: Lower</li> <li>▪ 4: None</li> </ul>
LowLim	[]	Lower limit
SampleTime	[s]	Subsystem sample time
UpLim	[]	Upper limit

## For Iterator Delay

**Introduction**

The ForIteratorDelay block applies a sufficient delay when transporting signals out of a Simulink for iterator subsystem. The block is used specifically for the local oversampling method as described in [Numerical Aspects of ASMs](#) on page 63

**Imports**

The following table shows the imports:

Name	Unit	Description
In	[]	Variable to be delayed
ItNum	[]	Current iteration number of the oversampled subsystem

**Outports**

The following table shows the outports:

Name	Unit	Description
Out	[]	Delayed variable

**Parameters**

The following table shows the parameters:

Name	Unit	Description
ItNum	[]	Number of iterations

## Transfer Functions

### Where to go from here

### Information in this section

[First Order Dynamics](#).....157

The to FirstOrderDynamics block represents a first-order lag element.

[Second Order Dynamics](#).....158

The SecondOrderDynamics block represents a second-order lag element.

[Memory With Reset](#).....159

The MemoryWithReset block applies a resettable memory with parameterizable initial conditions.

## First Order Dynamics

### Introduction

The to FirstOrderDynamics block represents a first-order lag element.



The transfer function in the Laplace domain is

$$G(s) = \frac{1}{Ts + 1}$$

where T is the time constant.

**Imports**

The following table shows the imports:

Name	Unit	Description
In	[]	Input variable
Reset	[0 1]	Resets the states: <ul style="list-style-type: none"> <li>▪ 0: No influence on the integrator</li> <li>▪ 1: Resetting of states</li> </ul>
TimeConstant	[s]	Time constant T

**Outports**

The following table shows the outports:

Name	Unit	Description
Out	[]	Output signal

**Parameters**

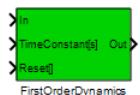
The following table shows the parameters:

Name	Unit	Description
Ts	[s]	Simulation step size
x0	[]	Initial condition

## Second Order Dynamics

**Introduction**

The SecondOrderDynamics block represents a second-order lag element.



The transfer function in the Laplace domain is

$$G(s) = \frac{1}{T^2 s^2 + 2dTs + 1}$$

where:

$T$  is the time constant

$d$  is the damping coefficient

The block applies a second order delay to the input signal with a fixed damping coefficient of  $\frac{1}{\sqrt{2}}$  and a predefined eigen angular frequency  $\omega$  with  $\omega = \frac{1}{T}$ .

**Imports**

The following table shows the imports:

Name	Unit	Description
In	[]	Input variable
Reset	[0 1]	Resets the states: <ul style="list-style-type: none"> <li>▪ 0: No influence on the integrator</li> <li>▪ 1: Resetting of states</li> </ul>
wn	[]	Eigen angular frequency ( $w=1/T$ )

**Outports**

The following table shows the outports:

Name	Unit	Description
Out	[]	Output signal
Out_dt	[]	Derivative output signal ( $dOut/dt$ )

**Parameters**

The following table shows the parameters:

Name	Unit	Description
Ts	[s]	Simulation step size

## Memory With Reset

**Introduction**

The MemoryWithReset block applies a resettable memory with parameterizable initial conditions.

**Imports**

The following table shows the imports:

Name	Unit	Description
In	[]	Input signal
Reset	[0 1]	Resets the states: <ul style="list-style-type: none"> <li>▪ 0: No influence on the integrator</li> <li>▪ 1: Resetting of states</li> </ul>

**Outports**

The following table shows the outports:

Name	Unit	Description
Out	[]	Delayed signal

**Parameters**

The following table shows the parameters:

Name	Unit	Description
x0	[]	Initial condition

## ModelDesk Interface

**Introduction**

The ModelDesk Interface contains blocks that are needed to support the parameterization and communication with ModelDesk.

**Where to go from here****Information in this section**

<a href="#">Maneuver and Reset Control</a> .....	160
<a href="#">Multi-Instance Blocks</a> .....	165
<a href="#">ModelDesk Parameter Group</a> .....	166

## Maneuver and Reset Control

**Where to go from here****Information in this section**

<a href="#">Overview of the Maneuver and Reset Control</a> .....	161
If your model includes the maneuver and reset control blocks of the ASM Utils library, you can start and stop a maneuver and reset the model states via ModelDesk.	
<a href="#">Maneuver Start</a> .....	162
The maneuver start parameter is set to start a vehicle dynamics driving maneuver or an engine driving cycle.	

**Maneuver Stop.....** 163

The maneuver stop parameter is set to stop a vehicle dynamics driving maneuver or an engine driving cycle.

**Maneuver State.....** 164

The maneuver state parameter is read by ModelDesk. The information about the state of the maneuver is displayed in ModelDesk's Message Viewer.

**Reset.....** 164

The reset parameter is set to reset the states of the simulation model.

## Overview of the Maneuver and Reset Control

### Description

If your model includes the maneuver and reset control blocks of the ASM Utils library, you can start and stop a maneuver and reset the model states via ModelDesk. See [Basics of Controlling Maneuvers and Driving Cycles \(ModelDesk Scenario Creation\)](#).

To use this functionality, the maneuver start and stop, and the reset parameter have to be set in ModelDesk. In addition, ModelDesk reads the current maneuver state from the simulation and displays the information in the Message Viewer.

The maneuver and reset control blocks can be used for the following ASM model components:

- ASM VehicleDynamics
- ASM Engine
- ASM Electric Components

If a model, for example, a virtual vehicle with engine and vehicle dynamics, includes several instances of these control blocks, ModelDesk can only control the maneuver and reset of one component. Only the component with the highest priority is controlled by ModelDesk.

The following level of priority account for the different components:

1. ASM VehicleDynamics
2. ASM Engine
3. ASM Electric Components

### Related topics

#### Basics

[Basics of Controlling Maneuvers and Driving Cycles \(ModelDesk Scenario Creation\)](#)

## Maneuver Start

### Description

The maneuver start parameter is set to start a vehicle dynamics driving maneuver or an engine driving cycle.

For a detailed description of the variable for an ASM VehicleDynamics application refer to [Maneuver \(ASM Environment Reference\)](#) and for an ASM Engine application to [Maneuver Control \(ASM Drivetrain Basic Reference\)](#).

The following illustration shows the Simulink representation of the MANEUVER\_START block.



### Note

To control the maneuver start in a Simulink offline simulation via ModelDesk, the MDLDCTRL\_ManeuverStart parameter must be set to zero.

### Outports

The following table shows the outports:

Name	Unit	Description
ManeuverStart	[]	Maneuver start control signal

### Parameters

The following table shows the parameters:

Name	Unit	Description
MDLDCTRL_ManeuverStart	[]	Maneuver start parameter, set via ModelDesk

### Related topics

### References

[Maneuver \(ASM Environment Reference\)](#)  
[Maneuver Control \(ASM Drivetrain Basic Reference\)](#)

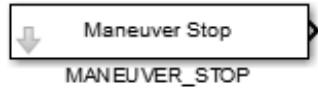
## Maneuver Stop

### Description

The maneuver stop parameter is set to stop a vehicle dynamics driving maneuver or an engine driving cycle.

For a detailed description of the variable for an ASM VehicleDynamics application refer to [Maneuver \(ASM Environment Reference\)](#) and for an ASM Engine application to [Maneuver Control \(ASM Drivetrain Basic Reference\)](#).

The following illustration shows the Simulink representation of the MANEUVER\_STOP block.



### Note

To control the maneuver stop in a Simulink offline simulation via ModelDesk, the MDLDCTRL\_ManeuverStop block must be set to zero.

### Outports

The following table shows the outports:

Name	Unit	Description
ManeuverStop	[]	Maneuver stop control signal

### Parameters

The following table shows the parameters:

Name	Unit	Description
MDLDCTRL_ManeuverStop	[]	Maneuver stop parameter, set via ModelDesk

### Related topics

### References

[Maneuver \(ASM Environment Reference\)](#)  
[Maneuver Control \(ASM Drivetrain Basic Reference\)](#)

## Maneuver State

<b>Description</b>	The maneuver state parameter is read by ModelDesk. The information about the state of the maneuver is displayed in ModelDesk's Message Viewer.
--------------------	--

For a detailed description of the maneuver state and its numeric values for an ASM VehicleDynamics application, refer to [Maneuver \(ASM Environment Reference\)](#) and for an ASM Engine application to [Maneuver Control \(ASM Drivetrain Basic Reference\)](#).

The following illustration shows the Simulink representation of the MANEUVER\_STATE block.



### Imports

The following table shows the imports:

Name	Unit	Description
ManeuverState	[]	Maneuver state, read by ModelDesk

### Related topics

#### References

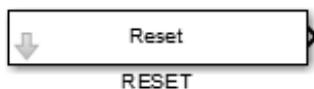
Maneuver Start.....	162
Maneuver Stop.....	163
Reset.....	164

## Reset

### Description

The reset parameter is set to reset the states of the simulation model.

The following illustration shows the Simulink representation of the Reset block.



### Outport

The following table shows the outports:

Name	Unit	Description
Reset	[]	Maneuver reset signal

**Parameters**

The following table shows the parameters:

Name	Unit	Description
MDLDCTRL_Reset	[]	Reset parameter, set via ModelDesk

**Related topics****References**

Maneuver Start.....	162
Maneuver State.....	164
Maneuver Stop.....	163

## Multi-Instance Blocks

### Multi-Instance Overview

**Introduction**

The MultiInstanceOverview block opens an overview user interface for the multi-instances that are included in the active Simulink system. The blocks triggers the `asm_multiinstance` function. Refer to [asm\\_multiinstance](#) on page 201.

For more information on the ASM multi-instance feature, refer to [Multi-Instance](#) on page 53.

**Related topics****Basics**

<a href="#">asm_multiinstance</a> .....	201
<a href="#">Parameterizing ASM Blocks of the Same Type (ModelDesk Parameterizing)</a>	

# ModelDesk Parameter Group

## Where to go from here

## Information in this section

<a href="#">ModelDesk Parameter Group Overview</a>	166
The ModelDeskParaGroupOverview block opens an overview user interface for the ModelDesk parameter group blocks that are included in the active Simulink system.	
<a href="#">ModelDesk Parameter Group Block</a>	167
To specify a group name for structuring the view of the parameter set in ModelDesk.	
<a href="#">ModelDesk Write Access</a>	167
The MODEL_DESK_WRITE_ACCESS block can set this flag during a parameter-write operation from ModelDesk. The flag can be used in the model.	
<a href="#">ModelDesk Main Component Block</a>	168
The ModelDesk_MC_ID block marks an ASM main component in the Simulink model, so ModelDesk can recognize it in the next Update Model process.	

## ModelDesk Parameter Group Overview

### Description

The ModelDeskParaGroupOverview block opens an overview user interface for the ModelDesk parameter group blocks that are included in the active Simulink system.



### Related topics

#### Basics

[Basics on Grouping the View of a Parameter Set \(ModelDesk Parameterizing\)](#)

## ModelDesk Parameter Group Block

---

<b>Purpose</b>	To specify a group name for structuring the view of the parameter set in ModelDesk.
	
<b>Description</b>	The block is used to mark subsystems of an ASM. Each subsystem that contains the block with the same group name is grouped under this name in ModelDesk. Refer to <a href="#">Basics on Grouping the View of a Parameter Set (ModelDesk Parameterizing)</a> .

---

<b>Parameters</b>	The following table shows the parameters:						
	<table border="1"> <thead> <tr> <th>Name</th> <th>Unit</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Group name</td> <td>[ - ]</td> <td>Lets you specify a group name.</td> </tr> </tbody> </table>	Name	Unit	Description	Group name	[ - ]	Lets you specify a group name.
Name	Unit	Description					
Group name	[ - ]	Lets you specify a group name.					

---

<b>Related topics</b>	HowTos
	<a href="#">How to Extend the Simulink Model for Grouping the View (ModelDesk Parameterizing)</a>

## ModelDesk Write Access

---

<b>Introduction</b>	The MODEL_DESK_WRITE_ACCESS block can set this flag during a <b>parameter-write</b> operation from ModelDesk. The flag can be used in the model.
	

---

<b>Outports</b>	The following table shows the outports:						
	<table border="1"> <thead> <tr> <th>Name</th> <th>Unit</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>ModelDesk_WriteAccess</td> <td>[]</td> <td>Flag is set to nonzero values if ModelDesk downloads parameters to the model.</td> </tr> </tbody> </table>	Name	Unit	Description	ModelDesk_WriteAccess	[]	Flag is set to nonzero values if ModelDesk downloads parameters to the model.
Name	Unit	Description					
ModelDesk_WriteAccess	[]	Flag is set to nonzero values if ModelDesk downloads parameters to the model.					

**Parameters**

The following table shows the parameters:

Name	Unit	Description
ModelDesk_WriteAccess	[]	ModelDesk write access flag

## ModelDesk Main Component Block

**Description**

The ModelDesk\_MC\_ID\_ block marks an ASM main component in a Simulink model, so ModelDesk can recognize it in the next Update Model process.

You can parameterize the block to mark the following ASM main components:

- Drivetrain
- DrivetrainBasic
- ElectricComponents
- Engine
- EngineDiesel
- EngineGasoline
- Environment
- EnvironmentBasic
- SoftECU
- Trailer
- VehicleDynamics
- VehicleDynamicsBasic

**Note**

Do not change the ModelDesk\_MC\_ID blocks in the ASM demo models. This will change the navigation tree of the corresponding ModelDesk project.

The following illustration shows the ModelDesk\_MC\_ID\_ block configured for the EngineGasoline component.



---

**Related topics****Basics**

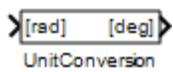
Migrating ASM Models..... 67

# Unit Conversion

## Unit Conversion

**Introduction**

The UnitConversion block provides the common unit conversions via a drop down list. The content of the block is created depending on the requirements of the selected conversion.

**Imports**

The following table shows the imports:

Name	Unit	Description
Input	[]	Signal with input unit

**Outports**

The following table shows the outports:

Name	Unit	Description
Output	[]	Signal with output unit

**Parameters**

The following table shows the parameters:

Name	Unit	Description
Selection	[]	Unit conversion to be applied

# Matrix Operations

## Where to go from here

## Information in this section

<a href="#">Unit Vector</a>	171
The UNIT_VECTOR block calculates the unit vector for an arbitrary vector.	
<a href="#">Unit Vectors</a>	171
The UNIT_VECTORS block calculates the unit vector in the y-direction of the newly rotated coordinate system.	
<a href="#">Transpose</a>	172
The TRANSPOSE block transposes a matrix.	
<a href="#">Skew Symmetric Matrix</a>	173
The SKEW_SYMMETRIC_MATRIX block applies the skew symmetric operator to an input vector.	
<a href="#">Rotation Matrix Angles</a>	173
The ROTATION_MATRIX_ANGLES block provides the rotation matrix with the orientation order gamma, beta and alpha (Cardan angles).	
<a href="#">Rotation Matrix Angles Small</a>	174
The ROTATION_MATRIX_ANGLES_SMALL block approximates the rotation matrix with the orientation order gamma, beta, and alpha (Cardan angles) for small angles for efficient calculations.	
<a href="#">Rotation Matrix to Angles</a>	174
The ROTATION_MATRIX_2_ANGLES block calculates the Cardan angles based on a given rotation matrix.	
<a href="#">Transform GammaAlphaBeta</a>	175
The Transform_GammaAlphaBeta block leads to a homogeneous transformation with the rotation order Gamma-Alpha-Beta, e.g., for wheel animation in MotionDesk.	
<a href="#">Cross Product 3x1</a>	176
The CROSS_PRODUCT block calculates the cross product of two 3x1 vectors.	
<a href="#">Cross Product 6x1</a>	176
The CROSS_PRODUCT_2 block calculates the cross product of two 6x1 vectors.	
<a href="#">Dot Product</a>	177
The DOT_PRODUCT block calculates the dot product of two vectors.	
<a href="#">Calc Caster Left</a>	177
The CALC_CASTER_LEFT block calculates the caster angle (rotation around the y-axis) based on the superpositions of the kinematic and compliance results.	

[Coordinate Transformation B to A.....](#) 178

The COORDINATE\_TRANSFORMATION\_B\_TO\_A block transforms input vectors from a coordinate system B to a coordinate system A.

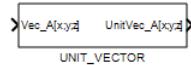
[Inverse Coordinate Transformation A to B.....](#) 179

The INVERSE\_COORDINATE\_TRANSFORMATION\_A\_TO\_B block executes an inverse transformation on input vectors from a coordinate system A to a coordinate system B.

## Unit Vector

### Introduction

The UNIT\_VECTOR block calculates the unit vector for an arbitrary vector.



### Imports

The following table shows the imports:

Name	Unit	Description
Vec_A	[x;y;z]	Input vector

### Outports

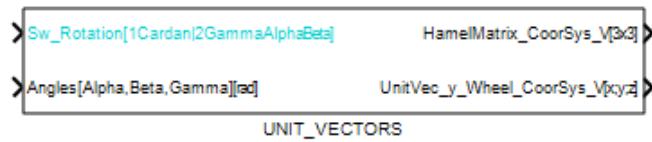
The following table shows the outports:

Name	Unit	Description
UnitVec_A	[x;y;z]	Unit vector

## Unit Vectors

### Introduction

The UNIT\_VECTORS block calculates the unit vector in the y-direction of the newly rotated coordinate system. The rotation is applied with respect to the inputs signals.



## Imports

The following table shows the imports:

Name	Unit	Description
Angles[Alpha,Beta,Gamma]	[rad]	Radian measure of alpha, beta, and gamma
Sw_Rotation	[1 2]	Switch for Cardan rotation: <ul style="list-style-type: none"><li>▪ 1: Cardan</li><li>▪ 2: GammaAlphaBeta</li></ul>

## Outports

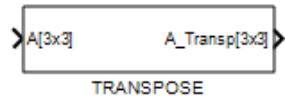
The following table shows the outputs:

Name	Unit	Description
HamelMatrix_CoorSys_V	[3x3]	Hamel Matrix
UnitVec_y_Wheel_CoorSys_V	[x;y;z]	Unit vector in the y-direction

## Transpose

## Introduction

The TRANSPOSE block transposes a matrix.



## Imports

The following table shows the imports:

Name	Unit	Description
A	[3x3]	Input of the original matrix

## Outports

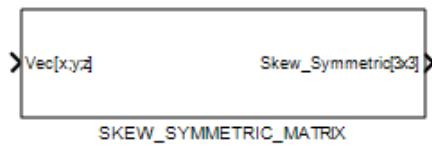
The following table shows the outputs:

Name	Unit	Description
A_Transp	[3x3]	Output of the transposed matrix

## Skew Symmetric Matrix

### Introduction

The SKEW\_SYMMETRIC\_MATRIX block applies the skew symmetric operator to an input vector.



### Imports

The following table shows the imports:

Name	Unit	Description
Vec	[x;y;z]	Input vector

### Outports

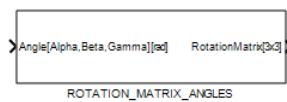
The following table shows the outports:

Name	Unit	Description
Skew_Symmetric	[3x3]	Matrix of the asymmetrical distribution

## Rotation Matrix Angles

### Introduction

The ROTATION\_MATRIX\_ANGLES block provides the rotation matrix with the orientation order gamma, beta, and alpha (Cardan angles).



### Imports

The following table shows the imports:

Name	Unit	Description
Angle[Alpha,Beta,Gamma]	[rad]	Radian measure of alpha, beta, and gamma

### Outports

The following table shows the outports:

Name	Unit	Description
RotationMatrix	[3x3]	Rotation matrix

## Rotation Matrix Angles Small

### Introduction

The ROTATION\_MATRIX\_ANGLES\_SMALL block approximates the rotation matrix with the orientation order gamma, beta, and alpha (Cardan angles) for small angles with efficient calculations.



### Imports

The following table shows the imports:

Name	Unit	Description
Angle[Alpha,Beta,Gamma]	[rad]	Radian measure of alpha, beta, and gamma
Sw_Rotation	[1 2]	Switch for Cardan rotation: <ul style="list-style-type: none"> <li>▪ 1: Cardan</li> <li>▪ 2: GammaAlphaBeta</li> </ul>

### Outports

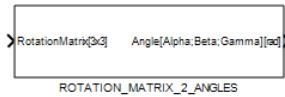
The following table shows the outports:

Name	Unit	Description
RotationMatrix	[3x3]	Rotation matrix

## Rotation Matrix to Angles

### Introduction

The ROTATION\_MATRIX\_2\_ANGLES block calculates the Cardan angles based on a given rotation matrix.



### Imports

The following table shows the imports:

Name	Unit	Description
RotationMatrix	[3x3]	Rotation matrix

**Outports**

The following table shows the outports:

Name	Unit	Description
Angle[Alpha;Beta;Gamma]	[rad;rad;rad]	Radian of alpha, beta, and gamma

## Transform GammaAlphaBeta

**Introduction**

The Transform\_GammaAlphaBeta block leads to a homogeneous transformation with the rotation order Gamma-Alpha-Beta, e.g., for wheel animation in MotionDesk.

**Imports**

The following table shows the imports:

Name	Unit	Description
beta	[deg]	Beta angle
ey	[x;y;z]	Unit vector in the y-direction of the rotated coordinate system
r	[x;y;z]	Position vector to the origin of the rotated coordinate system
T_In	[4x4]	Homogeneous transformation input

**Outports**

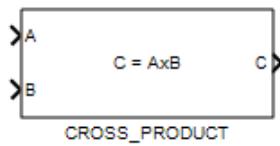
The following table shows the outports:

Name	Unit	Description
T_Out	[4x4]	Homogeneous transformation output

## Cross Product 3x1

### Introduction

The CROSS\_PRODUCT 3x1 block calculates the cross product of two 3x1 vectors.



### Imports

The following table shows the imports:

Name	Unit	Description
A	[3x1]	First input vector
B	[3x1]	Second input vector

### Outports

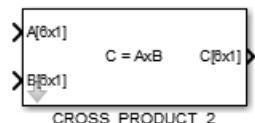
The following table shows the outports:

Name	Unit	Description
C	[3x1]	Output vector

## Cross Product 6x1

### Introduction

The CROSS\_PRODUCT\_2 block calculates the cross product of two 6x1 vectors.



### Imports

The following table shows the imports:

Name	Unit	Description
A	[6x1]	First input vector
B	[6x1]	Second input vector

**Outports**

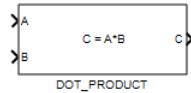
The following table shows the outports:

Name	Unit	Description
C	[6x1]	Output vector

## Dot Product

**Introduction**

The DOT\_PRODUCT block calculates the dot product of two vectors.

**Imports**

The following table shows the imports:

Name	Unit	Description
A	[3x1]	First input vector
B	[3x1]	Second input vector

**Outports**

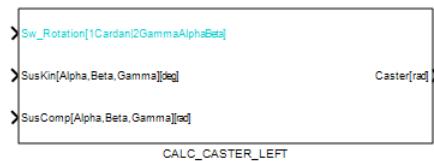
The following table shows the outports:

Name	Unit	Description
C	[3x1]	Output vector

## Calc Caster Left

**Introduction**

The CALC\_CASTER\_LEFT block calculates the caster angle (rotation around the y-axis) based on the superpositions of the kinematic and compliance results.



**Imports**

The following table shows the imports:

Name	Unit	Description
SusComp[Alpha,Beta,Gamma]	[rad]	Wheel angle rotation based on the compliance effects.
SusKin[Alpha,Beta,Gamma]	[deg]	Wheel angle rotation based on the kinematics effects.
Sw_Rotation	[1 2]	Switch for Cardan rotation: ▪ 1: Cardan ▪ 2: GammaAlphaBeta

**Outports**

The following table shows the outports:

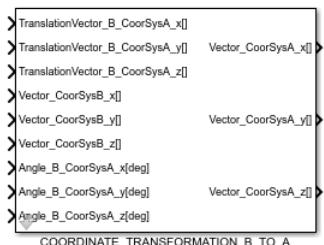
Name	Unit	Description
Caster	[rad]	Caster angle

## Coordinate Transformation B to A

**Introduction**

The COORDINATE\_TRANSFORMATION\_B\_TO\_A block transforms input vectors from a coordinate system B to vectors of a coordinate system A. The transformation is done by rotating the vectors and then translating them. The rotation order is: Z - Y' - X"

This block supports a dynamic amount of inputs.

**Imports**

The following table shows the imports:

Name	Unit	Description
Angle_B_CoorSysA_x	[deg]	The rotation angle of coordinate system B in the counterclockwise direction about the x-axis of coordinate system A.
Angle_B_CoorSysA_y	[deg]	The rotation angle of coordinate system B in the counterclockwise direction about the y-axis of coordinate system A.

Name	Unit	Description
Angle_B_CoorSysA_z	[deg]	The rotation angle of coordinate system B in the counterclockwise direction about the z-axis of coordinate system A.
TranslationVector_B_CoorSysA_x	[]	The translation in the x-direction from the origin of coordinate system A to the origin of coordinate system B.
TranslationVector_B_CoorSysA_y	[]	The translation in the y-direction from the origin of coordinate system A to the origin of coordinate system B.
TranslationVector_B_CoorSysA_z	[]	The translation in the z-direction from the origin of coordinate system A to the origin of coordinate system B.
Vector_CoorSysB_x	[]	The x-component of the vector to be transformed in coordinate system B.
Vector_CoorSysB_y	[]	The y-component of the vector to be transformed in coordinate system B.
Vector_CoorSysB_z	[]	The z-component of the vector to be transformed in coordinate system B.

## Outports

The following table shows the outports:

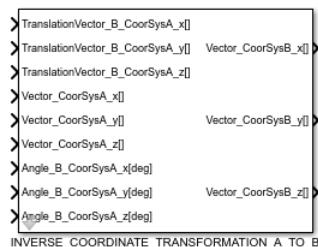
Name	Unit	Description
Vector_CoorSysA_x	[]	The x-component of the transformed vector in coordinate system A.
Vector_CoorSysA_y	[]	The y-component of the transformed vector in coordinate system A.
Vector_CoorSysA_z	[]	The z-component of the transformed vector in coordinate system A.

## Inverse Coordinate Transformation A to B

### Introduction

The INVERSE\_COORDINATE\_TRANSFORMATION\_A\_TO\_B block executes an inverse transformation on input vectors from a coordinate system A to a coordinate system B. The transformation is done by subtracting the translation from the input vectors and then rotating them. The rotation order is Z - Y' - X".

This block supports a dynamic amount of inputs.



**Imports**

The following table shows the imports:

Name	Unit	Description
Angle_B_CoorSysA_x	[deg]	The rotation angle of coordinate system B in the counterclockwise direction about the x-axis of coordinate system A.
Angle_B_CoorSysA_y	[deg]	The rotation angle of coordinate system B in the counterclockwise direction about the y-axis of coordinate system A.
Angle_B_CoorSysA_z	[deg]	The rotation angle of coordinate system B in the counterclockwise direction about the z-axis of coordinate system A.
TranslationVector_B_CoorSysA_x	[]	The translation in the x-direction from the origin of coordinate system A to the origin of coordinate system B.
TranslationVector_B_CoorSysA_y	[]	The translation in the y-direction from the origin of coordinate system A to the origin of coordinate system B.
TranslationVector_B_CoorSysA_z	[]	The translation in the z-direction from the origin of coordinate system A to the origin of coordinate system B.
Vector_CoorSysA_x	[]	The x-component of the vector to be transformed in coordinate system A.
Vector_CoorSysA_y	[]	The y-component of the vector to be transformed in coordinate system A.
Vector_CoorSysA_z	[]	The z-component of the vector to be transformed in coordinate system A.

**Outports**

The following table shows the outports:

Name	Unit	Description
Vector_CoorSysB_x	[]	The x-component of the transformed vector in coordinate system B.
Vector_CoorSysB_y	[]	The y-component of the transformed vector in coordinate system B.
Vector_CoorSysB_z	[]	The z-component of the transformed vector in coordinate system B.

## Miscellaneous

### Where to go from here

### Information in this section

<a href="#">ASM Function Call Generator</a> .....	181
The ASM_FCN_CALL_GENERATOR generates function call events separated by a defined sample-based gap.	

<a href="#">Compare Value</a>	182
The compare_value block compares the current value of a signal with a mask parameter.	
<a href="#">Test Cycle</a>	184
The Driving Cycles block selects the test cycle for the current simulation by choosing an element from a list.	
<a href="#">Engine OP Selector</a>	185
The EngOP_Selector block selects a signal from the engine operating point vector.	
<a href="#">Flush Denormals to Zero</a>	185
You can activate flushing to zero during the compile process on PHS-bus-based systems (e.g., the DS1006).	
<a href="#">Select Maneuver</a>	186
The Select Maneuver block selects the maneuver type and maneuver control when the simulation starts via a drop-down menu.	
<a href="#">Variable Delay</a>	187
The block contains an s-function, which delays the input signal by a delay time specified.	

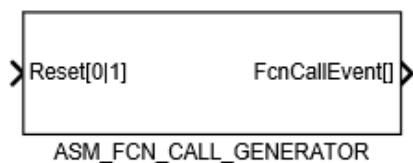
## ASM Function Call Generator

**Description** The ASM\_FCN\_CALL\_GENERATOR generates function call events separated by a defined sample-based gap.

When the Reset input is set to 1, the block immediately generates a function call event and then continues generating function call events with the defined sample-based gaps.

The NumSamples parameter specifies the number of time steps between two function call events. For example, if the step size of the model is set to 1 ms and the block parameter is set to 20, the block generates function call events every 20 ms.

The block lets you select the ASM license to be checked in a drop-down list.



**Imports**

The following table shows the imports:

Name	Unit	Description
Reset	[0 1]	Signal to reset the ASM_FCN_CALL_GENERATOR block.

**Outports**

The following table shows the outports:

Name	Unit	Description
FcnCallEvent	[]	Function call events generated depending on the NumSamples parameter.

**Parameters**

The following table shows the parameters:

Name	Unit	Description
NumSamples	[]	Number of time steps between two function call events.
Library for Licensecheck	[]	ASM license to be checked in the block.

## Compare Value

**Introduction**

The compare\_value block compares the current value of a signal with a mask reference parameter.

**Description**

The block contains an S-function to which the message string parameter is passed to. If the input value does not match the reference value, the S-function sends a message. The mismatch output is set to one. When the input is switched back to the reference value, no message is sent, but the output is set back to zero. The reference value is set with the mask parameter and the message string can be adapted within the mask.

This block is mainly used to ensure the matching between the current parameterized value (as the input variable) and value of the variable in the compiled code (mask variable), e.g., within the ENGINE\_SETUP block of all the combustion engine models.

**Message string format** The input value and the reference value are passed to the provided string format as doubles. They can be accessed by a format specifier, e.g., %g.

A simple message string could be: 'Error: The input value is %g whereas the reference value is %g.'

If you use multiple instances of this block, you are recommended to introduce a relationship to the block, for example by adding the current block with the gcb command.

```
sprintf('Error: In Block %. The input value is %g whereas the reference value is %g.',asmtool('str2printf',gcb))
```

#### Note

The sprintf command is evaluated in MATLAB during initialization. This adds the block path and converts the double %% to one.%. The string presented to the S-function is then: 'Error: In Block <....>. The input value is %g whereas the reference value is %g.'

The call of `asmtool('str2printf',...)`) ensures that all '%' and '\' are prepared (duplicated) so they do not influence the print function inside the S-function.

#### Imports

The following table shows the imports:

Name	Unit	Description
value	[]	Input value that is compared to the reference parameter

#### Outports

The following table shows the outports:

Name	Unit	Description
mismatch	[0 1]	Flag to indicate mismatch: <ul style="list-style-type: none"> <li>▪ 0: The input matches the reference value.</li> <li>▪ 1: The input does not match the reference value.</li> </ul>

#### Parameters

The following table shows the parameters:

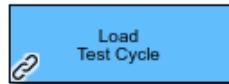
Name	Unit	Description
Library for licensecheck	[]	To specify the ASM library used for the license check of the underlying S-function. The Inherited from Block option is available only if the block is used in an ASM block.
Message String	[]	Message string (For instructions on how to create a message string, refer to <a href="#">Message string format</a> on page 183)
Reference Value	[]	Reference Value
Severity of Message	[]	Lets you select the severity of the message. Possible values: <ul style="list-style-type: none"> <li>▪ Info</li> </ul>

Name	Unit	Description
		<ul style="list-style-type: none"> <li>▪ Warning</li> <li>▪ Error</li> </ul> <p>On dSPACE platforms, this can be passed on to the severity of the log message.</p>

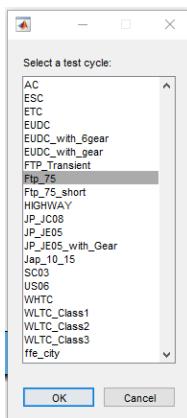
## Test Cycle

### Description

The Load Test Cycle block selects the test cycle for the current simulation.



You can select from the test cycles stored in `\Simulation\IniFiles\DrivingCycles` using the following Simulink dialog.



### Related topics

### HowTos

- [How to Select a Test Cycle \(ASM Diesel Engine InCylinder Model Description\)](#)
- [How to Select a Test Cycle \(ASM Diesel Engine Model Description\)](#)
- [How to Select a Test Cycle \(ASM Gasoline Engine Model Description\)](#)
- [How to Select a Test Cycle \(ASM Gasoline Engine InCylinder Model Description\)](#)

## Engine OP Selector

### Introduction

The EngOP\_Selector block selects a signal from the engine operating point vector.



### Imports

The following table shows the imports:

Name	Unit	Description
EngOP	[]	Engine operating point vector

### Outports

The following table shows the outports:

Name	Unit	Description
EngOP1	[]	Engine operating point 1
EngOP2	[]	Engine operating point 2

### Parameters

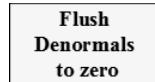
The following table shows the parameters:

Name	Unit	Description
Const_Idx_EngOP	[]	Selected signals for EngOP
Const_num_Sign_EngOP	[]	Number of signals in EngOP vector

## Flush Denormals to Zero

### Introduction

You can add the Flush Denormals to zero block to the task of the Simulink model. This lets you activate flushing to zero during the compile process on PHS-bus-based systems (e.g., the DS1006).



For more information, refer to [Troubleshooting with Rising Turnaround Times at Steady Operating Points](#) on page 276.

## Select Maneuver

### Introduction

The Select Maneuver block selects the maneuver type and maneuver control when the simulation starts via a drop-down menu.



The block changes the respective control parameters in the MATLAB workspace. Therefore, during the simulation, any change can only take action after updating the model, i.e. with **Ctrl + D**.

The following table describes the maneuver types:

Maneuver type	Description
Manual	To simulate the model manually in ControlDesk on a dSPACE platform or in Simulink on a standard PC.
Stimulus	To define a time-dependent stimulus maneuver to test the behavior of the model. The maneuver can have the following states: <ul style="list-style-type: none"> <li>▪ Stop: Maneuver is stopped.</li> <li>▪ Start: Maneuver is started.</li> </ul>
Test cycle	Contains the longitudinal driving and engine cycles, which can be defined by a MATLAB function that is part of your installation. The maneuver can have the following states: <ul style="list-style-type: none"> <li>▪ Stop: Maneuver is stopped.</li> <li>▪ Start: Maneuver is started.</li> </ul>

### Parameters

The following table shows the parameters:

Name	Unit	Description
SW_Maneuver	[]	Maneuver type
SW_ManeuverStart	[]	Maneuver default control

### Related topics

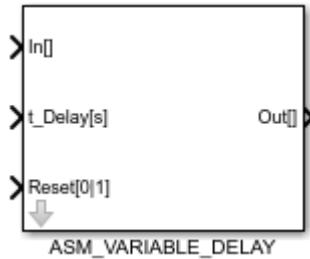
#### HowTos

- [How to Select Maneuver Conditions \(ASM Diesel Engine Model Description\)](#)
- [How to Select Maneuver Conditions \(ASM Drivetrain Basic Model Description\)](#)
- [How to Select Maneuver Conditions \(ASM Diesel Engine InCylinder Model Description\)](#)
- [How to Select Maneuver Conditions \(ASM Gasoline Engine InCylinder Model Description\)](#)
- [How to Select Maneuver Conditions \(ASM Gasoline Engine Model Description\)](#)

## Variable Delay

### Introduction

The block contains an S-function that delays the input signal by a specified delay time.



The function reserves a ring buffer at the start of the simulation based on the discrete step size and the maximum possible delay time parameter. The output is initially set to zero and the current input is saved until the delay time elapsed.

The output is updated only once in each step and no interpolation or extrapolation is performed for the values that lie between successive steps. For a continuously changing delay time, the result might be inaccurate, especially for fast changes within one step size.

### Imports

The following table shows the imports:

Name	Unit	Description
In	[]	Input to be delayed
t_Delay	[s]	Delay time
Reset	[0 1]	Reset

### Outports

The following table shows the outports:

Name	Unit	Description
Out	[]	Delayed input

### Parameters

The following table shows the parameters:

Name	Unit	Description
Maximum delay time	[s]	Maximum possible delay time
Library for licensecheck	[]	To specify the ASM library used for the license check of the underlying S-function. The <b>Inherited from Block</b> option is available only if the block is used in an ASM block.

# Extended Look-Up Table

## Where to go from here

## Information in this section

[1-D Look-Up Table](#)..... 188

The `asm_1d_lookup` block interpolates the one-dimensional look-up table  $v=v(x)$ .

[2-D Look-Up Table](#)..... 189

The `asm_2d_lookup` block interpolates a two-dimensional look-up table  $v=v(x,y)$ .

[Shift Look-Up Table](#)..... 190

The `asm_shiftable_lookup` block uses a look-up-table to calculate the shift speed.

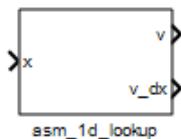
[1-D Reciprocal Integral Lookup](#)..... 191

This block interpolates a one-dimensional look-up table with constant extrapolation, where the v-data must not cross zero.

## 1-D Look-Up Table

### Introduction

The `asm_1d_lookup` block interpolates the one-dimensional look-up table  $v=v(x)$ . It also calculates the derivative of the function  $v$  with respect to  $x$ , i.e.,  $dv/dx$ .



### Imports

The following table shows the imports:

Name	Unit	Description
<code>x</code>	[]	Input value

### Outports

The following table shows the outports:

Name	Unit	Description
<code>v</code>	[]	Output value as result of the look-up table
<code>v_dx</code>	[]	Derivative of current working point

**Parameters**

The following table shows the parameters:

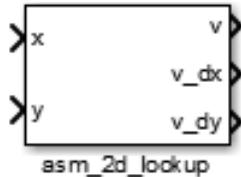
Name	Unit	Description
extr_type	[]	Extrapolation type
Library for licensecheck	[]	To specify the ASM library used for the license check of the underlying S-function. The Inherited from Block option is available only if the block is used in an ASM block.
v	[]	Vector of table data
x	[]	Vector of breakpoints

## 2-D Look-Up Table

**Introduction**

The `asm_2d_lookup` block interpolates a two-dimensional look-up table  $v=v(x,y)$ . It also calculates the derivative of the function  $v$  with respect to  $x$  and  $y$ , i.e.,  $dv/dx$  and  $dv/dy$ .

The following illustration shows the Simulink representation of the block.

**Imports**

The following table shows the imports:

Name	Unit	Description
x	[]	Input value 1
y	[]	Input value 2

**Outports**

The following table shows the outports:

Name	Unit	Description
v	[]	Output value as a result of table look-up
v_dx	[]	Derivative of current working point in x-direction
v_dy	[]	Derivative of current working point in y-direction

**Parameters**

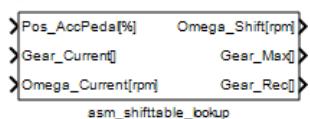
The following table shows the parameters:

Name	Unit	Description
extr_type	[]	Extrapolation type
Library for licensecheck	[]	To specify the ASM library used for the license check of the underlying S-function. The Inherited from Block option is available only if the block is used in an ASM block.
v	[]	Matrix of the table data
x	[]	Row index break point
y	[]	Column index of break points

## Shift Look-Up Table

**Introduction**

The `asm_shiftable_lookup` block uses a look-up-table to calculate the shift speed.



The transmission output speed at which a gear change is required is determined from the current gear and the accelerator pedal position. Additionally, the maximum gear is calculated from the given gear range. If the difference between two gear values drops to less than one, the lower gear is assumed to be maximum gear. A recommended gear (for a zero accelerator pedal position) is calculated from the table data and the current transmission output speed.

**Imports**

The following table shows the imports:

Name	Unit	Description
Name Pos_AccPedal	[%]	Acceleration pedal position
Gear_Current	[]	Current gear selection
Omega_Current	[rpm]	Current transmission output speed

**Outports**

The following table shows the outports:

Name	Unit	Description
Omega_Shift	[rpm]	Transmission output speed for next gear change
Gear_Max	[]	Maximum gear
Gear_Rec	[]	Recommended gear

**Parameters**

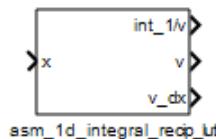
The following table shows the parameters:

Name	Unit	Description
extr_type	[]	Extrapolation type
Library for licensecheck	[]	To specify the ASM library used for the license check of the underlying S-function. The Inherited from Block option is available only if the block is used in an ASM block.
sw_recgear	[]	Calculate recommended gear
v	[rpm]	Downshift speed values
x	[%]	Accelerator pedal input values
y	[]	Gear input values

## 1-D Reciprocal Integral Lookup

**Introduction**

The `asm_1d_integral_recip_lut` block interpolates a one-dimensional look-up table with constant extrapolation, where the v-data must not cross zero.



The block also provides the integral of the reciprocal function value (e.g., to be used in steering angle calculations).

**Imports**

The following table shows the imports:

Name	Unit	Description
x	[]	Input signal

**Outports**

The following table shows the outports:

Name	Unit	Description
Int_1/v	[]	Description integral output of the reciprocal table value
v	[]	Output value as a result of the table look-up
v_dx	[]	Derivative of the current working point.

**Parameters**

The following table shows the parameters:

Name	Unit	Description
Library for licensecheck	[]	To specify the ASM library used for the license check of the underlying S-function. The Inherited from Block option is available only if the block is used in an ASM block.
v	[]	Vector of table data
x	[]	Vector of break points

## Replace Scopes

### Scope Handling User Interface

**Introduction**

The ScopeHandlingGui block lets you disable, restore, or show all scopes of the active Simulink system via a user interface.

**Related topics****HowTos**

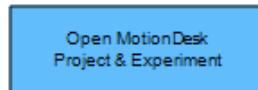
- [How to Handle Simulink Plotters \(ASM Trailer Model Description\)](#)
- [How to Handle Simulink Plotters \(ASM Vehicle Dynamics Model Description\)](#)
- [How to Handle Simulink Plotters \(ASM Drivetrain Basic Model Description\)](#)
- [How to Handle Simulink Plotters \(ASM Gasoline Engine InCylinder Model Description\)](#)
- [How to Handle Simulink Plotters \(ASM Gasoline Engine Model Description\)](#)
- [How to Handle Simulink Plotters \(ASM Diesel Engine Model Description\)](#)
- [How to Handle Simulink Plotters \(ASM Diesel Engine InCylinder Model Description\)](#)

# Open Experiment

## Open Experiment

### Introduction

The OpenExperiment block can be used to open a defined project file (typically, a MotionDesk or a ModelDesk experiment file).



### Parameters

The following table shows the parameters:

Name	Unit	Description
dSProduct	[]	dSPACE experiment tool
ExpDir	[]	Experiment directory

# Operator

### Where to go from here

### Information in this section

[Activate Developer Version](#)..... 194

The ActivateDeveloperVersion block lets you activate the developer version.

[Activate Operator Version](#)..... 194

The ActivateOperatorVersion block lets you activate the operator version.

## Activate Developer Version

### Introduction

The ActivateDeveloperVersion block lets you activate the developer version.



When you double-click the ActivateDeveloperVersion block, the model is parsed for ASM blocks. For each identified ASM library used in this model, a corresponding developer version is checked for availability. If the developer version is available, the block links are changed to the developer version library. An overview of corresponding link operations is written to the MATLAB Command Window.

### Parameters

The following table shows the parameters:

Name	Unit	Description
plotblock_updt	[0 1]	Switch for updating the ModelDesk plotting block after activation: <ul style="list-style-type: none"><li>▪ 0: No</li><li>▪ 1: Yes</li></ul>

### Related topics

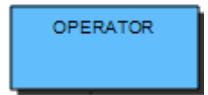
### References

<a href="#">Activate Operator Version</a> .....	194
---	-----

## Activate Operator Version

### Introduction

The ActivateOperatorVersion block lets you activate the operator version. For more information on the operator version, refer to [Operator Version](#) on page 61.



When you double-click the ActivateOperatorVersion block, the model is parsed for ASM blocks. For each identified ASM library used in this model, a corresponding operator version is checked for availability. If the operator version is available, the block links are changed to the operator version library. An overview of corresponding link operations is written to the MATLAB Command Window.

**Parameters**

The following table shows the parameters:

Name	Unit	Description
plotblock_updt	[0 1]	Switch for updating the ModelDesk plotting block after activation: ■ 0: No ■ 1: Yes

**Related topics****References**

Activate Developer Version.....	194
---------------------------------	-----

## Block Replacement

**Where to go from here****Information in this section**

Replace Tire TMEasy.....	195
--------------------------	-----

The ReplaceTireTMEasy block lets you replace the dummy TMeasy tire model with the corresponding TMeasy full version or vice versa.

Replace Tire MagicFormula.....	196
--------------------------------	-----

The ReplaceTireMF block lets you replace the dummy Magic Formula tire model with the corresponding Magic Formula full version or vice versa.

## Replace Tire TMEasy

**Introduction**

The ReplaceTireTMEasy block lets you replace the dummy TMeasy tire model with the corresponding TMeasy full version or vice versa.



When you double-click the ReplaceTireTMEasy block, a user interface opens that lets you replace the model.

## Replace Tire MagicFormula

---

### Introduction

The ReplaceTireMF block lets you replace the dummy Magic Formula tire model with the corresponding Magic Formula full version or vice versa.



When you double-click the ReplaceTireMF block, a user interface opens that lets you replace the model.

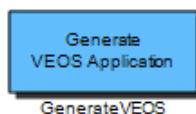
## Code Generation

### Generate VEOS

---

#### Introduction

The GenerateVEOS block lets you generate an SIC and OSA file that is used for VEOS simulations.



When you double-click the GenerateVEOS block, an SIC file is generated by using the DSRT compiler. The SIC file is then used to generate an OSA file via the VEOS tool automation. This file can be used for VEOS simulations.

**Related topics****HowTos**

- [How to Generate an OSA File for VEOS \(ASM Electric Components Model Description\)](#)
- [How to Generate an OSA File for VEOS \(ASM Trailer Model Description\)](#)
- [How to Generate an OSA File for VEOS \(ASM Diesel Engine InCylinder Model Description\)](#)
- [How to Generate an OSA File for VEOS \(ASM Diesel Engine Model Description\)](#)
- [How to Generate an OSA File for VEOS \(ASM Gasoline Engine Model Description\)](#)
- [How to Generate an OSA File for VEOS \(ASM Gasoline Engine InCylinder Model Description\)](#)
- [How to Generate an OSA File for VEOS \(ASM Drivetrain Basic Model Description\)](#)
- [How to Generate an OSA File for VEOS \(ASM Vehicle Dynamics Model Description\)](#)

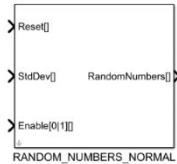
## Random Numbers

### Random Numbers with Normal Distribution

**Description**

The RANDOM\_NUMBERS\_NORMAL block calculates reproducible random numbers with normal distribution.

The following illustration shows the Simulink representation of the block:

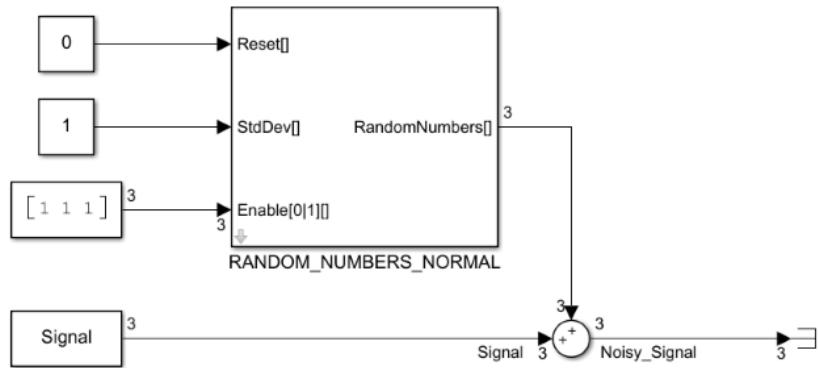


The Seed parameter defines the sequence of random numbers.

The Reset input resets the sequence of random numbers to its beginning. The random numbers are therefore reproducible for a given seed. The normal distribution is used to calculate the random numbers. The normal distribution is defined by the standard deviation in the StdDev input and has a mean of zero.

The Enable input defines which elements of the RandomNumbers output contain random numbers.

This block can be used to add normal noise to a signal. The noise is calculated by adding the random numbers to a signal. The following illustration shows an example:

**Imports**

The following table shows the imports:

Name	Unit	Description
Enable	[0 1]	Enable for adaptive creation of random numbers: <ul style="list-style-type: none"> <li>▪ 0: Off</li> <li>▪ 1: On</li> </ul>
Reset	[]	Resets the random number calculation.
StdDev	[]	Standard deviation of the random numbers.

**Outports**

The following table shows the outports:

Name	Unit	Description
RandomNumbers	[]	Random numbers in the dimension of the Enable inport

**Parameters**

The following table shows the parameters:

Name	Unit	Description
Seed	[]	Starting seed (integer) for a random number calculation. Random numbers are reproducible for a given seed.

# Useful Functions

## Introduction

The topics below describe several functions useful for working with the ASM. Depending on the installed ASM licenses, it can happen that not all of these functions are available.

## Where to go from here

## Information in this section

<a href="#">asm_eng_drivingcycles</a> .....	200
To load the driving cycles to an ASM engine model.	
<a href="#">asm_multinstance</a> .....	201
To set the multi-instance parameters of an ASM block or to get an overview of all multi-instance blocks in the model.	
<a href="#">asm_plot_table</a> .....	203
To plot parameterization data.	
<a href="#">asm_plot_geosusp</a> .....	204
To visualize geometrically described suspension.	
<a href="#">asm_tablegenerator</a> .....	205
To generate maps from data vectors.	
<a href="#">asm_traffic_mdl_update_num_fellows</a> .....	219
To change the maximum number of fellows supported in a traffic model.	
<a href="#">CreateIndependentParameterxml.py</a> .....	220
To create renamed parameter XML files, which are directly linked to the corresponding parameter page.	
<a href="#">CreateGeoOptTrajectory.py</a> .....	223
To automatically define trajectories for vehicles in a road model.	

## asm\_eng\_drivingcycles

### asm\_eng\_drivingcycles

---

<b>Purpose</b>	To load the engine and chassis dynamomotor test cycles to the ASM engine model.
----------------	---

---

<b>Basic concept</b>	<p>The <code>asm_eng_drivingcycles</code> function enables you to load the engine and chassis dynamomotor test cycles to the ASM engine model. These test cycles are defined as M files in the <code>DrivingCycles</code> folder in the project directory. The function can be used in three different ways:</p> <ul style="list-style-type: none"><li>▪ By including it as a function call in an initialization script as follows: <code>asm_eng_drivingcycles('drivingcycle','DRIVINGCYCLE_NAME')</code>, where <code>DRIVINGCYCLE_NAME</code> is the test cycle name.</li><li>▪ By typing <code>asm_eng_drivingcycles</code>. This opens a file dialog that displays a list of the included test cycles which are contained in the <code>DrivingCycles</code> folder of the current project. The chosen test cycle will then be initialized. In this context, you can define a new test cycle and initialize it as mentioned above. To define a new test cycle, the included M files in the <code>DrivingCycles</code> folder can be used as templates. Depending on whether the new test cycle is an engine or chassis dynamomotor cycle, you can select one of the mentioned M files as template. The selected M file should then be copied to the same folder and renamed. The content of the new M file can then be edited. At the next call of <code>asm_eng_drivingcycles</code>, the new test cycle will appear in the file dialog and can be chosen as well as initialized.</li><li>▪ By typing <code>asm_eng_drivingcycles('plot')</code>. This plots the currently loaded test cycle.</li></ul>
----------------------	--

# asm\_multiinstance

## asm\_multiinstance

### Purpose

To set the multi-instance parameters of an ASM block or to get an overview of all multi-instance blocks in the model.

### Setting multi-instance parameters

To set the multi-instance parameters of an ASM block, use the following command:

```
asm_multiinstance('BlockUser interface','CurrentBlock',gcb)
```

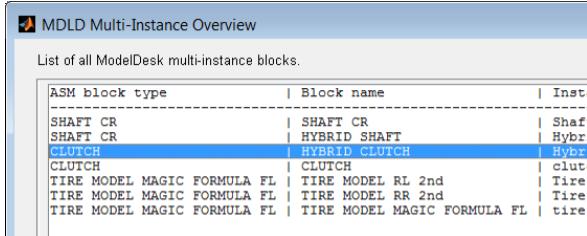
A dialog is opened to set the parameters of the specified block.

### Getting an overview of all multi-instance blocks

To get an overview of all multi-instance blocks in the model, use the following command:

```
asm_multiinstance('Overview','ModelName',bdroot)
```

A dialog opens, see the following illustration.



The screenshot shows a Windows-style dialog titled "MDLD Multi-Instance Overview". The title bar has a small icon on the left and the title text. Below the title bar is a light gray header bar with the text "List of all ModelDesk multi-instance blocks.". The main area is a table with a light gray background and a thin black border. The table has three columns with headers: "ASM block type", "Block name", and "Instance name". There are seven rows of data, each representing a different block type and its instances. The last row is partially cut off at the bottom. The "CLUTCH" row is highlighted with a blue selection bar, indicating it is currently selected.

ASM block type	Block name	Instance name
SHAFT CR	SHAFT CR	Shaft
SHAFT CR	HYBRID SHAFT	Hybrid
CLUTCH	HYBRID CLUTCH	Hybre
CLUTCH	CLUTCH	clutc
TIRE MODEL MAGIC FORMULA FL	TIRE MODEL RL 2nd	Tire
TIRE MODEL MAGIC FORMULA FL	TIRE MODEL RR 2nd	Tire
TIRE MODEL MAGIC FORMULA FL	TIRE MODEL MAGIC FORMULA FL	tire

The dialog displays the following parameters:

- ASM block type: ASM type of the block
- Block name: Simulink block name
- Instance name: Instance name of the block
- Instance ID: Instance ID of the block
- Block path: Model path of the block

You can modify the multi-instance parameters of a selected block in the list with the **Modify** button.

### Note

You can also open the multi-instance overview via the **Multi-Instance Overview** button in your model. Refer to [Multi-Instance Overview](#) on page 165.

**Related topics**

**Basics**

Multi-Instance..... 53

**References**

Multi-Instance Overview..... 165

## asm\_plot\_table

### asm\_plot\_table

---

Purpose	To plot your parameterization data.
---------	-------------------------------------

---

Basic concept	This function allows you to plot your parameterization data which has been set up according to the variable definitions in <a href="#">Variable Structure</a> on page 47. Then you can call <code>asm_plot_table</code> with a structure as an input argument, for example,
---------------	---

```
asm_plot_table(MDL.Engine.PistonEngine.Map_Lambda_a)
```

During the parameterization, there are routines within ModelDesk extrapolating tables to a predefined size to ensure fixed sizes of maps. This is required for consistent variable operations on the real time platform. To plot only the axis range, which you have chosen and which is not extrapolated, you can use the following command:

```
asm_plot_table(MDL.Engine.PistonEngine.Map_Lambda_a, 1)
```

## asm\_plot\_geosusp

### asm\_plot\_geosusp

---

<b>Purpose</b>	To visualize geometrically described suspension.
----------------	--

<b>Basic concept</b>	This function allows you to visualize geometrically described suspension. The function plots each linkage point and shows suspension at configuration state. You can simply call <code>asm_plot_geosusp</code> or use the following input arguments: <code>'all'</code> , <code>'McPhersonStrut'</code> , <code>'SemiTrailingArm'</code> , <code>'RigidAxe'</code> .
----------------------	--

For further information on using

```
asm_plot_geosusp
```

type the following in the MATLAB Command Window:

```
help asm_plot_geosusp
```

# asm\_tablegenerator

<b>Introduction</b>	To generate maps. You can use two different methods with the <b>asm_tablegenerator</b> function.
---------------------	--

<b>Where to go from here</b>	<b>Information in this section</b>								
	<table> <tr> <td><b>2D_Table Method</b></td> <td>205</td> </tr> <tr> <td>The <b>2D_Table</b> method of <b>asm_tablegenerator</b> is a function to generate 2-D tables from data vectors.</td> <td></td> </tr> <tr> <td><b>1D_PsiTable Method</b></td> <td>215</td> </tr> <tr> <td>The <b>1D_PsiTable</b> method of <b>asm_tablegenerator</b> is a function to generate the psi table.</td> <td></td> </tr> </table>	<b>2D_Table Method</b>	205	The <b>2D_Table</b> method of <b>asm_tablegenerator</b> is a function to generate 2-D tables from data vectors.		<b>1D_PsiTable Method</b>	215	The <b>1D_PsiTable</b> method of <b>asm_tablegenerator</b> is a function to generate the psi table.	
<b>2D_Table Method</b>	205								
The <b>2D_Table</b> method of <b>asm_tablegenerator</b> is a function to generate 2-D tables from data vectors.									
<b>1D_PsiTable Method</b>	215								
The <b>1D_PsiTable</b> method of <b>asm_tablegenerator</b> is a function to generate the psi table.									

## 2D\_Table Method

<b>Where to go from here</b>	<b>Information in this section</b>																				
	<table> <tr> <td><b>Basics of 2D_Table Method</b></td> <td>206</td> </tr> <tr> <td>The <b>2D_Table</b> method is a function to generate 2-D tables from data vectors.</td> <td></td> </tr> <tr> <td><b>Function Call to the 2D_Table Method</b></td> <td>206</td> </tr> <tr> <td>To call the <b>2D_Table</b> method.</td> <td></td> </tr> <tr> <td><b>Execution Order of the 2D_Table Method</b></td> <td>209</td> </tr> <tr> <td>Some parameters influence the execution of the <b>2D_Table</b> method when creating the axes, interpolating data, and extrapolating data.</td> <td></td> </tr> <tr> <td><b>Optional Parameters of the 2D_Table Method for Extrapolation</b></td> <td>211</td> </tr> <tr> <td>You can use optional parameters in the <b>2D_Table</b> method to extrapolate the 2-D table.</td> <td></td> </tr> <tr> <td><b>Optional Parameters of the 2D_Table Method for Plotting</b></td> <td>214</td> </tr> <tr> <td>You can use optional parameters in the <b>2D_Table</b> method to set options for the plot.</td> <td></td> </tr> </table>	<b>Basics of 2D_Table Method</b>	206	The <b>2D_Table</b> method is a function to generate 2-D tables from data vectors.		<b>Function Call to the 2D_Table Method</b>	206	To call the <b>2D_Table</b> method.		<b>Execution Order of the 2D_Table Method</b>	209	Some parameters influence the execution of the <b>2D_Table</b> method when creating the axes, interpolating data, and extrapolating data.		<b>Optional Parameters of the 2D_Table Method for Extrapolation</b>	211	You can use optional parameters in the <b>2D_Table</b> method to extrapolate the 2-D table.		<b>Optional Parameters of the 2D_Table Method for Plotting</b>	214	You can use optional parameters in the <b>2D_Table</b> method to set options for the plot.	
<b>Basics of 2D_Table Method</b>	206																				
The <b>2D_Table</b> method is a function to generate 2-D tables from data vectors.																					
<b>Function Call to the 2D_Table Method</b>	206																				
To call the <b>2D_Table</b> method.																					
<b>Execution Order of the 2D_Table Method</b>	209																				
Some parameters influence the execution of the <b>2D_Table</b> method when creating the axes, interpolating data, and extrapolating data.																					
<b>Optional Parameters of the 2D_Table Method for Extrapolation</b>	211																				
You can use optional parameters in the <b>2D_Table</b> method to extrapolate the 2-D table.																					
<b>Optional Parameters of the 2D_Table Method for Plotting</b>	214																				
You can use optional parameters in the <b>2D_Table</b> method to set options for the plot.																					

## Basics of 2D\_Table Method

---

<b>Purpose</b>	To generate 2-D tables from data vectors.
<b>Basic concepts</b>	<p>The <b>2D_Table</b> method of <code>asm_tablegenerator</code> is a function to generate 2-D tables from data vectors (for example, measurement data). The interpolation is done by <code>griddata</code>. This results in a table whose contents are within the range of the provided data. The extrapolation can be defined separately for each axis direction (x up and down, y up and down). You can also</p> <ul style="list-style-type: none"><li>▪ Add constraints, for example, limits with upper and lower values</li><li>▪ Set fixed values to specified points</li><li>▪ Add further points to the given data</li><li>▪ Skip some points of the given data</li></ul> <p>For a better understanding of what happens, the table can be plotted after every extrapolation step.</p>

## Function Call to the 2D\_Table Method

---

<b>Purpose</b>	To call the 2D_Table method.
<b>Function call with measurement data</b>	<p>The 2D_Table method works with parameter value pairs.</p> <pre>OutMdlStruct = asm_tablegenerator('2D_Table',...     'table', OutMdlStruct, ...     'meas_data',{x_Data, y_Data, v_Data}, ...     'x_axis',x_axis, ...     'y_axis',y_axis, ...     &lt;optional parameter&gt;);</pre> <p><b>table</b> The <b>table</b> parameter is optional. It is a structure similar to the generated output structure. The existing fields are transferred to the output. If it already contains data fields (x, y, v) and no <code>meas_data</code> is specified, they are used.</p>

If you are working in ModelDesk Processing, it is easy to create the correct structure with:

```
OutMdlStruct      =...
asm_proc('get_parameter',...
'MainComp',MainComp,...
'SubComp',SubComp,...
'ParaName',ParaName,...
'InstID',InstID,...
'FieldName','Infofields');
```

**meas\_data** The **meas\_data** parameter is optional. It is a 1x3 cell of measurement data for x, y, and v. Each element is either a column vector or a structure with the fields **vName**, **vUnit**, and **v**. The data length has to be the same for all axes.

The **table** and **meas\_data** parameters are optional. It is required that either the specified table contains data fields or **meas\_data** is specified.

**x\_axis** The **x\_axis** parameter is optional. It is either a column vector or a structure with the fields **vName**, **vUnit**, and **v**.

The **table** and **x\_axis** parameters are optional. It is required that either the specified table contains the field x or **x\_axis** is specified.

**y\_axis** The **y\_axis** parameter is optional. It is either a column vector or a structure with the fields **vName**, **vUnit**, and **v**.

The **table** and **y\_axis** parameters are optional. It is required that either the specified table contains the y field or **y\_axis** is specified.

### Function call table inversion

The **2D\_Table** method can be used to invert tables with parameter value pairs.

```
OutMdlStruct = asm_tablegenerator('2D_Table',...


```

**Table2invert** The **table2invert** parameter is required. It is a structure similar to the generated output structure. The existing fields are transferred to the output under consideration of the new axis order.

**new\_axis\_order** The **new\_axis\_order** parameter is required. It is a 1 × 3 cell with the strings x, y, and v.

Example: {'x', 'v', 'y'} means the x-axis is the same as before, v is used as y, and the old y is used as v.

**x\_axis** The **x\_axis** parameter is optional. It is either a column vector or a structure with the fields **vName**, **vUnit**, and **v**. If **x\_axis** is not specified, the related axis of the **table2invert** parameter is used. If the new x variable is the old v variable, a default axis with 100 breakpoints between the minimum and maximum data is created.

**y\_axis** The **y\_axis** parameter is optional. It is either a column vector or a structure with the fields **vName**, **vUnit**, and **v**. If **y\_axis** is not specified, the

related axis of the `table2invert` parameter is used. If the new `y` variable is the old `v` variable, a default axis with 100 breakpoints between the minimum and maximum data is created.

#### Backwards-compatible function call

The following calls to the `2D_Table` method are obsolete. They are still working, but it is recommended to use the parameter value pairs described above.

```
table = asm_tablegenerator('2D_Table',...
    X,Y,V,XI,YI,<optional parameter>)
```

where `X`, `Y`, `V` represent the data column vectors, and `XI`, `YI` the x- and y-axis values as column vectors or as structures.

If the data is already a structure with the `x`, `y` and `v` fields and `[size(x, 1), size(y, 1)] = size(v)`, it can be directly provided to the function.

```
table = asm_tablegenerator('2D_Table',...
    input_struct,<optional parameter>)
```

If the data is already a structure with the `x`, `y` and `v` fields and `[size(x, 1), size(y, 1)] = size(v)`, but new axes are desired:

```
table = asm_tablegenerator('2D_Table',...
    input_struct,XI,YI,<optional parameter>)
```

where `XI`, `YI` represent the x- and y-axis values as column vectors or as structures.

If the axes are provided as a structure, the `v` field is required. The `vName` and `vUnit` fields are optional. All three fields are used for the related fields (`x`, `xName`, `xUnit`, `y`, `yName`, `yUnit`) of the axis.

#### Optional parameters

The optional parameters always have to be specified as parameter/value pairs, in which the value often is a cell array.

#### Output structure

In all cases, a table is returned with the following fields. If the input is a structure, all fields are kept, and only `x`, `y`, and `v` are modified. If `DimView` is not set, it is added.

Field	Description
<code>xName</code>	Name of x-axis, default is 'x'
<code>yName</code>	Name of y-axis, default is 'y'
<code>vName</code>	Name of v-axis, default is 'v'
<code>xUnit</code>	Unit of x-axis, default is '[]'
<code>yUnit</code>	Unit of y-axis, default is '[]'
<code>vUnit</code>	Unit of v-axis, default is '[]'
<code>DimView</code>	Size before extrapolation according DimV [DimX DimY]
<code>Comment</code>	Comment (if provided as an optional parameter)
<code>x</code>	x-axis vector, dimension [n,1]

Field	Description
y	y-axis vector dimension [m, 1]
v	v-table, dimension [n, m]

## Execution Order of the 2D\_Table Method

### Axes names and units

The output structure contains the `xName`, `xUnit`, `yName`, `yUnit`, `vName`, and `vUnit` fields. The content of these fields can come from different sources and are used with the following priorities:

The fields are transferred from the `table` or the `table2invert` parameter. If the fields do not exist, the axes fields are used. If the axis does not contain the fields, the measurement data is used. The optional parameters from the table below have the highest priority. If no specification is provided at all, the default values described in [Output structure](#) on page 208 are used.

#### Note

To avoid mistakes, the table generator checks if the provided variable names and units are consistent between table, axes and measurement data. It will issue a warning, for example, if `table.xUnit = 'bar'` is different to `x_axis.Unit = 'Pa'`. This warning can be disabled by

```
warning('off', 'asm:tablegenerator:inconsistentname')
```

This is similar for the `Comment` field. The `Comment` value of the input structure is used if the fields exist, but it is overwritten by the optional `Comment` parameter. If both are empty, an empty string is assigned.

#### Note

When using the `asm_tablegenerator` function in ModelDesk Processing, it is recommended to provide the `table` or `table2invert` parameter, read with

```
asm_proc('get_parameter', ..., 'FieldName', 'Infofields')
```

The optional `<...>Name` and `<...>Unit` parameter should not be used for ModelDesk because ModelDesk only reads the `x`, `y`, and `v` fields back from processing. Changes in names and units are not possible and ignored.

Parameter	Description
<code>xName</code>	The provided string is assigned to the <code>xName</code> field of the output structure.
<code>xUnit</code>	The provided string is assigned to the <code>xUnit</code> field of the output structure.

Parameter	Description
yName	The provided string is assigned to the <b>yName</b> field of the output structure.
yUnit	The provided string is assigned to the <b>yUnit</b> field of the output structure.
vName	The provided string is assigned to the <b>vName</b> field of the output structure.
vUnit	The provided string is assigned to the <b>vUnit</b> field of the output structure.
Comment	The provided string is assigned to the <b>Comment</b> field of the output structure.

**Creating axes**

First the axes are created. The following parameters influence this:

Parameters <sup>1)</sup>	Description
table	The axes x and y are used basic axes if <b>x_axis</b> and <b>y_axis</b> are not specified.
table2invert	The axes x and y are used basic for the related new axes if <b>x_axis</b> and <b>y_axis</b> are not specified.
x_axis	Basic for x- axis
y_axis	Basic for y- axis
'x_axispoints',{x}	Values are added to the x-axis
'y_axispoints',{y}	Values are added to the y-axis
'xvset',{[x,v]}	The x values are added to the x-axis
'yvset',{[y,v]}	The y values are added to the y-axis
'xyvset',{[x,y,v]}	The x/y values are added to the x-/y- axis (if not desired, use 'xyvdata',{[x,y,v]})
'DimV',{[x,y]}	After all other changes, the sizes of the axes are increased according to the values specified by <b>DimV</b> . If the provided structure has a <b>DimV</b> field, its value is used unless a separate <b>DimV</b> parameter is provided.

<sup>1)</sup> For details on the parameters, refer to [Optional Parameters of the 2D\\_Table Method for Extrapolation](#) on page 211.

The axes can be standardized. That means that they are scaled within a window. This standardization is only internal and does not affect the axes of the output structure. Standardization has a strong influence on the result of nearest, smooth\_linear and smooth\_constant extrapolation.

**Interpolating data**

When the axes have been set up, an initial interpolation is performed with `griddata`. The following parameter are used:

Parameter <sup>1)</sup>	Description
<code>'InterpolMethod',{method}</code>	The method is passed to <code>griddata</code> as an interpolation method. It can be 'linear' (default), 'cubic', or 'nearest'. The 'nearest' option is not recommended, because, <code>griddata</code> then also performs an extrapolation.
<code>'SkipList',{index}</code>	Data points with the specified index are ignored.

<sup>1)</sup> For details on the parameters, refer to [Optional Parameters of the 2D\\_Table Method for Extrapolation](#) on page 211.

**Related topics****Basics**

[get\\_parameter Method.....](#) 232

## Optional Parameters of the 2D\_Table Method for Extrapolation

**Introduction**

You can use optional parameters in the `2D_Table` method to extrapolate the 2-D table.

**Parameters for extrapolation**

The optional parameters always have to be specified as parameter/value pairs, in which the value often is a cell array. The following parameters can be used to extrapolate the 2-D table.

Parameter/Value Pair	Description
<code>'StandarizeAxes',{method}</code>	<p>The axes are scaled. Several methods are available:</p> <ul style="list-style-type: none"> <li>▪ <code>'StandarizeAxes',{'value'}</code> The scaling is done so that the highest/lowest values of the provided data fit 0.5 / -0.5.</li> <li>▪ <code>'StandarizeAxes',{'axes'}</code> (default) The scaling is done so that the highest/lowest axes values fit 0.5 / -0.5 (default).</li> <li>▪ <code>'StandarizeAxes',{'none'}</code> No scaling is used.</li> </ul>
<code>'SkipList',{[indexes]}</code>	The input points with the specified indexes are ignored for table generation. In the generated development plot they are shown in red.

Parameter/Value Pair	Description
'x_axispoints', {[points]}	The specified points are added to the x-axis.
'y_axispoints', {[points]}	The specified points are added to the y-axis.
'DimV', {[x,y]}	After all other changes, the sizes of the axes are increased according to the values specified by DimV. If the provided structure has a DimV field, this is also taken into account.
'InterpolMethod', {method}	The method is passed to <code>griddata</code> as an interpolation method. The method can be 'linear' (default), 'cubic' or 'nearest'. The nearest option is not recommended, because this option also performs extrapolation and no further extrapolation can be done.
'one_step_extrapolation', (n)	The method extrapolates linear to the next uncalculated grid point. You can use this method if the measurement data is very near but not exactly on the given grid data. It should be used if the edges of the look-up table look strange. n defines the number of iterations.
'xvset', {[x,v], <mode>}	For all points with x-axis value x, set the related v matrix value to v. Multiple points can be handled by setting {[x <sub>1</sub> v <sub>1</sub> ; x <sub>2</sub> v <sub>2</sub> ; ...]}. The x values are added to the x-axis before any extrapolation starts. There are three possible modes: <ul style="list-style-type: none"> <li>▪ 'xvset', {[x,v], 'replace'}</li> <li>If the value is already set in the table, it is overwritten (default).</li> <li>▪ 'xvset', {[x,v], 'keep'}</li> <li>If the value is already set in the table, it is not overwritten.</li> <li>▪ 'xvset', {[x,v], 'mean'}</li> <li>If the value is already set in the table, the mean value is used.</li> </ul>
'yvset', {[y,v], <mode>}	Set for all point with y-axis y, set the v matrix value to v. Multiple points can be handled by setting {[y <sub>1</sub> v <sub>1</sub> ; y <sub>2</sub> v <sub>2</sub> ; ...]}. The y values are added to the y-axis before any extrapolation starts. The mode is defined as for <code>xvset</code> (see above).
'xyvset', {[x,y,v], <mode>}	Set the v matrix value to v for specified xy pairs. Multiple points can be handled by setting {[x <sub>1</sub> y <sub>1</sub> v <sub>1</sub> ; x <sub>2</sub> y <sub>2</sub> v <sub>2</sub> ; ...]}. The x and y values are added to the related axes before any extrapolation starts. The mode is defined as for <code>xvset</code> (see above).
'xyvdata', {[x,y,v], <mode>}	This works in a similar way to <code>xyvset</code> , except that the x and y values are not added to the related axes before any extrapolation starts. The mode is defined as for <code>xvset</code> (see above).
'x_up', {method, options}	<p>An extrapolation to higher x values is done according to the specified table. Several methods are available. An extrapolation is only done if the next point with a lower x value is defined. In the <code>linear</code> and <code>smooth_linear</code> method, the next two lower points must be defined.</p> <ul style="list-style-type: none"> <li>▪ 'x_up', {'value', &lt;limit&gt;, &lt;setvalue&gt;}</li> <li>Extrapolation with the &lt;setvalue&gt; value up to an x value of &lt;limit&gt;. To extrapolate over the whole table, set &lt;limit&gt; = [].</li> <li>▪ 'x_up', {'constant', &lt;limit&gt;}</li> <li>Constant extrapolation up to an x value of &lt;limit&gt;. To extrapolate over the whole table, set &lt;limit&gt; = [] or omit it.</li> <li>▪ 'x_up', {'smooth_constant', &lt;limit&gt;, &lt;count&gt;, &lt;scale&gt;}</li> <li>Extrapolation up to an x value of &lt;limit&gt;. To extrapolate over the whole table set &lt;limit&gt; = []. The mean value of &lt;count&gt; next points is used for extrapolation. The default for &lt;count&gt; is 3. The axes can be wrapped. If &lt;scale&gt; is greater than 1, the x direction is mainly used: if it is less than 1, the</li> </ul>

Parameter/Value Pair	Description
'x_up',{method,options}	y direction is mainly used. The default for <scale> is 1. The distance for a point is defined as $(x - x_0)^2 + (y - y_0)^2 \cdot <\text{scale}>$ <ul style="list-style-type: none"> <li>▪ 'x_up',{'linear','limit'}</li> </ul> <p>Linear extrapolation up to an x value of &lt;limit&gt;. To extrapolate over the whole table set &lt;limit&gt; =[] or omit it.</p> <ul style="list-style-type: none"> <li>▪ 'x_up',{'smooth_linear','limit','count','scale'}</li> </ul> <p>Extrapolation up to an x value of &lt;limit&gt;. To extrapolate over the whole table, set &lt;limit&gt; =[]. The mean value of &lt;count&gt; next points is used for extrapolation. The default for &lt;count&gt; is 10. The axes can be wrapped. If &lt;scale&gt; is greater than 1, the x direction is more considered. If it is less than 1, the y direction is more considered. Default for &lt;scale&gt; is 1. The distance for a point is defined as <math>(x - x_0)^2 + (y - y_0)^2 \cdot &lt;\text{scale}&gt;</math>. If only points with identical x values are found, a constant extrapolation is done.</p>
'x_down',{method,options}	An extrapolation to lower x values is performed according to the specified table. Different methods are available. An extrapolation is only done if the next point with a higher x value is defined. In the <b>linear</b> and <b>smooth_linear</b> method, the next two higher points have to be defined. The available methods are the same as for <b>x_up</b> (see above).
'y_up',{method,options}	An extrapolation to higher y values is performed according to the specified table. Different methods are available. An extrapolation is only done if the next point with a lower y value is defined. In the <b>linear</b> and <b>smooth_linear</b> method, the next two lower points have to be defined. The available methods are the same as for <b>x_up</b> (see above).
'y_down',{method,options}	An extrapolation to lower y values is performed according to the specified table. Different methods are available. An extrapolation is only done if the next point with a higher y value is defined. In the <b>linear</b> and <b>smooth_linear</b> method, the next two higher points have to be defined. The available methods are the same as for <b>x_up</b> (see above).
'v_uplim',{value,method}	The table values are limited to the specified value. Several methods are available: <ul style="list-style-type: none"> <li>▪ 'v_uplim',{&lt;value&gt;,'warning'} <ul style="list-style-type: none"> <li>If the table contains points greater than &lt;value&gt;, a warning is issued.</li> </ul> </li> <li>▪ 'v_uplim',{&lt;value&gt;,'warning and correction'} <ul style="list-style-type: none"> <li>If the table contains points greater than &lt;value&gt;, a warning is issued and these values are set to &lt;value&gt;.</li> </ul> </li> <li>▪ 'v_uplim',{&lt;value&gt;,'correction'} <ul style="list-style-type: none"> <li>If the table contains points greater than &lt;value&gt;, these values are set to &lt;value&gt;. This is the default method.</li> </ul> </li> </ul>
'v_lowlim',{value,method}	The table values are limited to the specified value. The methods are equal to <b>v_uplim</b> but a check is made for points lower than <value>.

## Extrapolating data

The result of **asm\_tablegenerator** depends on the order of the following optional parameters:

- 'xvset'
- 'yvset'
- 'xyvset'

- 'xyvdata'
- 'x\_up'
- 'x\_down'
- 'y\_up'
- 'y\_down'
- 'v\_uplim'
- 'v\_lowlim'

The commands are executed one after the other in the user-defined order. The extrapolation is always based on the result of the previous step. It makes a difference whether the x-axis is extrapolated first and then the y-axis or the other way round.

It is recommended to place 'v\_uplim' and 'v\_lowlim' at the end. Otherwise, these limits might be destroyed by other extrapolations.

After all the extrapolation commands, the remaining undefined areas are determined by a nearest interpolation with grid data.

---

#### Related topics

##### Basics

[Function Call to the 2D\\_Table Method.....](#) 206

## Optional Parameters of the 2D\_Table Method for Plotting

---

#### Introduction

You can use optional parameters in the `2D_Table` method to set options for the plot.

---

#### Parameters for plotting

The optional parameters always have to be specified as parameter/value pairs, in which the value often is a cell array. The following parameters are used to set options for the plot.

Parameter/Value Pair	Description
'InfoFlag',value	This flag makes it is possible to visualize the development of the extrapolated table. <ul style="list-style-type: none"> <li>▪ 0: No plot is created (default).</li> <li>▪ 2: A 2-D plot of the table development is shown, including the original data points as green circles.</li> </ul>

Parameter/Value Pair	Description
'FigureOptions',{string}	<ul style="list-style-type: none"> <li>▪ 3: A 2-D plot of the table development is shown, including the original data points as green circles. Additionally, data points are marked with their index numbers. These indexes can be used for the SkipList.</li> </ul> <p>The string is applied to the figure with the command <code>set(gcf, string)</code>. This can be used to set the position and size of the figure. Example:  <code>'FigureOptions',{'Position',[860 130 560 430]}</code></p>
'AxesOptions',{string}	<p>The string is applied to the figure with the command <code>set(gca, string)</code>. This can be used to set the view of the axes. Example:</p> <pre>'AxesOptions',{'View', [58 40]}</pre>
'PauseTime',value	<p>Depending on the 'InfoFlag', a figure is generated and updated after every extrapolation step. The &lt;value&gt; value defines how long it is to pause after every step. The value can be either a positive numerical value or the string 'pause'. If it is 'pause', you must strike any key to continue. The default value is 0.5.</p>
'PlotConfig',struct	<p>The structure is used to control the plot creation in ModelDesk Processing. It overwrites the specified values for 'InfoFlag' and 'PauseTime'.</p>

**Related topics****Basics**

<a href="#">Function Call to the 2D_Table Method.....</a>	206
---	-----

## 1D\_PsiTable Method

**Where to go from here****Information in this section**

<a href="#">Basics of 1D_PsiTable Method.....</a>	216
---	-----

The `1D_PsiTable` method is a function to generate the psi table.

<a href="#">Function Call of 1D_PsiTable Method.....</a>	216
--	-----

You can use the `1D_PsiTable` method to generate a psi table.

## Basics of 1D\_PsiTable Method

---

<b>Purpose</b>	To generate the psi table.
<b>Basic concept</b>	<p>The <b>1D_PsiTable</b> method is a function to generate the psi table. This table is needed to calculate the mass flow through an orifice such as a throttle or valve in engine models.</p> <p>This method provides a central definition which can be used in all models.</p>

## Function Call of 1D\_PsiTable Method

---

<b>Introduction</b>	You can use the <b>1D_PsiTable</b> method to generate a psi table.									
<b>Function call</b>	<p>The <b>1D_PsiTable</b> method expects the pressure ratio and kappa as input parameters. There are also some optional parameters.</p> <p>The output is a structure.</p> <pre>table = asm_tablegenerator('1D_PsiTable',...     p_ratio,kappa,&lt;optional parameter&gt;)</pre>									
<b>Required parameter</b>	<p>The following table describes the required parameters:</p> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Unit</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>p_Ratio</td> <td>[]</td> <td>Pressure ratio vector</td> </tr> <tr> <td>kappa</td> <td>[]</td> <td>Isentropic exponent of the gas</td> </tr> </tbody> </table>	Parameter	Unit	Description	p_Ratio	[]	Pressure ratio vector	kappa	[]	Isentropic exponent of the gas
Parameter	Unit	Description								
p_Ratio	[]	Pressure ratio vector								
kappa	[]	Isentropic exponent of the gas								
<b>Optional parameter</b>	<p>Optional parameters must be provided as parameter/value pairs. The following parameters can be used in the <b>1D_PsiTable</b> method.</p> <table border="1"> <thead> <tr> <th>Parameter/Value Pair</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>'p_Ratio_linear',&lt;value&gt;</td> <td>The slope of the psi function becomes very large for pressure ratios around 1. This can lead to oscillating mass flows during simulation. If this parameter is set, the psi function will be linear between p_Ratio_linear and 1/p_Ratio_linear. This reduces the slope</td> </tr> </tbody> </table>	Parameter/Value Pair	Description	'p_Ratio_linear',<value>	The slope of the psi function becomes very large for pressure ratios around 1. This can lead to oscillating mass flows during simulation. If this parameter is set, the psi function will be linear between p_Ratio_linear and 1/p_Ratio_linear. This reduces the slope					
Parameter/Value Pair	Description									
'p_Ratio_linear',<value>	The slope of the psi function becomes very large for pressure ratios around 1. This can lead to oscillating mass flows during simulation. If this parameter is set, the psi function will be linear between p_Ratio_linear and 1/p_Ratio_linear. This reduces the slope									

Parameter/Value Pair	Description
	for pressure ratios around 1. The value must fulfill the following condition: 0 < value $\leq$ 1 The default value is 1, which deactivates linearization.
'CheckRatio', <method>	The psi function is 0 if the pressure ratio is 1. If the input vector does not have a pressure ratio of 1, the next lower and higher points are interpolated. This might lead to results unequal to zero for pressure ratio 1. If <b>method</b> is 'on' (default), a check is made whether the pressure ratio vector contains 1. If not, it is extended with 1 and a warning is issued. If <b>method</b> is 'off', no check or modification on the pressure ratio vector is performed.

## Calculation algorithm

To simulate back flow, the definition of the psi function has been extended for pressure ratios greater than 1. In this case the reciprocal pressure ratio is used and the result is multiplied by -1.

The following formula shows the definitions used for the psi function.

$$\Psi\left(\frac{p_2}{p_1}, \kappa\right) = \begin{cases} \left(\frac{2}{\kappa+1}\right)^{\frac{1}{\kappa-1}} \sqrt{\frac{\kappa}{\kappa+1}}, & 0 < \frac{p_2}{p_1} < \left(\frac{2}{\kappa+1}\right)^{\frac{\kappa}{\kappa-1}} \\ \sqrt{\frac{\kappa}{\kappa-1} \left[ \left(\frac{p_2}{p_1}\right)^{\frac{2}{\kappa}} - \left(\frac{p_2}{p_1}\right)^{\frac{\kappa+1}{\kappa}} \right]}, & \left(\frac{2}{\kappa+1}\right)^{\frac{\kappa}{\kappa-1}} \leq \frac{p_2}{p_1} < p_{RLin} \\ pLinCoeff * \frac{1 - \left(\frac{p_2}{p_1}\right)}{1 - pRLin}, & p_{RLin} \leq \frac{p_2}{p_1} < 1 \\ 0, & \frac{p_2}{p_1} = 1 \\ pLinCoeff * \frac{1 - \left(\frac{p_2}{p_1}\right)}{1 - \frac{1}{pRLin}}, & 1 < \frac{p_2}{p_1} < \frac{1}{p_{RLin}} \\ -\sqrt{\frac{\kappa}{\kappa-1} \left[ \left(\frac{p_1}{p_2}\right)^{\frac{2}{\kappa}} - \left(\frac{p_1}{p_2}\right)^{\frac{\kappa+1}{\kappa}} \right]}, & \frac{1}{p_{RLin}} \leq \frac{p_2}{p_1} < \left(\frac{2}{\kappa+1}\right)^{-\frac{\kappa}{\kappa-1}} \\ -\left(\frac{2}{\kappa+1}\right)^{\frac{1}{\kappa-1}} \sqrt{\frac{\kappa}{\kappa+1}}, & \left(\frac{2}{\kappa+1}\right)^{-\frac{\kappa}{\kappa-1}} \leq \frac{p_2}{p_1} \end{cases}$$

With

$$pLinCoeff = \sqrt{\frac{\kappa}{\kappa - 1} \left[ (p_RLin)^{\frac{2}{\kappa}} - (p_RLin)^{\frac{\kappa + 1}{\kappa}} \right]}$$

Where

$\frac{p_2}{p_1}$  is the pressure ratio

$\kappa$  is the isentropic exponent kappa

$p_RLin$  is the input parameter `p_Ratio_linear`

#### Output structure

The return argument is a structure with the following fields:

Field	Content
<code>xName</code>	'pressure ratio'
<code>vName</code>	'psi'
<code>xUnit</code>	[]
<code>vUnit</code>	[]
<code>Comment</code>	'Flow function PSI, psi = f(pout/pin, kappa)'
<code>x</code>	Provided pressure ratio vector (may be extended with '1')
<code>v</code>	Calculated psi values for pressure ratio vector

## asm\_traffic\_mdl\_update\_num\_fellows

### asm\_traffic\_mdl\_update\_num\_fellows

---

<b>Purpose</b>	To change the maximum number of supported fellows in a traffic model.
----------------	---

<b>Basic concept</b>	This function allows you to change the maximum number of fellows supported by a traffic model.
----------------------	--

To update a model, perform the following steps. Here, the model is called **ASM\_Traffic**.

1. Open the `asm_traffic_ini.m` file of your ASM Traffic project folder. Go to `MDL.Traffic.MaxNumFellows.v` and modify the number of fellows.  
E.g.: `MDL.Traffic.MaxNumFellows.v = 50.`
2. In MATLAB, open the Simulink model with the `go` command.
3. Start the modification process via  
`asm_traffic_mdl_update_num_fellow('ASM_Traffic').`
4. Update the ModelDesk Plotting block via  
`asm_mdld_plot_block_update('ASM_Traffic').`

## CreateIndependentParameterxml.py

### CreateIndependentParameterxml.py

<b>Purpose</b>	To create parameter sets with defined parameter XML files.  You can use this function either to create independent parameter sets (if defined parameter XML files do not exist) or to sync parameter sets (if parameter XML files already exist).
<b>Description</b>	You can use the <code>CreateIndependentParameterxml.py</code> file to modify the links to the parameter XML files in a parameter set. You can find it in the Python Scripts folder in ModelDesk (%ModelDesk-Project%\PythonScripts) and directly edit the code for the individual use case.  The following use cases for this modification are possible: <ul style="list-style-type: none"> <li>▪ Creating an independent parameter set</li> <li>▪ Automatic renaming for a number of multi-instances (e.g., for the measurement interface within engine projects)</li> <li>▪ Synchronizing parameter sets to use the same parameter XML files</li> </ul>
<b>Basic concept</b>	The script supports three standard modes. You can select a mode by assigning the corresponding index to the mode with <code>Mode = Modes[index]</code> : <ul style="list-style-type: none"> <li>▪ <code>Modes[0] "SetParameterxmlName"</code>: The defined <code>ParameterxmlName</code> string is used as the file name.</li> <li>▪ <code>Modes[1] "ApplyPrefixandSuffixToFileName"</code>: The defined <code>NamePrefix</code> and <code>NameSuffix</code> strings are used to extend the existing file name. The new name is created by: <code>NamePrefix + ParameterRecord.FileName + NameSuffix</code></li> <li>▪ <code>Modes[2] "ApplyPrefixandSuffixToInstanceName"</code>: The defined <code>NamePrefix</code> and <code>NameSuffix</code> strings are used to extend the instance name. The new name is created by: <code>NamePrefix + ParameterRecord.InstanceName + NameSuffix</code></li> </ul>

#### Note

To familiarize with the function, you are recommended to start with a copy of your productive project or use an ASM demo.

## Definitions

Several definitions are available to control the result of the Python script.

It can be divided in two parts:

- How the new parameter XML file name is created. See [New XML file names](#) on page 221.
- To which parameter pages the name is applied in which way. See [Selecting specific parameter pages](#) on page 221.

In general, if the estimated parameter XML file name already exists for the related parameter page, the existing file is linked.

### NOTICE

Linking existing files can change the content of the parameter set.

If no suitable parameter XML file exists, a copy of the currently linked parameter XML file is created by *Save as* and linked. The original file is kept in the pool.

**New XML file names** Definitions for new XML file names:

- ParameterxmlName

Default: `ParameterxmlName = "Engine_A":`

Valid for `Modes[0]`.

With `Modes[0]`, all defined parameter pages are linked to a file name `ParameterxmlName`.

- NamePrefix and NameSuffix

Default: `NamePrefix = ParameterxmlName + "_"`

Default: `NameSuffix = ""`

Valid for `Modes[1]` and `Modes[2]`.

With `Modes[1]`, all defined parameter pages are renamed to `NamePrefix + ParameterRecord.FileName + NameSuffix`. With `Modes[2]`, the estimated parameter page name is `NamePrefix + ParameterRecord.InstanceName + NameSuffix`. Each field can also be left empty to define no prefix or suffix in the mode.

**Selecting specific parameter pages** Definitions to select specific parameter pages for modification:

- SkipMainComponents

Default: `SkipMainComponents = ["VehicleDynamics"]`.

This is a commaseparated list which can be extended as for example

`SkipMainComponents = ["VehicleDynamics", "DrivetrainBasic"]`

This list contains the main components (e.g., SoftECU, EngineDiesel, VehicleDynamics) that are skipped during modification. The main component names are shown in the parameter address, available in each parameter's Properties pane.

The parameter address is defined as

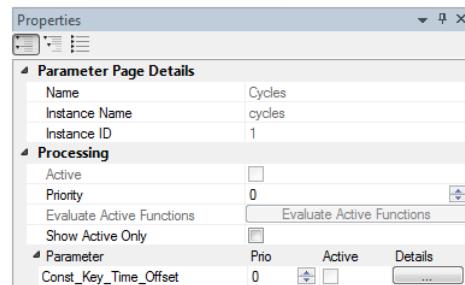
`Maincomponent.Subcomponent.Parameter`.

For ModelDesk custom components, the library is used as the main component.

- SkipRecords

Default: `SkipRecords = ["Cycles", "Test Cycle"]`

This list contains the records (i.e. a final tree element within the ModelDesk parameter set) that are skipped during modification. A corresponding record can be found either by manually browsing through the active parameter set object within Python or by checking the `Name` property in the Properties pane of the corresponding parameter page:



You can easily extend the list with further records:

```
SkipRecord = ["RecordName1", "RecordName2", "RecordName3", ...]
```

**Note**

The record name and the subcomponent name often have a 1:1 relationship, but this is not always the case.

- InstanceRecords

Default: `InstanceRecords = ["LUT1D", "LUT2D", "Angular Processing Unit Event"];`

Defines records which are always renamed with  
`"NamePrefix + ParameterRecord.InstanceName + NameSuffix"`.

This list must contain all records for which multi-instances exist and where each instance needs to be parameterized independently.

**Logging** During script execution, a log is written to the interpreter window. It provides information about which records were adapted and which main components and records were skipped. For adapted records, the log entries also state if a new file is created, an existing file is linked, or the correct file was already linked.

# CreateGeoOptTrajectory.py

## CreateGeoOptTrajectory.py

---

<b>Purpose</b>	To automatically define trajectories for vehicles in a road model. You can then export these trajectories to your ModelDesk project.
----------------	--

---

<b>Trajectories</b>	A trajectory describes a vehicle's path on a road. If no trajectory is defined, a vehicle simulation follows the road reference line (preferred lane) along the corresponding route. The default road reference line is generally not a geometrically optimal path.  If you want the vehicles to follow different reference paths on a route, you have to define different trajectories for the vehicles.  For more information on trajectories, refer to <a href="#">Trajectories (ModelDesk Road Creation)</a> .
---------------------	--

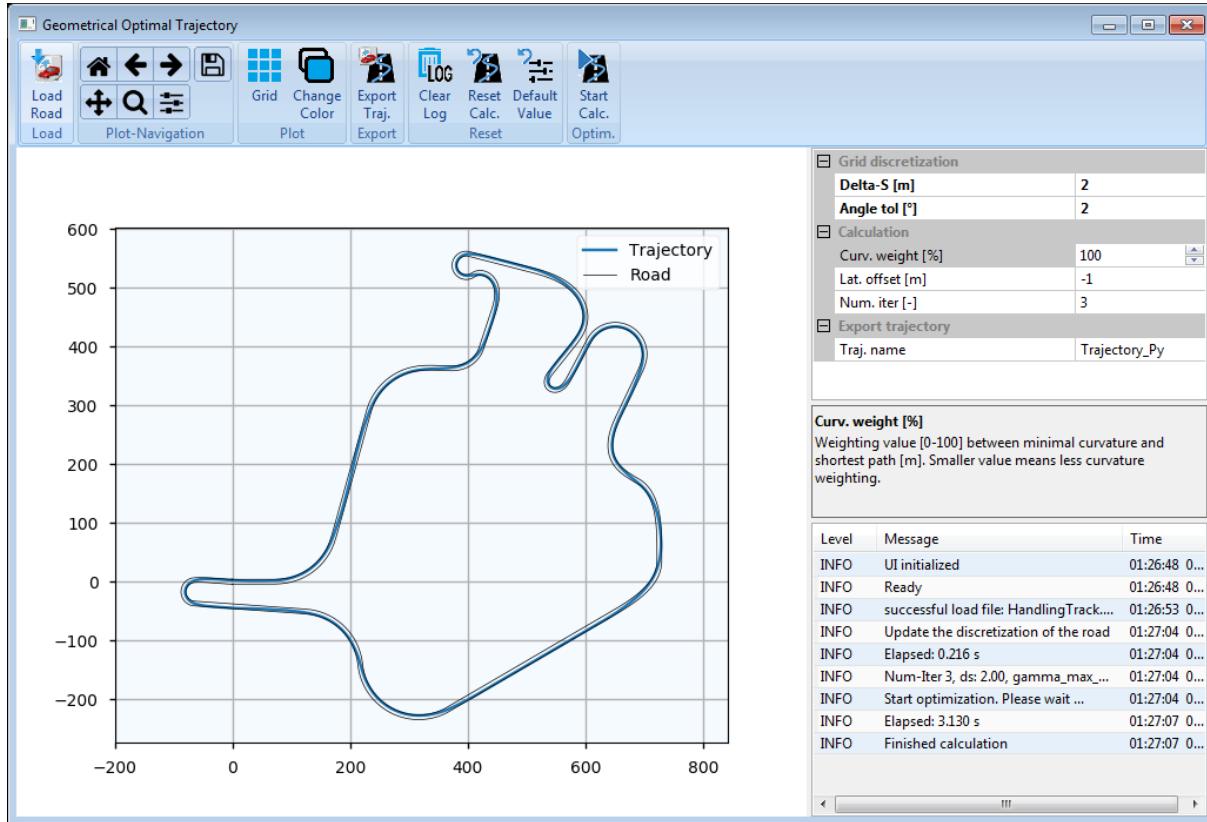
---

<b>Python script</b>	The Python script is called by <code>CreateGeoOptTrajectory.py</code> . The script is located in the ModelDesk project of the vehicle dynamics demo model.  It automatically calculates a geometrically optimal trajectory of a (closed) road. The calculation of the following trajectories is possible: <ul style="list-style-type: none"><li>▪ Geometrically optimal trajectory with minimal curvature.</li><li>▪ Geometrically optimal trajectory with shortest path length.</li><li>▪ Geometrically optimal trajectory with a compromise between minimal curvature and shortest path length.</li></ul>
----------------------	---

---

<b>User interface</b>	You can start a user interface for the geometrical trajectory calculation by running the script. It provides different parameters for the trajectory.
-----------------------	---

The user interface looks like this:



It contains the following elements:

#### Ribbon

Command	Description
Load	
Load Road	To load the currently active road model from ModelDesk.
Plot-Navigation	
	To reset the plot to the original view.
	To go back to the previous view in plot history.
	To go forward to the next view in plot history.
	To zoom on a specific part of the plot. Hold the left mouse button and specify the part of the plot with the rectangle.
	To save the current plot.
	To zoom or move the plot. <ul style="list-style-type: none"> <li>▪ Hold the right mouse button and move the mouse to zoom in and out.</li> </ul>

Command	Description
	<ul style="list-style-type: none"> <li>Hold the left mouse button and move the mouse to move the plot.</li> </ul>
<b>Plot</b>	To configure subplots. A dialog opens for you to configure different parameters.
Grid	To show/hide the grid in the Preview.
Change color	To change the background color in the Preview.
Export	
Export Traj.	To export the trajectory to your ModelDesk project.
Reset	
Clear Log	To clear the Log Viewer.
Reset Calc.	To reset the calculation.
Default Value	To reset the values in the Parameters pane to default.
Optim.	
Start Calc.	To start calculation of the trajectory.

**Log Viewer** To display messages generated when using the script.

**Preview** To display a preview of the calculated trajectory on a road.

**Parameters** To display and specify the parameters of the trajectory. The following parameters are available.

Name	Unit	Description
Grid discretization		
Delta-S	[m]	<p>Minimal discretization of the road path <math>s</math>. Minimum value = 1 m A smaller parameter value means more sampling points for the trajectory and requires more calculation performance.</p>
Angle tol.	[deg]	<p>Angle tolerance Minimum value = <math>0^\circ</math> Used to calculate an adaptive discretization. The angle between three discretization points is calculated. If this angle is bigger than the angle tolerance, new discretization points are calculated to satisfy the angle tolerance. A smaller parameter value means that more sampling points are generated and it requires more calculation performance.</p>
Calculation		
Curv. weight	[%]	<p>Curvature weight Value between 0% and 100%</p> <ul style="list-style-type: none"> <li>100%: The geometrical calculation of the trajectory considers only the minimal curvature.</li> <li>0%: The geometrical calculation of the trajectory considers only the shortest path length.</li> </ul>

Name	Unit	Description
Lat. offset	[m]	<ul style="list-style-type: none"> <li>A value between 0% and 100%: The geometrical calculation of the trajectory is a compromise between minimal curvature and shortest path length.</li> </ul> <p>Lateral offset To take the vehicle width into account in the geometrical calculation of the trajectory, define a lateral offset from the most outer road width. A negative value reduces the original road width.</p>
Num. iter.	[]	Number of iterations of the calculation
Export trajectory	[]	
Traj. name	[]	Name of the trajectory for export to ModelDesk

**Limitations**

The following limitations apply for the automatic calculation of the trajectory:

- The trajectory can be calculated for only one (closed/open) road, not for an entire road network of several roads and/or junctions.
- Height and lateral slope of the road are not considered.
- Surface conditions of the road are not considered.

**Related topics****Basics**

[Lanes of a Road Element \(ModelDesk Road Creation\)](#)  
[Trajectories \(ModelDesk Road Creation\)](#)

# ModelDesk Processing

---

## Introduction

To implement M functions for processing in ModelDesk. This section provides information on the required methods, describes the approach, and provides information on how to configure the processing infrastructure using the general settings file.

---

## Where to go from here

### Information in this section

[asm\\_proc Function](#)..... 228

To realize an interface between ModelDesk and MATLAB.

[Parameter Functions](#)..... 242

To handle parameters.

[Files for ModelDesk Processing](#)..... 247

The files are used for processing: general settings file, additional functions, measurement functions, and initialization files.

# asm\_proc Function

## Where to go from here

## Information in this section

<a href="#">Description of the asm_proc Function</a> .....	229
To realize an interface between ModelDesk and MATLAB.	
<a href="#">calc Method</a> .....	230
To evaluate all relevant parameters from the measurement data.	
<a href="#">check_parameter Method</a> .....	230
To check if a parameter or subcomponent exists.	
<a href="#">get_paramchilid Method</a> .....	231
To get all available children of a parameter.	
<a href="#">get_parameter Method</a> .....	232
To get various kinds of information on a parameter.	
<a href="#">set_parameter Method</a> .....	234
To set a specific parameter, e.g., while using additional functions.	
<a href="#">set_measurementnumber Method</a> .....	235
To set a defined number of measurements for the measurement type.	
<a href="#">check_measurement Method</a> .....	236
To check if a specific measurement variable exists.	
<a href="#">get_measurement Method</a> .....	237
To get the specified measurement data.	
<a href="#">set_measurement Method</a> .....	238
To save additional measurement data to a measurement structure.	
<a href="#">set_round Method</a> .....	238
To define a default rounding behavior when writing parameters with 'set_parameter'.	
<a href="#">write2log Method</a> .....	239
To trigger a log file message in ModelDesk's Message Viewer and log file with a predefined message text.	
<a href="#">get_plotconfig Method</a> .....	240
To get information about the configuration of plots.	

## Description of the asm\_proc Function

<b>Purpose</b>	To realize an interface between ModelDesk and MATLAB. This function triggers the calculation of all selected parameterization functions. Several functions inside <b>asm_proc</b> can be called.
----------------	--

<b>Getting help</b>	To get a list of available methods, type <code>asm_proc('help')</code> in the MATLAB Command Window.
	To get more information about a specific method, type <code>asm_proc(&lt;method&gt;, 'help')</code>

<b>Overview of methods</b>	The following table gives an overview of the methods which can be used with the <b>asm_proc</b> function.
----------------------------	---

Method	Purpose
'calc'	To evaluate all relevant parameters from measurement data.
'check_parameter'	To check if a parameter or subcomponent exists.
'get_paramchilds'	To get all available children of a parameter, i.e. the next layer of the structure. This function is also used for importing the workspace variables.
'get_parameter'	To get various kinds of information on a parameter.
'set_parameter'	To set a specific parameter, e.g., while using additional functions.
'set_measurementnumber'	To set a defined number of measurements for the measurement type.
'check_measurement'	To check if a specific measurement variable exists.
'get_measurement'	To get the specified measurement data.
'set_measurement'	To save calculated measurement data to a measurement structure. Refer to <a href="#">Measurement Functions</a> on page 252.
'set_round'	To set a default rounding behavior for parameters.
'write2log'	To trigger a message in ModelDesk's Message Viewer and log file with a predefined message text to provide sufficient feedback about an issue during a processing function run.
'get_plotconfig'	To get information about the configuration of plots. This is mainly for use relating to <b>asm_tablegenerator</b> . You can use it as a generic argument, so you do not have to define the plot configuration manually for each table.

<b>Related topics</b>	Basics
	<a href="#">Basics of Processing (ModelDesk Processing)</a> <a href="#">Workflow for Processing (ModelDesk Processing)</a>

## calc Method

**Purpose** To evaluate all relevant parameters from the measurement data. This function is called by ModelDesk.

**Arguments** The argument of this function is a structure that contains a list of parameter functions and corresponding setting information.

**Result** A structure containing all calculated parameters is returned to ModelDesk.

**Related topics**

Basics

Basics of Processing (ModelDesk Processing) 

Description of the `asm_proc` Function..... 229

## check\_parameter Method

**Purpose** To check if a parameter or subcomponent exists.

**Function call** Use the following call to check if the specified parameter exists in the parameter storage:

```
[flag,foundfields] = asm_proc('check_parameter','MainComp','SubComp',
...
SubComp,'ParaName',ParaName,'InstID',InstID,
'CheckType',[0 0 0 0])
```

The following table shows the specified input arguments:

Parameter	Description	Required/Optional
Address	Alternative definition by <MainComp>. <SubComp>. <ParaName>. <InstID>. <FieldName>	Required for Option 1 <sup>1)</sup>
MainComp	Main component whose existence is checked	Required for Option 2 <sup>1)</sup>
SubComp	Subcomponent whose existence is checked	Optional
ParaName	Parameter name whose existence is checked	Optional
InstID	Instance ID whose existence is checked	Optional
FieldName	Field below the parameter name whose existence is checked	Optional

Parameter	Description	Required/Optional
CheckType	<p>Array with the length of the fields to check. First relates to the main component, second to the subcomponent and so on. Default is [1 1 1 1 0].</p> <p>True throws an error if the field does not exist.</p> <p>False just writes to the return flag.</p>	Optional

<sup>1)</sup> You can provide the desired parameter either as a dot-separated string representing the address (Option 1) or as single arguments containing the main component, subcomponent, and so on (Option 2). In the case of the address string, the entries after the main component are optional.

## Result

The output arguments have the following meanings:

Field	Description
FoundFlag	Indicates if the specified parameter was found.
FoundFields	Returns the found fields as a cell array. This represents the address of the found field.

If a field with **CheckType 0** does not exist, an informative error message is displayed.

## Related topics

### Basics

Basics of Processing (ModelDesk Processing 	.....	229
Description of the asm_proc Function.....		

## get\_paramchilds Method

### Purpose

To get all available children of a parameter, i.e. the next layer of the structure. This function is also used for importing the workspace variables.

### Function call

Use the following call to get the children of a specified parameter:

```
asm_proc('get_paramchilds','MainComp',MainComp,'SubComp',...
SubComp,'ParaName',ParaName,'InstID',InstID)
```

Parameter	Description	Required/Optional
Address	Alternative definition by <MainComp>. <SubComp>. <ParaName>. <InstID>	Required for Option 1 <sup>1)</sup>
MainComp	Main component of which all available children are found	Required for Option 2 <sup>1)</sup>

Parameter	Description	Required/Optional
SubComp	Subcomponent of which all available children are found	Optional
ParaName	Name of the parameter of which all available children are found	Optional
InstID	Instance ID of which all available children are found	Optional

<sup>1)</sup> You can provide the desired parameter either as a dot-separated string representing the address (Option 1) or as single arguments containing main component, subcomponent, and so on (Option 2). In the case of the address string, the entries after the main component are optional.

---

**Result** The result structure contains all the available children of the desired parent.

---

**Related topics** Basics

<a href="#">Basics of Processing (ModelDesk Processing)</a>	
<a href="#">Description of the asm_proc Function</a>	.....
	229

## get\_parameter Method

---

**Purpose** To get various kinds of information on a parameter.

---

**Function call** **Case 1** To get information on the parameter value and axis, use the following call:

```
asm_proc('get_parameter', 'MainComp', MainComp, ...
         'SubComp', SubComp, 'ParaName', ParaName, ...
         'InstID', InstID, 'readfrom', mfilename)
```

Parameter Name	Parameter Value	Required/Optional	Example
MainComp	Main component of parameter to be read	Required	EngineGasoline
SubComp	Subcomponent of parameter to be read	Required	exhaust_manifold
ParaName	Parameter name of parameter to be read	Required	Map_T_ExhMan
InstID	Instance ID of parameter to be read	Required	ID1

Parameter Name	Parameter Value	Required/Optional	Example
readfrom	Identifier: e.g., function name which reads the parameter	Optional	Mfilename (returns name of the executed file)

**Case 2** To get information on the parameter dimension, axis names and units, use the following call:

```
asm_proc('get_parameter','MainComp',MainComp, ...
'SubComp',SubComp,'ParaName',ParaName, ...
'InstID',InstID,'FieldName','Infofields')
```

Parameter Name	Parameter Value	Required/Optional	Example
MainComp	Main component of parameter to be read	Required	EngineGasoline
SubComp	Subcomponent of parameter to be read	Required	exhaust_manifold
ParaName	Parameter name of parameter to be read	Required	Map_T_ExhMan
InstID	Instance ID of parameter to be read	Required	ID1
FieldName	Field name: if not specified, all fields are returned as struct	Optional	Infofields

## Result

**Case 1** The returned structure in case 1 contains the following fields:

Field	Description
Record	Subcomponent name
Path	Location in Md1DProcessing structure
Comment	Comment
TypeName	Parameter type: e.g., Scalar, LUT2D
RealTimePath	Location in Simulink model
MaskVariable	Mask entry in Simulink model
ModificationTime	Date of last modification
Function	Function used for calculation
Setting	Settings file used for calculation
Status	Status
LastFunctionRun	Date of last evaluation
Priority	Calculation priority
DimV	Maximum size of v
DimView	Used size of v

Field	Description
xName	Name of x-axis (if it exists)
xUnit	Name of x unit (if it exists)
x	Value of x-axis (if it exists)
yName	Name of y-axis (if it exists)
yUnit	Name of y unit (if it exists)
y	Value of y-axis (if it exists)
vName	Name of v-axis
vUnit	Name of v unit
v	Value of v-axis
ReadBy	List of functions that have used this parameter <sup>1)</sup>

<sup>1)</sup> The ReadBy field is used to indicate if a function was read before it is written. This means that the flag is set if x, y or v is returned. If these fields are set at a later time, a warning is issued.

**Case 2** The returned structure in case 2 contains the same fields as in case 1, except for x, y, v and ReadBy.

## Related topics

### Basics

Basics of Processing (ModelDesk Processing 	229
Description of the asm_proc Function.....	229

## set\_parameter Method

### Purpose

To set a specific parameter, e.g., while using additional functions.

### Function call

Use the following call to set the parameter:

```
asm_proc('set_parameter', 'MainComp', MainComp, 'SubComp', ...
SubComp, 'ParaName', ParaName, 'InstID', InstID, ...
'Value', Value)
```

Parameter	Description	Required/Optional
Address	Alternative definition by <MainComp>. <SubComp>. <ParaName>. <InstID>. <Value>	Required for Option 1 <sup>1)</sup>
MainComp	Main component of the parameter to set	Required for Option 2 <sup>1)</sup>

Parameter	Description	Required/Optional
SubComp	Subcomponent of the parameter to set	Required for Option 2 <sup>1)</sup>
ParaName	Parameter name of the parameter to set	Required for Option 2 <sup>1)</sup>
InstID	Instance ID of the parameter to set	Required for Option 2 <sup>1)</sup>
Value	Structure with parameter values	Required for Option 2 <sup>1)</sup>

<sup>1)</sup> You can provide the desired parameter either as a dot-separated string representing the address (Option 1) or as single arguments containing main component, subcomponent, and so on (Option 2). In the case of the address string, all entries are required.

Another possible function call is:

```
asm_proc('set_parameter','MainComp',MainComp,'SubComp',...
'SubComp','ParaName',ParaName,'InstID',InstID,....
'Value',Value,'Round',{N,'Type'})
```

Parameters	Description	Required/Optional
Round	Cell array with number of digits 'N' and rounding type 'Type' Both parameters are required.	Optional

## Result

The parameter is set to the internally used parameter storage and transferred to ModelDesk later. It is then available for further use in ModelDesk.

Several checks are included to ensure that only plausible values can be written:

- DimV check
- Check for existing fields
- Writing back the result is skipped if the x, y and v fields do not exist

## Related topics

### Basics

Basics of Processing (ModelDesk Processing  )	.....	229
Description of the asm_proc Function.....		

## set\_measurementnumber Method

### Purpose

To set a defined number of measurements for the measurement type.

**Function call**

The number can only be set once during one processing execution. Changing the measurement number resets all measurement variables of the related measurement type to its default value. Use the following call to set the number of measurements:

```
asm_proc('set_measurementnumber', 'MeasurementType', ...
MeasurementType, 'Number', Number)
```

Parameter	Description	Required/Optional
MeasurementType	Measurement type to set the number of measurements	Required
Number	Number of measurements	Required

**Result**

The number of measurements is set for the specified measurement type in the measurement store.

**Related topics****Basics**

<a href="#">Basics of Processing (ModelDesk Processing)</a>	229
<a href="#">Description of the asm_proc Function</a>	.....

## check\_measurement Method

**Purpose**

To check if a specific measurement variable exists.

**Function call**

Use the following call to check the measurement data:

```
asm_proc('check_measurement', 'MeasurementType', ...
MeasurementType, 'Variable', Variable, 'CheckType', CheckType)
```

Parameter	Description	Required/Optional
MeasurementType	Measurement type of the measurement data whose existence is checked	Required
Variable	Variable name of the measurement data whose existence is checked	Optional
CheckType	Array with the length of the fields to check. The first relates to MeasurementType, the second to Variable. <b>True</b> throws an error if the	Optional

Parameter	Description	Required/Optional
	field does not exist. <code>False</code> just writes to return flag. Default is [1 1].	

---

**Result** Returns a flag that indicates whether the specified MeasurementType and Variable of the measurement data exist in the measurement storage or throws an error if it does not exist.

---

**Related topics** Basics

[Basics of Processing \(ModelDesk Processing\)](#)  
[Description of the asm\\_proc Function](#)..... 229

## get\_measurement Method

---

**Purpose** To get the specified measurement data.

---

**Function call** Use the following call to get measurement data:

```
asm_proc('get_measurement','MeasurementType',MeasType,...  

'Variable',Variable)
```

---

**Result** The returned structure contains the following fields:

Field	Description
vName	Name of measurement
vUnit	Unit of measurement
v	Measurement data
ReadBy	List of functions that have used this measurement data

---

**Related topics** Basics

[Basics of Processing \(ModelDesk Processing\)](#)  
[Description of the asm\\_proc Function](#)..... 229

## set\_measurement Method

---

<b>Purpose</b>	To save calculated measurement data to a measurement structure. Refer to <a href="#">Measurement Functions</a> on page 252.
----------------	---

---

<b>Function call</b>	Use the following call to set measurement data:
----------------------	---

```
asm_proc('set_measurement','MeasurementType',MeasurementType,  
'Variable',Variable,'Value',Value)
```

Parameter	Description	Required/Optional
MeasurementType	Measurement type of the variable to write	Required
Variable	Name of the variable to write	Required
Value	Variable value struct with the fields v and vUnit	Required

---

<b>Result</b>	The specified measurement variable is updated in the internally used measurement storage and is transferred to ModelDesk later. It is then available for further use by ModelDesk, e.g., for processing executions where the calculation of the measurement variable is not included.
---------------	---

---

<b>Related topics</b>	Basics
-----------------------	--------

Basics of Processing (ModelDesk Processing 	229
Description of the asm_proc Function.....	229
Measurement Functions.....	252

## set\_round Method

---

<b>Purpose</b>	To define a default rounding behavior when writing parameters with <code>set_parameter</code> .
----------------	---

---

<b>Function call</b>	Use the following function call to set the rounding behavior:
----------------------	---

```
asm_proc('set_round','N',N,'Type',Type)
```

Parameter	Parameter	Description	Required/Optional
	N	Number of digits to round to. 6 is the default.	Optional
	Type	The rounding type can be one of the following: <ul style="list-style-type: none"><li>▪ Decimals</li><li>▪ Significant</li></ul> <i>Significant</i> is the default.	Optional

To change the default values for all parameters written by `asm_proc`, it is recommended to include this method in the general settings function. It ensures the same rounding behavior across all parameter functions.

To change the precision of a single parameter, you can specify input values in the `set_parameter` method of `asm_proc`.

Result	The ModelDesk parameter called by <code>asm_proc</code> is rounded according to the number of digits and type specified. If you do not specify these parameters, the parameter is rounded to the sixth significant digit by default.
--------	--

Related topics	Basics
	<a href="#">Description of the <code>asm_proc</code> Function</a> ..... 229 <a href="#">set_parameter Method</a> ..... 234

## write2log Method

Purpose	To trigger a log file message in the ModelDesk's Message Viewer and log file with a predefined message text to provide sufficient feedback about an issue during a processing function run.
---------	---

Function call	Use the following call to trigger a log message:
	<pre>asm_proc('write2log','Severity',Severity,'Level',WarningLevel, 'Target','LogFile','Identifier','asm:processing:WarnID', ... 'Message',sprintf('Reduced y Dimension from %i to %i', size(Value.v,2),OutMdlStruct.DimV(2)));</pre>

Parameter	Parameter	Description
	<b>Severity</b>	Severity of message: "Info", "Warning", "Error"; "Error" is the default
	<b>WarningLevel</b>	This is the warning hierarchy level of the message as it appears in the ModelDesk log: "0", "1", ...; 1 is the default
	<b>WarnID</b>	The warning is tagged with this identifier. It can be used to enable or disable the display of warnings with this identifier. If empty, "asm:processing:general" is used.

**Related topics****Basics**

[Basics of Processing \(ModelDesk Processing\)](#)  
[Description of the asm\\_proc Function](#)..... 229

## get\_plotconfig Method

**Purpose**

To get information about the configuration of plots. This is mainly for use relating to `asm_tablegenerator`. You can use it as a generic argument, so you do not have to define the plot configuration manually for each table.

**Function call**

Use the following call to get the plot configuration for the use project:

```
asm_proc('get_plotconfig')
```

**Result**

The returned structure contains the following fields:

Field	Description
<b>FinalResult</b>	Overall indicator if plots are enabled
<b>InputData</b>	Indicates if single measurement points are shown
<b>InputDataLabel</b>	Indicates if measurement points are labeled
<b>ErrorPlot</b>	Indicates if error plots are enabled
<b>Incremental</b>	Indicates if incremental plots are enabled
<b>IncrementalPause</b>	Delay between incremental plots
<b>MaxNumPlots</b>	Maximum number of plots

**Related topics**

**Basics**

asm_tablegenerator.....	205
Basics of Processing (ModelDesk Processing  <td></td>	
Description of the asm_proc Function.....	229

# Parameter Functions

## Where to go from here

## Information in this section

[Example of Calculating a Parameter in a Function](#)..... 242

To give an example of a function which calculates a specific parameter.

[ImportParamfromWorkspace.m Function](#)..... 246

To transfer a single parameter from the MATLAB workspace to ModelDesk.

## Example of Calculating a Parameter in a Function

### Purpose

To give an example of a function which calculates a specific parameter.

### Overview of the sections

A function consists of several sections:

1. Function name and input arguments
2. Header with information
3. Settings
4. Measurement data
5. Parameters
6. Parameter attributes
7. Processing and plotting

### Description of example

The example shows a function that creates a table of the temperature in the exhaust manifold for each engine operating point.

**Function name and input arguments** Input arguments have to be given in the form: Settings, MainComp, SubComp, ParaName, InstID, varargin. varargin can be used if additional parameters or information is needed.

```
function [OutMdlStruct] = Map_T_ExhManifold (Settings, MainComp, SubComp, ParaName, InstID, varargin)
```

**Header with information** The header can be created by ModelDesk.

```
%%
%
% INPUT:
%   Settings, MainComp, SubComp, ParaName, ID, varargin
%
% FUNCTION:
% calculates the temperature in the exhaust manifold according to measurement data and engine operating point
%
%
% Copyright (c) 2014 by dSPACE GmbH, GERMANY
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INSERT YOUR CODE HERE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The following table shows the specified input arguments:

Argument	Description
Settings	Merged structure of General settings and settings defined in the local settings file
MainComp	String for corresponding main component of the block, which contains this parameter
SubComp	String for corresponding subcomponent of the block, which contains this parameter
ParaName	String for the corresponding parameter name
InstID	String for the corresponding instance ID of the block (for more information on multi-instance handling, refer to <a href="#">asm_multiinstance</a> on page 201)

For a better understanding of these parameters, the information on parameter addresses in ModelDesk might be useful. The parameter address in ModelDesk is uniquely defined as

`MainComponent.Subcomponent.ParameterName.InstanceID`. The `MainComponent` defines the primary tree element in which the corresponding block is located (e.g. EngineDiesel or DrivetrainBasic). The `SubComponent` is related to the block itself (e.g. egr\_cooler or crankshaft). The `ParameterName` corresponds to the name of the parameter in the internal block parameter workspace (e.g. `Map_eta_Cooler` or `Const_Inertia_Mod`). The `InstanceID` refers to the ID of the block instance (e.g. 2 for a second instance).

**Settings** Settings that are needed for the actual calculation should be accessed here. In this example the settings provide information about the engine operating point and the axis vectors for map generation.

```
%% Settings
MeasTypeEng      = Settings.(MainComp).MeasTypeEng.v;
EngOPSSignal     = Settings.(MainComp).EngOPSSignal;
```

```
% axis for map generation
Vector_EngOP = cell(size(EngOPSignal));
for idx = 1:length(EngOPSignal)
    Vector_EngOP{idx} = Settings.(MainComp).([ 'Vector_ ', EngOPSignal(idx).v]);
end
```

**Measurement data** The measurement data of the defined engine operating point and the exhaust manifold temperature is read by calling `asm_proc`.

```
% Measurement data
Data_EngOP = cell(size(EngOPSignal));
for idx = 1:length(EngOPSignal)
    Data_EngOP{idx} = asm_proc('get_measurement','MeasurementType',MeasTypeEng,'Variable',EngOPSignal(idx).v);
end
T_ExhMan = asm_proc('get_measurement','MeasurementType',MeasTypeEng,'Variable','T_ExhMan');
```

**Parameters** Parameters that are used for table generation are read by calling `asm_proc`. In this example the environment reference temperature is needed.

```
% Parameters
T_Environ_Ref = asm_proc('get_parameter','MainComp',MainComp,'SubComp','common_engine_parameters',...
    'ParaName','T_Environ_Ref','InstID',InstID,'readfrom',mfilename);
T_Environ_Ref = T_Environ_Ref.v - 273.15;
```

**Parameter attributes** With this call of `asm_proc` the additional information of the calculated parameter is read. For example, the returned structure contains `DimV`, `vName`, `vUnit`.

```
% Parameter attributes
OutMdlStruct = asm_proc('get_parameter','MainComp',MainComp,'SubComp',SubComp,...
    'ParaName',ParaName,'InstID',InstID,'FieldName','Infofields');
Idx_EngOPSignal = asm_proc('get_parameter','MainComp',MainComp,'SubComp',SubComp,'ParaName',...
    ['Const_Idx_EngOP', ParaName(4:end)],'InstID',InstID,'FieldName','v','Default',[1 2]);
```

**Skip calculation** By adding the `SkipReason` structure field to the `OutMdlStruct` variable, the calculation can be aborted (by using `return`). The result will not be passed to ModelDesk and a predefined warning will be generated. This might be useful if insufficient measurement data is detected by the processing function during a function run and the existing data should not be overwritten.

```
%% Skip calculation if no positive mass flow available
if ~ any(mdot_Valve.v >0)
    OutMdlStruct.SkipReason = 'Measurement does not contain massflow';
    return
end
```

**Processing and Plotting** In this section the actual calculation of the map is done. In this example, the `asm_tablegenerator` is used. For information on how to use `asm_tablegenerator`, refer to [asm\\_tablegenerator](#) on page 205.

```
%> Processing & Plotting
OutMdlStruct = asm_tablegenerator('2D_Table',...
    'table',OutMdlStruct, ...
    'meas_data',{Data_EngOP{Idx_EngOPSignal},T_ExhMan}, ...
    'x_axis',Vector_EngOP{Idx_EngOPSignal(1)}, ...
    'y_axis',Vector_EngOP{Idx_EngOPSignal(2)}, ...
    'x_down',{'constant'}, ...
    'x_up',{'constant'}, ...
    'y_down',{'constant'}, ...
    'y_up',{'smooth_constant'}, ...
    'xvset',[{0,T_Environ_Ref}], ...
    'PlotConfig',asm_proc('get_PlotConfig')));
return
```

**Example function**

The complete function is shown in the following listing.

```
function [OutMdlStruct] = Map_T_ExhManifold (Settings, MainComp, SubComp, ParaName, InstID, varargin)

%%
%
% INPUT:
%   Settings, MainComp, SubComp, ParaName, ID, varargin
%
% FUNCTION:
% calculates the temperature in the exhaust manifold according to measurement data and engine operating point
%
%
% Copyright (c) 2014 by dSPACE GmbH, GERMANY
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INSERT YOUR CODE HERE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Settings
MeasTypeEng      = Settings.(MainComp).MeasTypeEng.v;
EngOPSignal      = Settings.(MainComp).EngOPSignal;
% axis for map generation
Vector_EngOP = cell(size(EngOPSignal));
for idx = 1:length(EngOPSignal)
    Vector_EngOP{idx} = Settings.(MainComp).{['Vector_',EngOPSignal(idx).v]};
end
% Measurement data
Data_EngOP = cell(size(EngOPSignal));
for idx = 1:length(EngOPSignal)
    Data_EngOP{idx} = asm_proc('get_measurement','MeasurementType',MeasTypeEng,'Variable',EngOPSignal(idx).v);
end
T_ExhMan      = asm_proc('get_measurement','MeasurementType',MeasTypeEng,'Variable','T_ExhMan');
% Parameters
T_Environ_Ref = asm_proc('get_parameter','MainComp',MainComp,'SubComp','common_engine_parameters',...
    'ParaName','T_Environ_Ref','InstID',InstID,'readfrom',mfilename);
T_Environ_Ref = T_Environ_Ref.v - 273.15;
% Parameter attributes
OutMdlStruct = asm_proc('get_parameter','MainComp',MainComp,'SubComp',SubComp,...
    'ParaName',ParaName,'InstID',InstID,'FieldName','Infofields');
Idx_EngOPSignal = asm_proc('get_parameter','MainComp',MainComp,'SubComp',SubComp,'ParaName',...
    ['Const_Idx_EngOP', ParaName(4:end)],'InstID',InstID,'FieldName','v','Default',[1 2]);
```

```
%> Processing & Plotting
OutMdlStruct = asm_tablegenerator('2D_Table',...
    'table',OutMdlStruct,...
    'meas_data',{Data_EngOP{Idx_EngOPSignal},T_ExhMan},...
    'x_axis',Vector_EngOP{Idx_EngOPSignal(1)},...
    'y_axis',Vector_EngOP{Idx_EngOPSignal(2)},...
    'x_down',{'constant'},...
    'x_up',{'constant'},...
    'y_down',{'constant'},...
    'y_up',{'smooth_constant'},...
    'xvset',[{[0,T_Environ_Ref]}],...
    'PlotConfig',asm_proc('get_PlotConfig'));
return
```

## ImportParamfromWorkspace.m Function

### Purpose

To transfer a single parameter from the MATLAB workspace to ModelDesk.

Choose this function in ModelDesk's Processing Configuration pane for each parameter that you want to transfer from the MATLAB workspace. This will replace any calculation of the parameter.

# Files for ModelDesk Processing

## Where to go from here

## Information in this section

<a href="#">General Settings File</a>	247
To define the settings that are used throughout the project.	
<a href="#">Additional Functions</a>	250
To define additional functions, which can be executed during an execution process.	
<a href="#">Measurement Functions</a>	252
To consider measurement data that is not contained in the original measurement data file.	
<a href="#">Initialization Files</a>	253
To manipulate specific parameters during start-up.	

## General Settings File

### Purpose

To define the settings that are used throughout the project. It is also used to control several additional features called in pre- and post-functions.

### Features overview

The general settings file can be used for the following features:

- First and second instance of the engine's main component
- Measurement types
- Engine operating point
- Axis definitions of vectors
- Measurement interface
- Settings for additional functions

### Feature description

Here is some more detailed information on the features mentioned above.

**First and second instance of the engine's main component** There are two instances of engine main components in the demo settings file. These instances are mainly used in the SoftECU functions as they are designed independently of engine types. Another reason is the possibility to use diesel and gasoline engines in the same model. The definition of main component instances in the settings file looks like the following extract.

```
% =====
Settings.MainCompInst(1).Author      = 'dSPACE';
Settings.MainCompInst(1).Origin      = 'standard';
Settings.MainCompInst(1).Version      = '1.0';
Settings.MainCompInst(1).LastModified = 'DD-MM-YYYY';
Settings.MainCompInst(1).Comment      = 'Referenced Engine Maincomponent for first instance in SoftECU, Driver';
Settings.MainCompInst(1).vName        = 'ECUType';
Settings.MainCompInst(1).vUnit        = '[]';
Settings.MainCompInst(1).v           = 'EngineGasoline';
% =====
Settings.MainCompInst(2).Author      = 'dSPACE';
Settings.MainCompInst(2).Origin      = 'standard';
Settings.MainCompInst(2).Version      = '1.0';
Settings.MainCompInst(2).LastModified = 'DD-MM-YYYY';
Settings.MainCompInst(2).Comment      = 'Referenced Engine Maincomponent for second instance in SoftECU, Driver';
Settings.MainCompInst(2).vName        = 'ECUType';
Settings.MainCompInst(2).vUnit        = '[]';
Settings.MainCompInst(2).v           = 'EngineDiesel';
% =====
```

Here is an example of using this structure to access the right engine's main component in a soft ECU function:

```
%% Settings
MainCompInst      = Settings.MainCompInst(str2double(strrep(InstID,'ID','')));
MeasTypeEng       = Settings.(MainCompInst).MeasTypeEng.v;
```

**Measurement types** The measurement types that are used for calculation are defined here. The value of the v field must be the name of the measurement type in ModelDesk.

```
% =====
Settings.EngineGasoline.MeasTypeEng.Author      = 'dSPACE';
Settings.EngineGasoline.MeasTypeEng.Origin      = 'standard';
Settings.EngineGasoline.MeasTypeEng.Version      = '1.0';
Settings.EngineGasoline.MeasTypeEng.LastModified = 'DD-MM-YYYY';
Settings.EngineGasoline.MeasTypeEng.Comment      = 'Name of InData Structfield of Engine Measurement';
Settings.EngineGasoline.MeasTypeEng.vName        = 'MeasTypeEng';
Settings.EngineGasoline.MeasTypeEng.vUnit        = '[]';
Settings.EngineGasoline.MeasTypeEng.v           = 'Engine';
% =====
Settings.EngineGasoline.MeasTypeFric.Author      = 'dSPACE';
Settings.EngineGasoline.MeasTypeFric.Origin      = 'standard';
Settings.EngineGasoline.MeasTypeFric.Version      = '1.0';
Settings.EngineGasoline.MeasTypeFric.LastModified = 'DD-MM-YYYY';
Settings.EngineGasoline.MeasTypeFric.Comment      = 'Name of InData Structfield of Engine Measurement';
Settings.EngineGasoline.MeasTypeFric.vName        = 'MeasTypeFric';
Settings.EngineGasoline.MeasTypeFric.vUnit        = '[]';
Settings.EngineGasoline.MeasTypeFric.v           = 'Friction';
% =====
```

**Engine operating point** Several functions calculate maps which depend on the engine operating point. The engine operating point is therefore defined in the general settings and available to all functions.

```
% =====
Settings.EngineGasoline.EngOPSSignal(1).Author      = 'dSPACE';
Settings.EngineGasoline.EngOPSSignal(1).Origin       = 'standard';
Settings.EngineGasoline.EngOPSSignal(1).Version      = '1.0';
Settings.EngineGasoline.EngOPSSignal(1).LastModified = 'DD-MM-YYYY';
Settings.EngineGasoline.EngOPSSignal(1).Comment      = 'Fits to first signal in engine operating points vector';
Settings.EngineGasoline.EngOPSSignal(1).vName        = 'EngOPSSignal(1)';
Settings.EngineGasoline.EngOPSSignal(1).vUnit        = '[rpm]';
Settings.EngineGasoline.EngOPSSignal(1).v            = 'n_Engine';
% =====
Settings.EngineGasoline.EngOPSSignal(2).Author      = 'dSPACE';
Settings.EngineGasoline.EngOPSSignal(2).Origin       = 'standard';
Settings.EngineGasoline.EngOPSSignal(2).Version      = '1.0';
Settings.EngineGasoline.EngOPSSignal(2).LastModified = 'DD-MM-YYYY';
Settings.EngineGasoline.EngOPSSignal(2).Comment      = 'Fits to second signal in engine operating points vector';
Settings.EngineGasoline.EngOPSSignal(2).vName        = 'EngOPSSignal(2)';
Settings.EngineGasoline.EngOPSSignal(2).vUnit        = '[-]';
Settings.EngineGasoline.EngOPSSignal(2).v            = 'RelAirmass';
% =====
```

**Axis definitions of vectors** You can define vectors that are used by several functions in the general settings file. This is for consistency when generating maps for different parameters that have the same axis information.

```
% =====
Settings.EngineGasoline.Vector_n_Engine.Author      = 'dSPACE';
Settings.EngineGasoline.Vector_n_Engine.Origin       = 'standard';
Settings.EngineGasoline.Vector_n_Engine.Version      = '1.0';
Settings.EngineGasoline.Vector_n_Engine.LastModified = 'DD-MM-YYYY';
Settings.EngineGasoline.Vector_n_Engine.Comment      = 'Engine speed vector for map generation';
Settings.EngineGasoline.Vector_n_Engine.vName        = 'n_Engine';
Settings.EngineGasoline.Vector_n_Engine.vUnit        = '[rpm]';
% Please make sure that these interpolation points cover the measurement data
% it should start with an engine speed that is a bit smaller than the
% measurement data
Settings.EngineGasoline.Vector_n_Engine.v           = [0 250 500 750 1000 1250 1500 1750 2000 2500 3000 3500 4500 5000 5500
6000 6500 7000];
% =====
Settings.EngineGasoline.Vector_q_Mean_Inj.Author      = 'dSPACE';
Settings.EngineGasoline.Vector_q_Mean_Inj.Origin       = 'standard';
Settings.EngineGasoline.Vector_q_Mean_Inj.Version      = '1.0';
Settings.EngineGasoline.Vector_q_Mean_Inj.LastModified = 'DD-MM-YYYY';
Settings.EngineGasoline.Vector_q_Mean_Inj.Comment      = 'Injection quantity vector for map generation';
Settings.EngineGasoline.Vector_q_Mean_Inj.vName        = 'q_Mean_Inj';
Settings.EngineGasoline.Vector_q_Mean_Inj.vUnit        = '[mm3|cyc]';
Settings.EngineGasoline.Vector_q_Mean_Inj.v           = [0:3:100];
% =====
```

**Measurement interface** Definitions for the measurement interface in your model. The measurement interface generation is activated with the corresponding additional function.

```
% =====
```

```

Settings.GenerateMeasuremenInterface(1).Active = true;
Settings.GenerateMeasuremenInterface(1).MeasurementTypeName = 'Engine';
Settings.GenerateMeasuremenInterface(1).ModelName = 'Engine_MeasurementInterface';
Settings.GenerateMeasuremenInterface(1).ModelPath = fileparts(mfilename('fullpath'));
Settings.GenerateMeasuremenInterface(1).Calculated = {...% activation, result signal, operation (+ or *) , first
signal, second signal or factor
1,'p_Out_Comp[Pa]', '+', 'p_Ambient[Pa]', 'p_Out_Comp_Rel[Pa]';
1,'p_ExhMan[Pa]', '+', 'p_Ambient[Pa]', 'p_ExhMan_Rel[Pa]';
1,'p_InMan[Pa]', '+', 'p_Ambient[Pa]', 'p_InMan_Rel[Pa]';
};

% =====

```

**Settings for additional functions** You can add further settings required for additional functions to the general settings file.

**Writing measurement data** Use this feature to write the measurement data of a specific measurement type to an M file. The generated file can be imported by ASM Engine Testbench. The WriteMeasFile structure field displays the command to generate a measurement data file. The interpretation of the assigned is: *{string1, string2}* with *string1* as the absolute path to the GeneralSettings-file and *string2* as the name of the measurement type. The template in the general settings file shows a method to evaluate the path to the RawData folder within the parameterization project, based on relative path information by using the 'fileparts' command in MATLAB.

```

% Uncomment the following line to write the measurement data for a specific measurement type to an m-file (e.g. for ASM
Test bench)
% Filepath, Measurement Type
Settings.WriteMeasFile
= {fullfile(fileparts(fileparts(fileparts(fileparts(fileparts(mfilename('fullpath')))))), 'RawData', 'Gasoline6C
y12p91_Engine_Mapping.m'), 'Engine'};

```

## Related topics

### Basics

[Managing Measurement, Setting, Function, and Additional Function Files  
\(ModelDesk Processing\)](#)

## Additional Functions

### Purpose

To define additional functions, which can be executed during an execution process.

### Description

The ModelDesk Processing configuration lets you define additional functions, which can be executed during an execution process. For more information, refer to [Managing Measurement, Setting, Function, and Additional Function Files  
\(ModelDesk Processing\)](#).

To support you during the parameterization process, ASM provides the following functions, which can be used in the parameterization project.

---

<b>ImportMDLfromWorkspace.m</b>	<p>To transfer parameters from the MATLAB workspace to ModelDesk. This can be useful if parameters changed during online calibration or you need to transfer parameters from an optimized offline simulation to ModelDesk. In this function you can either use an include list to specify the subcomponents to be transferred from the MATLAB workspace or you can use an exclude list. In the exclude list, you can specify the subcomponents to be not transferred.</p>						
<b>GenerateMeasurementInterface.m</b>	<p>Use this feature to generate the measurement interface subsystem in a separate model. The generated model will be stored in the ModelDesk project next to the GeneralSettings file at file level (e.g. in Pool\Processing\setting\General). You can copy this subsystem to the model, which is referenced by ModelDesk. After copying the subsystem into the model to <b>MDLUserInterface\Environment\MDL_MEAS</b>, an updated model will include the look-up tables, which contain the measurement data, in the project in <b>Environment Basic - Measurement - LUT1D</b>. You can define the corresponding settings for this functionality in the GeneralSettings file.</p>						
<b>WriteCombinedMeasurementFile.m</b>	<p>Use this feature to write an M file with measurement data. This format can be imported to ASM Engine Testbench and ASM Optimizer. You are recommended to directly import the MeasurementData (*.md) file. This file can be used only in special cases, for example if the measurement is distributed over several measurement data files. In this case, all referenced measurement data must have the same number of operating points. To use this file, you have to add the following fields to the settings file:</p> <table border="1"> <thead> <tr> <th>Settings field</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>Settings.PostFcns.WriteMeasurement.FileFullPath.v</code></td><td>String with the path to the generated file.</td></tr> <tr> <td><code>Settings.PostFcns.WriteMeasurement.MeasurementTypes.v</code></td><td>Cell array of strings with the MeasurementType names included</td></tr> </tbody> </table>	Settings field	Description	<code>Settings.PostFcns.WriteMeasurement.FileFullPath.v</code>	String with the path to the generated file.	<code>Settings.PostFcns.WriteMeasurement.MeasurementTypes.v</code>	Cell array of strings with the MeasurementType names included
Settings field	Description						
<code>Settings.PostFcns.WriteMeasurement.FileFullPath.v</code>	String with the path to the generated file.						
<code>Settings.PostFcns.WriteMeasurement.MeasurementTypes.v</code>	Cell array of strings with the MeasurementType names included						

---

## Related topics

### Basics

[Managing Measurement, Setting, Function, and Additional Function Files \(ModelDesk Processing\)](#)

## Measurement Functions

<b>Purpose</b>	To consider measurement data that is not contained in the original measurement data file.
----------------	---

<b>Description</b>	In some cases, it might be necessary to consider measurement data that is not contained in the original measurement data file. To calculate additional data that is treated like measurement data, you must add a new variable to your measurement type and connect it with the corresponding function in the Measurement Functions section on the Processing Configuration page.
--------------------	---

<b>Example</b>	The example shows a function that calculates the relative air mass for each operating point. The result can be used as measurement data.
----------------	--

**Function name and input arguments** Input arguments have to be given in the form: `Settings, MeasType, MeasVar, varargin`.

`varargin` can be used if additional parameters or information is needed. See the following example:

```
function [OutMdlStruct] = RelAirmass(Settings, MeasType, MeasVar, varargin)
```

The following table shows the specified input arguments:

Argument	Description
<code>Settings</code>	Structure defined in the general settings
<code>MeasType</code>	Used measurement type
<code>MeasVar</code>	Name of variable defined in measurement type

**Calculation of values** For the actual calculation you can use both parameters and other measurement data.

**Output argument** The structure of the output argument is similar to the imported measurement data, i.e. with `v` and `vUnit`.

Structure field	Description
<code>v</code>	Values of calculated measurement in each operating point
<code>vUnit</code>	Unit of calculated measurement

### Related topics

#### Basics

[Managing Measurement, Setting, Function, and Additional Function Files](#)  
(ModelDesk Processing 

## Initialization Files

<b>Purpose</b>	To manipulate specific parameters during start-up.
----------------	--

<b>Changes.m</b>	In the parameterization process a <b>changes.m</b> file is no longer generated. This file was used to manipulate specific parameters during system start-up. An example of what the (empty) file looked like is shown below.
------------------	--

However, you can still use a **changes.m** file by copying it next to VariantIni in the ...\\Simulation\\\_<variantname>\\IniFiles folder. Then it is called automatically.

```
function changes(varargin)
% function CHANGES(varargin)
% =====
%
% Parameter <config> must be one of the following
% 'PC' : Simulation on single PC (Offline, non Real-Time)
% 'HIL' : Simulation on Real-Time-Hardware, with ECU
% 'CPT' : Simulation in Real-Time-Hardware, without I/O connected
%
%
% This file contains the changes of the parameters made by the user(s).
% The changes should have an intermediate status during the development
% and will be transferred into the files MDL_INI, IO_INI and CPT_INI
% later by the corresponding developers at dSPACE.
% Please make sure that all information for this reintegration of the
% data is available in this file.
%
% AUTHOR(S): dSPACE GmbH
%
% VERSION : 1.0
%
% MODIFICATION(S):
% -----
%
% NAME: | DATE: | REMARKS:
% -----|-----|-----
%
% | | |
% | | |
% | | |
%
DefaultSimmode = 'PC';
```

```
% Check input parameter
if ( nargin==0)
    fprintf('\n');
    warning('asm:undefinedSimmode','No simmode defined; use default mode: %s',DefaultSimmode);
    config = DefaultSimmode;
else
    config = varargin{2};
end
if ~any(strcmp(config,{ 'PC', 'HIL', 'CPT'}))
    fprintf('\n');
    error('asm:undefinedSimmode','Wrong simmode defined: %s. ',config);
else
    % Get structures from workspace
    CPT = evalin('base','CPT;','[];');
    IO = evalin('base','IO;','[];');
    MDL = evalin('base','MDL;','[];');
    SIM = evalin('base','SIM;','[];');

    % =====
    % =====      PLACE YOUR CODE HERE !!!!      =====
    % =====
    % Write structures back to workspace
    assignin('base','CPT',CPT);
    assignin('base','IO',IO);
    assignin('base','MDL',MDL);
    assignin('base','SIM',SIM);

end;
```

In this file every parameter can be overwritten. Note that consistency checks are not performed when you use a `changes.m` file.

Adding an adapted parameter could look like this:

#### Note

When ASM tables or custom library parameter tables are changed, the table dimension has to be preserved. Otherwise ModelDesk parameter download on platforms will fail for these parameters.

```
% =====
% =====      PLACE YOUR CODE HERE !!!!      =====
% =====

MDL.EngineGasoline.FuelSystem.Rail.Map_q_RailLeak.Author = 'dSPACE';
MDL.EngineGasoline.FuelSystem.Rail.Map_q_RailLeak.Origin = 'Manipulation of rail leakage for small engine speeds';
MDL.EngineGasoline.FuelSystem.Rail.Map_q_RailLeak.Version = '1.0';
MDL.EngineGasoline.FuelSystem.Rail.Map_q_RailLeak.LastModified = '28.11.2011';
MDL.EngineGasoline.FuelSystem.Rail.Map_q_RailLeak.Comment = 'Rail leakage map, [mm3|s] = f(p_Rail, n_Engine)';
MDL.EngineGasoline.FuelSystem.Rail.Map_q_RailLeak.xName = 'p_Rail';
MDL.EngineGasoline.FuelSystem.Rail.Map_q_RailLeak.xUnit = '[bar]';
MDL.EngineGasoline.FuelSystem.Rail.Map_q_RailLeak.yName = 'n_Engine';
MDL.EngineGasoline.FuelSystem.Rail.Map_q_RailLeak.yUnit = '[rpm]';
MDL.EngineGasoline.FuelSystem.Rail.Map_q_RailLeak.vName = 'q_RailLeak';
MDL.EngineGasoline.FuelSystem.Rail.Map_q_RailLeak.vUnit = '[mm3|s]';
table = [
    0       1       135      200
    0       0       0         0
    600     0       0         0
    601     1       1         50
    7000    1       1         50
];
MDL.EngineGasoline.FuelSystem.Rail.Map_q_RailLeak.x = table(1,2:end)';
MDL.EngineGasoline.FuelSystem.Rail.Map_q_RailLeak.y = table(2:end,1);
MDL.EngineGasoline.FuelSystem.Rail.Map_q_RailLeak.v = table(2:end,2:end)';
clear table;
```

**Related topics****Basics**

[Managing Measurement, Setting, Function, and Additional Function Files  
\(ModelDesk Processing\)](#)



# ModelDesk Utility Library

---

## Where to go from here

## Information in this section

[Introduction to the ModelDesk Utility Library](#)..... 258

Provides an introduction to the ModelDesk Utility Library.

[SignalInterface](#)..... 259

These blocks are used in the Simulink model to enable exchanging data with ModelDesk.

[Custom Component](#)..... 263

These blocks are used to identify and migrate custom components.

# Introduction to the ModelDesk Utility Library

## Introduction to the ModelDesk Utility Library

### Introduction

The ModelDesk Utility Library (MDLD\_Utils.Lib) serves as a basis for interface functionalities between ModelDesk and ASM models. The following description provides information on the blocks contained in this library.

For more information on the ModelDesk interface, refer to [ModelDesk Interface](#) on page 160.

### Related topics

#### Basics

[ModelDesk Interface.....](#) 160

# SignallInterface

## Where to go from here

## Information in this section

[ASMSignallInterface](#)..... 259

The ASMSignallInterface block provides signals from the simulation model.

[ASMSignallInterface Manager](#)..... 261

The ASMSignallInterface Manager allows you to act globally on the multiple ASMSignallInterface blocks.

# ASMSignallInterface

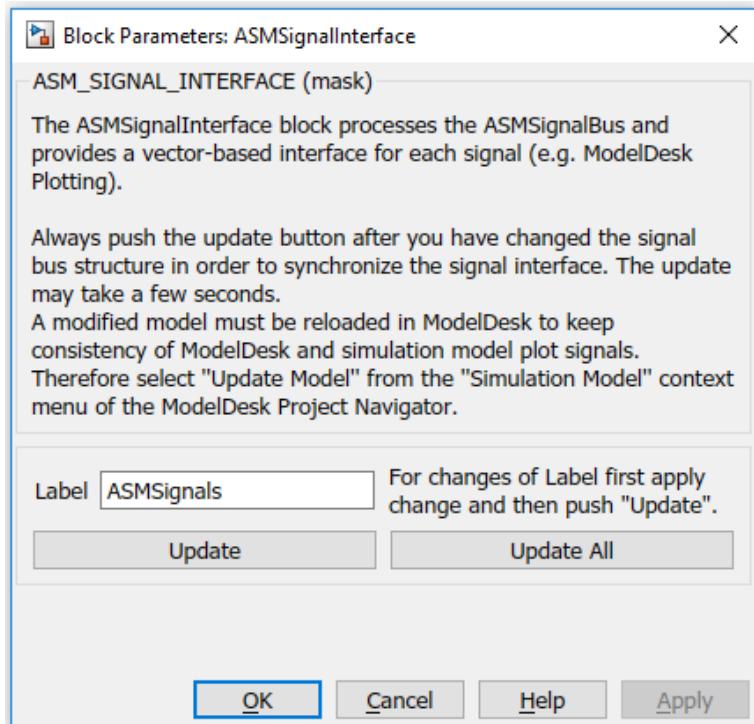
## Introduction

The ASMSignallInterface block provides signals from the simulation model. Every ASM module contains one ASMSignallInterface block. Inside the ASM module, the signals are provided by the ASMSignalBus. For signal visualization, you can use the Plot Manager in ModelDesk.



**Settings**

In the Block Parameters dialog, you can make the following settings:



**Label** You can rename the label that is shown as block annotation. The label conforms to the tab name of the Signal Selector in ModelDesk's Plot Manager, where you can find the signals provided by the ASMSignalInterface block.

**Update** Click Update after you changed the signal bus structure.

**Update All** Click Update All to update all ASMSignalInterface blocks in your model. The update can take a few seconds.

When you change a model, you have to update it also in ModelDesk to keep the consistency of ModelDesk and simulation model plot signals. For this, right-click the simulation model in the ModelDesk Project Navigator and select Update Model.

**Imports**

The following table shows the imports:

Name	Unit	Description
ASMSignalBus	[]	Signal bus containing signals of ASM components, refer to <a href="#">ASMSignalBus</a> on page 31.

---

**Related topics****Basics**

[Basics of the Plot Manager \(ModelDesk Plotting\)](#)

**HowTos**

[How to Update a Model in ModelDesk \(ModelDesk Parameterizing\)](#)

**References**

[Environment Configuration Dialog \(ModelDesk Scenario Creation\)](#)

## ASMSignallInterface Manager

---

**Introduction**

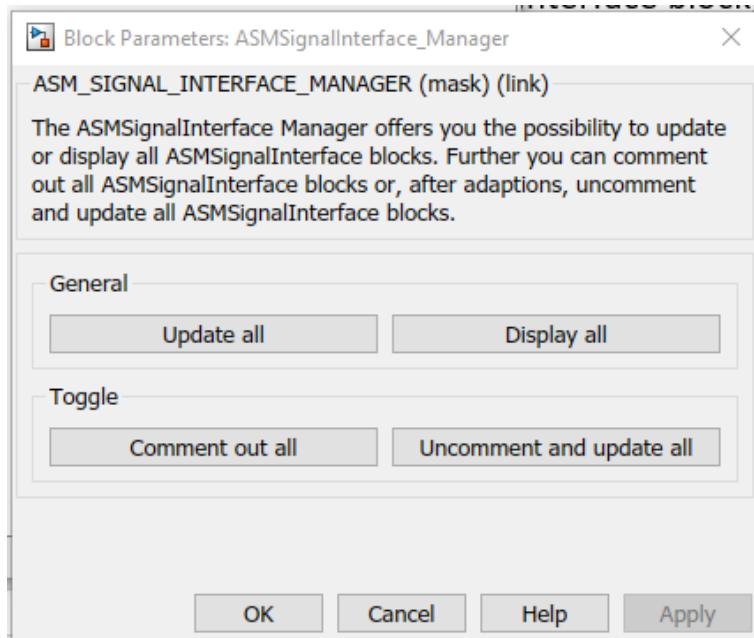
The ASMSignallInterface Manager allows you to act globally on multiple ASMSignallInterface blocks. It offers you the possibility to update or display all ASMSignallInterface blocks.

If no visualization of the Simulink simulation in ModelDesk is needed, you can comment out the ASMSignallInterface blocks to tighten the model structure. If it is needed again, you can uncomment and update them.



## Settings

In the Block Parameters, you can make the following settings:



**Update all** To update all ASMSignalInterface blocks in the model.

**Display all** To display a list of all ASMSignalInterface blocks in the model in the MATLAB Command Window.

**Comment out all** To comment out all ASMSignalInterface blocks in the model.

**Uncomment and update all** To uncomment all ASMSignalInterface blocks in the model and update them.

---

## Related topics

### Basics

[Basics of the Plot Manager \(ModelDesk Plotting\)](#)

# Custom Component

## Where to go from here

## Information in this section

[ModelDesk Custom Component.....](#) 263

To identify a *custom component* within ModelDesk.

[Custom Component Migration.....](#) 264

To migrate blocks of a custom library in the model.

## ModelDesk Custom Component

### Purpose

To identify a *custom component* within ModelDesk.

### Basics

The ModelDesk Custom Component block is used to identify a *custom component* within ModelDesk. The block is placed below the mask of the custom block, which needs to be registered.



For more information on handling custom components within Simulink and ModelDesk, refer to [Integrating and Using Custom Libraries \(ModelDesk Parameterizing\)](#).

### Related topics

#### Basics

[Integrating and Using Custom Libraries \(ModelDesk Parameterizing\)](#)

#### HowTos

[How to Create a Custom Library \(ModelDesk Parameterizing\)](#)

## Custom Component Migration

---

<b>Purpose</b>	To migrate blocks of a custom library in the model.
----------------	---

<b>Basics</b>	The Custom Component Migration block opens an overview user interface for all the custom component blocks that are not migrated and included in the active Simulink system. The block triggers the migration function for these blocks.
---------------	---



For further information, refer to [How to Migrate Blocks of Custom Component Libraries](#) on page 75.

---

### Related topics

### References

ModelDesk Custom Component.....	263
---------------------------------	-----

# Tutorial

## Introduction

In this tutorial, you will get first information on working with the ASM models.

## Where to go from here

### Information in this section

<a href="#">ASM Tutorial Videos.....</a>	266
There are different ASM tutorial videos.	
<a href="#">How to Open an ASM Library.....</a>	266
After successfully installing ASM on your PC, you can now easily open an ASM library.	
<a href="#">How to Start with an ASM Demo Model via MATLAB.....</a>	268
After successfully installing ASM on your PC, you can easily start an example model of your ASM installation via MATLAB.	
<a href="#">How to Start a Simulink Simulation.....</a>	270
Simulink simulation of the Automotive Simulation Models is possible in Simulink without dSPACE real-time hardware and RTI/Simulink® Coder™.	
<a href="#">How to Prepare a Model for Real-Time Simulation.....</a>	270
Real-time simulation of the ASM is possible with dSPACE real-time hardware and RTI/Simulink® Coder™.	
<a href="#">How to Generate Code for an ASM.....</a>	272
If you have a dSPACE RTI (Real-Time Interface) license, you can generate new code for an ASM.	
<a href="#">Specific Tutorials.....</a>	273
ASM contains different blocksets. Each blockset comes with model description that contains a tutorial for your first steps.	

## ASM Tutorial Videos

---

### Introduction

There are ASM tutorial videos on the dSPACE website. The videos describe the first steps with ASM.

To access the videos, go to

<https://www.dspace.com/en/pub/home/medien/videos.cfm>.

#### Note

The tutorial videos might show the handling in an earlier ASM version. If the handling in your ASM version is different, refer to the installed user documentation.

## How to Open an ASM Library

---

### Objective

After successfully installing ASM on your PC, you can now easily open an ASM library.

This lesson explains how to open the Gasoline Engine library. You can start all ASM products in a similar way. The MATLAB Command Window shows you all the products that are installed and covered by your license.

---

### Method

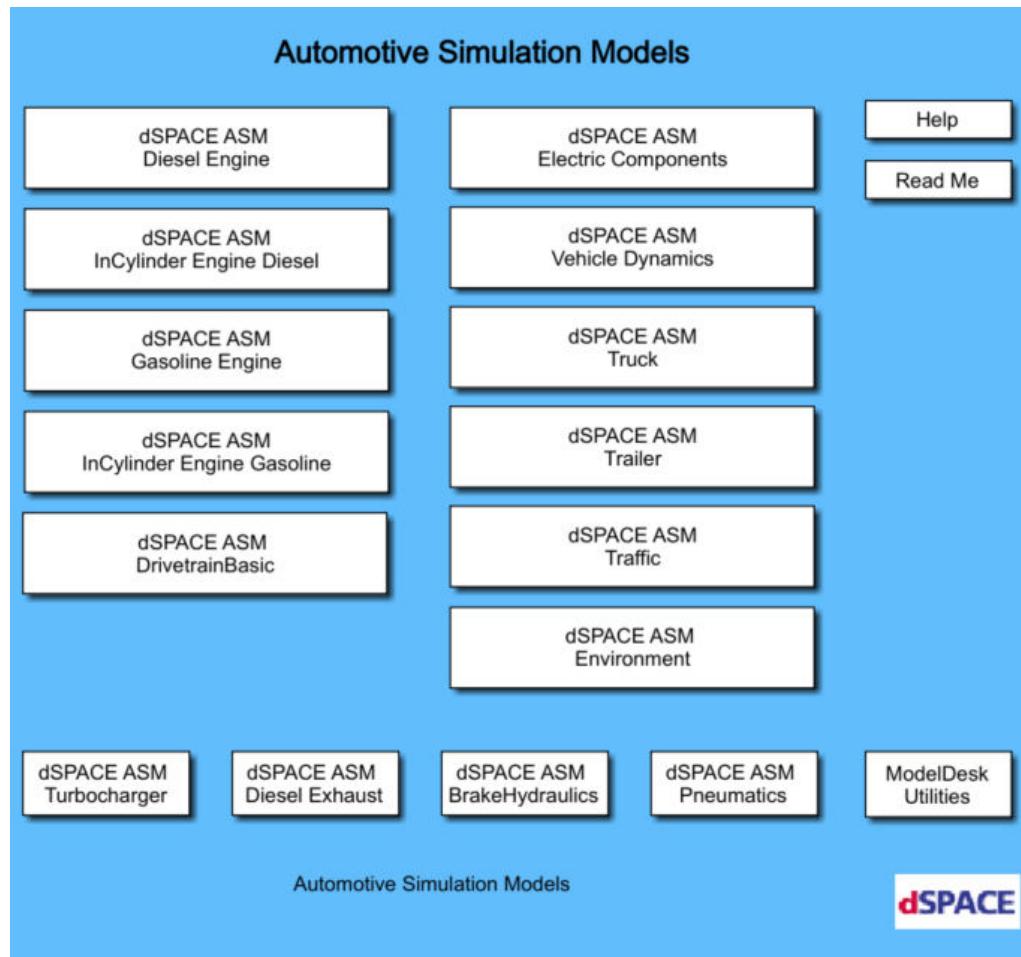
#### To open an ASM library

- 1 Start a MATLAB session.

When MATLAB starts, the MATLAB Command Window lists all the ASM blocksets that are installed on the PC.

**2** Type `asm` in your MATLAB Command Window.

The central ASM library opens.

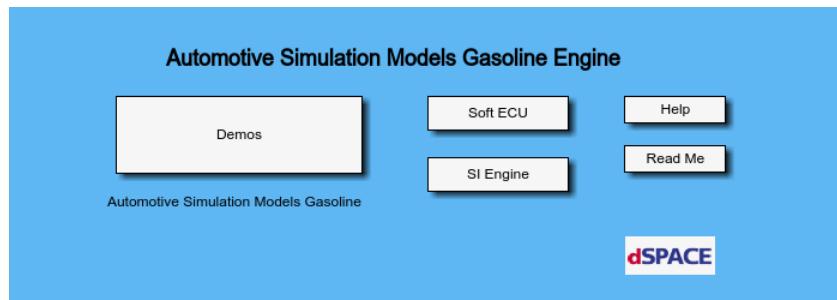


It provides an overview of all the accessible major ASM libraries:

- ASM Diesel Engine
- ASM InCylinder Engine Diesel
- ASM InCylinder Engine Gasoline
- ASM Gasoline Engine
- ASM Drivetrain Basic
- ASM Vehicle Dynamics
- ASM Electric Components
- ASM Trailer
- ASM Traffic
- ASM Truck
- ASM Environment

Depending on your software license, some of the libraries are grayed out. If a library is grayed, you do not have a valid license for it.

- 3 Double-click the dSPACE ASM Gasoline Engine subsystem.
- The subsystem opens. Depending on the installed license, you can open the developer or operator library. For details, refer to [Operator Version](#) on page 61.
- 4 Double-click the Developer or Operator subsystem to open the library.




---

## Result

The library opens. It is subdivided into three sections.

- On the left is a subsystem with all the demo models belonging to the library.
- The middle section contains the subsystems that are part of the library and which are contained in the example model as links.
- On the right are links to the documentation and to the last-minute information in the readme section.

---

## Related topics

### HowTos

How to Start with an ASM Demo Model via MATLAB.....	268
---	-----

## How to Start with an ASM Demo Model via MATLAB

---

### Objective

After successfully installing ASM on your PC, you can now easily start a demo model of your ASM installation via MATLAB.

This lesson explains how to start the Gasoline Engine model via MATLAB. You can start all ASM products in a similar way. The MATLAB Command Window shows you all the products that are installed and covered by your license.

---

### Methods to start with an ASM demo model

You can start an ASM demo model with the following methods:

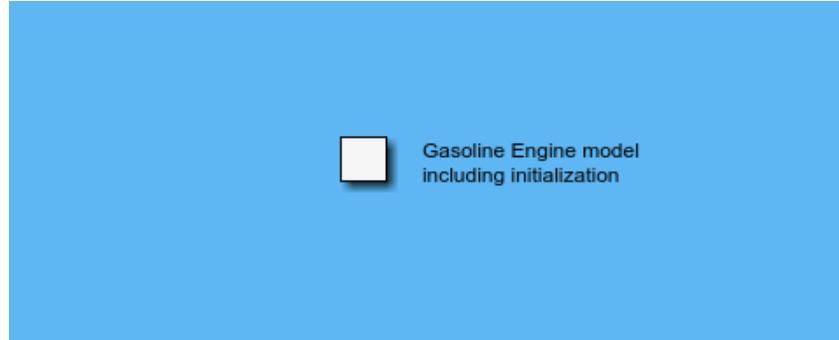
- Start an ASM demo model via MATLAB, see below.
- Generate an ASM project by using ModelDesk or by using MATLAB/Simulink. Refer to [How to Create a Project Based on an ASM Demo \(ModelDesk Project and Experiment Management\)](#).

**Preconditions**

The ASM library is open. Refer to [How to Open an ASM Library](#) on page 266.

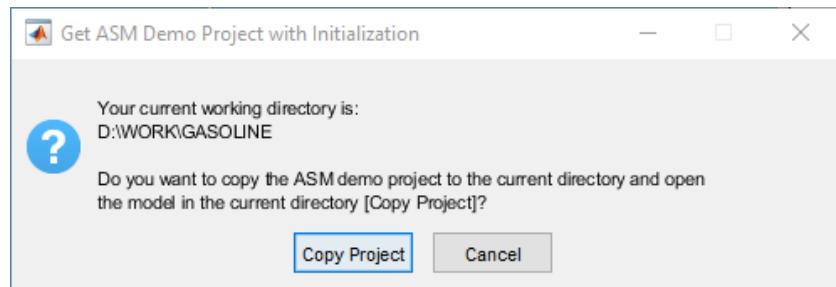
**Method****To start with an ASM demo model via MATLAB**

- 1 Double-click the Demos subsystem to open a new window which presents all the demo models that are part of the library.



- 2 Double-click the demo model to start it.

This opens a dialog that offers you the possibility to copy the project to your current working directory.



- 3 Click Copy Project to copy the current demo folder to your current MATLAB work folder.

All the files related to the demo model are copied to the current MATLAB work folder. This includes the entire parameterization and the ControlDesk example layouts.

**Result**

Now the demo model opens. All the initialization files are executed and you can start simulation.

**Related topics****HowTos**

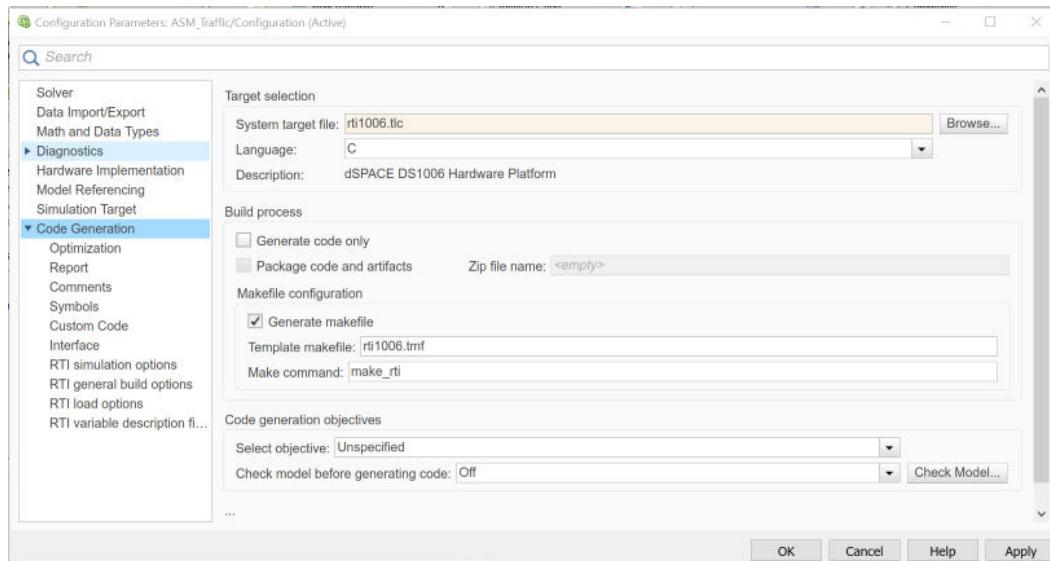
[How to Create a Project Based on an ASM Demo \(ModelDesk Project and Experiment Management\)](#)

## How to Start a Simulink Simulation

<b>Objective</b>	Simulink simulation of the Automotive Simulation Models is possible in Simulink without dSPACE real-time hardware and RTI/Simulink® Coder™.
<b>Simulink accelerator mode</b>	You can simulate the model in the Simulink accelerator mode. This can be set via a list to the right of the Start Simulation button. For details, refer to the Simulink documentation.
<b>Method</b>	<b>To start a Simulink simulation</b> <ol style="list-style-type: none"><li>1 For Simulink simulation, click the Start Simulation on the toolbar of the Simulink model. All the simulation parameters are set during the initialization procedure.</li></ol>

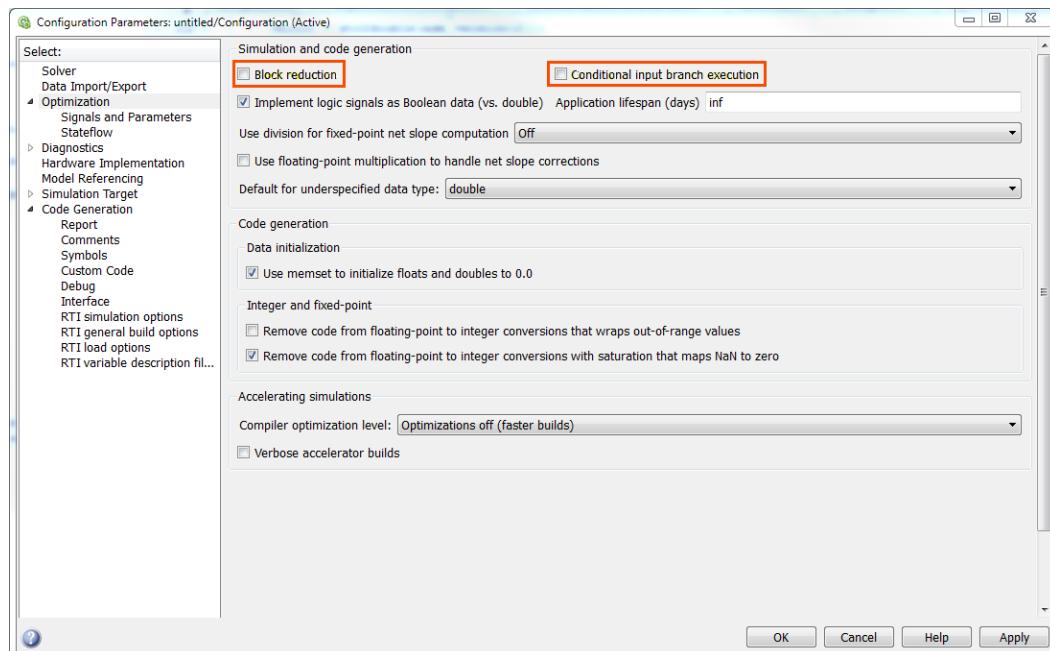
## How to Prepare a Model for Real-Time Simulation

<b>Objective</b>	Real-time simulation of the ASM is possible with dSPACE real-time hardware and RTI/Simulink® Coder™, software products from dSPACE and MathWorks®. To use your model for real-time simulation, you must prepare it.
<b>Method</b>	<b>To prepare a model for real-time simulation</b> <ol style="list-style-type: none"><li>1 All ASM example models are configured for generation of real-time code. Simply press <b>Ctrl+B</b>. The code generation process starts. When it has finished, you find a set of real-time application files in your current working folder. You can use these files for real-time simulation in combination with a dSPACE real-time hardware and ControlDesk.</li><li>2 The following steps are necessary only if you have changed any Simulink simulation parameters during your work. From the menu bar, choose <b>Simulation – Model Configuration Parameters</b> or press <b>Ctrl+E</b> to open the Configuration Parameters dialog. Open the <b>Code Generation</b> page and check if the settings correspond to the following illustration.</li></ol>

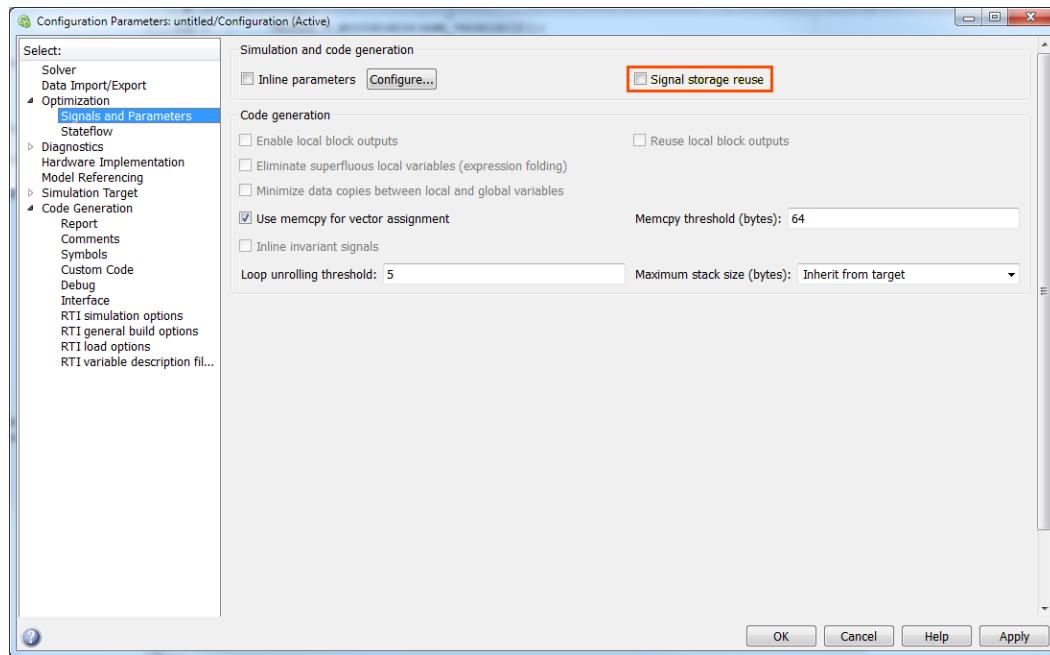


Parameter	Description
System target file	The system target file must fit the real-time processor you want to generate code for. A list of targets is accessible via the Browse button.
Template makefile	The template makefile is set automatically if you choose a system target file.
Make command	For dSPACE real-time hardware, the make command must be <code>make_rti</code> .

- 3 Open the Optimization page. Clear Block reduction and Conditional input branch execution.



**4** On the Signals and Parameters page, clear Signal storage reuse.

**Result**

You can use the ASM model for real-time simulation.

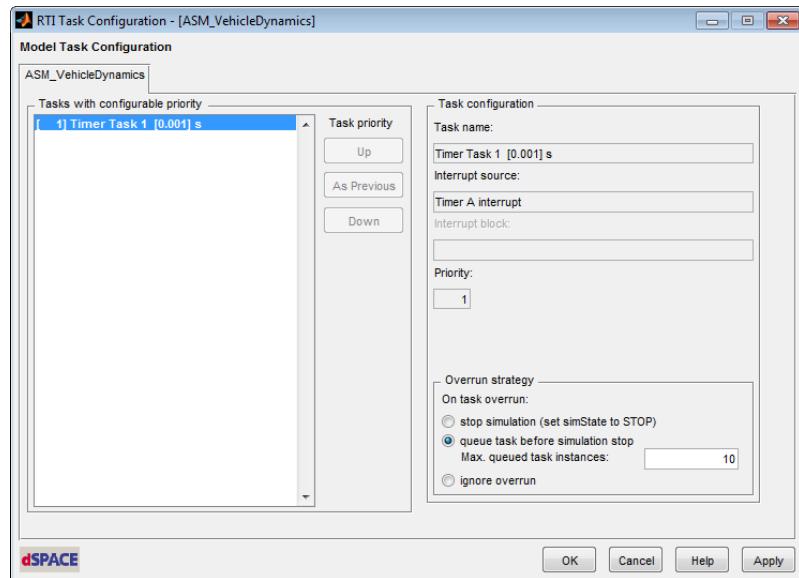
## How to Generate Code for an ASM

**Objective**

If you have a dSPACE RTI (Real-Time Interface) license, you can generate new code for the ASM. In some combinations of models and real-time systems it is necessary to modify the default task configuration.

**Method****To generate code for an ASM**

- 1 Start MATLAB, switch to <Projectfolder>/Simulation and call **go**. If MATLAB asks, select the automatic setting of the preferences. The demo model opens with the initial default parameters specified in the go file.
- 2 From the menu bar, choose **Simulation – Model Configuration Parameters** or press **Ctrl+E** to open the Configuration Parameters dialog.
- 3 Open the RTI Simulation Options page and click Task Configuration. In the Task Configuration dialog, set the overrun strategy of timer task 1 to queue task before simulation stop and the number of maximum task instances to 10. This setting has to be made, because the initialization of the model on the real-time hardware takes some time, so overruns will occur at the beginning of the simulation. A maximum of 10 permitted overruns will prevent the simulation from stopping during initialization.



- 4 To start code generation, click Build Model on the toolbar or press **Ctrl+B**.

## Result

The complete build process takes some minutes. After the build process is finished, the new real-time application is downloaded to the dSPACE platform. The initial parameters are the parameters which were loaded from the go file.

# Specific Tutorials

## Introduction

ASM contains different blocksets. Each blockset comes with model description that contains a tutorial for your first steps.

## Tutorials

The following list gives an overview of all the available tutorials for ASM blocksets. The tutorials are contained in the model description for the specific blocksets. Refer to the tutorial for your licensed blockset.

- Tutorial for an ASM Electric Components model (refer to [Tutorials \(ASM Electric Components Model Description\)](#))
- Tutorial for an ASM Gasoline Engine model (refer to [Tutorials \(ASM Gasoline Engine Model Description\)](#))
- Tutorial for an ASM Gasoline Engine InCylinder model (refer to [Tutorials \(ASM Gasoline Engine InCylinder Model Description\)](#))
- Tutorial for an ASM Diesel Engine model (refer to [Tutorials \(ASM Diesel Engine Model Description\)](#))
- Tutorial for an ASM Diesel Engine InCylinder model (refer to [Tutorials \(ASM Diesel Engine InCylinder Model Description\)](#))

- Tutorial for an ASM Drivetrain Basic model (refer to [Tutorials \(ASM Drivetrain Basic Model Description\)](#))
- Tutorial for an ASM Vehicle Dynamics model (refer to [Tutorials \(ASM Vehicle Dynamics Model Description\)](#))
- Tutorial for an ASM Trailer model (refer to [Tutorials \(ASM Trailer Model Description\)](#))

# Troubleshooting

## Where to go from here

## Information in this section

<a href="#">Troubleshooting with Rising Turnaround Times at Steady Operating Points.....</a>	276
Simulating the model on a real-time processor board at a steady state might significantly increase turnaround time after several seconds of simulating the same operating point. When another operating point is begun, the turnaround time decreases to the original lower value.	
<a href="#">Troubleshooting in MATLAB/Simulink.....</a>	277
Problems can occur when working with ASM and MATLAB/Simulink.	
<a href="#">Troubleshooting after ControlDesk Project Migration.....</a>	279
Variable connections become invalid because discontinued Simulink blocks within the ASM library blocks have been updated to a new standard Simulink block.	
<a href="#">Troubleshooting with Measurement Function Definitions in the General Settings.....</a>	279
ModelDesk's measurement functions feature lets you create pseudo measurements based on measured variables and parameters. These new variables are calculated with MATLAB scripts within ModelDesk Processing.	
<a href="#">Troubleshooting with Parameters in ModelDesk Processing After Migrating ModelDesk Projects.....</a>	284
After the migration of an ASM project, it is possible that a parameter exists with a different address when ModelDesk Processing is executed.	
<a href="#">Troubleshooting with Dimension Parameters Error Messages.....</a>	285
When parameter sets are downloaded to a platform, differences between dimension parameters can trigger error messages.	
<a href="#">Troubleshooting for Task Overruns with a MotionDesk/Animation Interface on SCALEXIO Platforms.....</a>	287
In ASM demo models, the execution of the ASM model and the execution of the MotionDesk Interface or Animation Interface can	

lead to high calculation load of the timer task or even to task overruns on a SCALEXIO platform.

[Troubleshooting for Task Overruns with a MotionDesk/Animation Interface on PHS Bus-based Platforms.....](#) 289

In ASM demo models, the execution of the ASM model and the execution of the MotionDesk Interface or *Animation Interface* can lead to high calculation load of the timer task or even to task overruns when simulating on a PHS bus-based platform.

## Troubleshooting with Rising Turnaround Times at Steady Operating Points

### Introduction

Simulating the model on a real-time processor board at a steady state might significantly increase turnaround time after several seconds of simulating the same operating point. When another operating point is begun, the turnaround time decreases to the original lower value. For example, with an engine model, this behavior might occur when you drag the engine to test bench mode at a fixed operating point.

### Basic information

The increase of the turnaround time is related to a floating-point underflow (or gradual underflow). The resulting floating-point numbers are called denormals. Calculations with denormals are much slower.

For example, the following use cases might lead to denormals in the model.

**First-order delays** Several first-order delays are connected to the output of look-up tables: for example, for an engine model, the output pressure ( $p_{Out\_Comp}$ ) of the map-based turbocharger. During steady state, the input to these tables and therefore also the output are constant. The output of the first-order delay converges asymptotically to the input. After a certain time the difference between input and output divided by the time constant becomes so small that it is handled as denormal. In most cases the output is not zero, so that, in the first-order delay, denormals are only used internally. This typically does not significantly affect the turnaround time.

**Flushing of chambers** In an engine model simulation, changing from an operating point with injection to an operating point without injection, the concentration of fuel and exhaust in the manifold and the cylinders slowly decreases. The engine is flushed with air. After some time, the masses and mass fractions for fuel and exhaust become so small that the calculation requires denormals. The denormals are propagated in the model for related calculations (for example, mass flow over the valves or estimation of thermodynamic variables), which must also handle these denormals. This typically significantly affects the turnaround time.

**Workaround**

The calculation with denormals can be suppressed by flushing them to zero. This leads to a minor loss of precision, but that does not significantly affect the precision of the entire model.

**PHS-bus-based systems** On PHS-bus-based systems (e.g., DS1006), flushing to zero can be activated during the compile process by adding the following block of the ASM\_Utils library to the tasks concerned:

ASM\_Utils.lib/Miscellaneous/FlushDenormals2Zero

**SCALEXIO** On SCALEXIO systems, flushing to zero can be activated by defining the `DS_ENABLE_DENORMALS_ZERO_HANDLING` macro in the related build configuration.

Perform these steps:

1. Open the table build configuration.
2. Select the build configuration set.
3. In the property grid, add `DS_ENABLE_DENORMALS_ZERO_HANDLING` to the preprocessor macros to be defined.

**Related topics****Basics**

[Building Real-Time Applications \(ConfigurationDesk Real-Time Implementation Guide\)](#)

## Troubleshooting in MATLAB/Simulink

**Introduction**

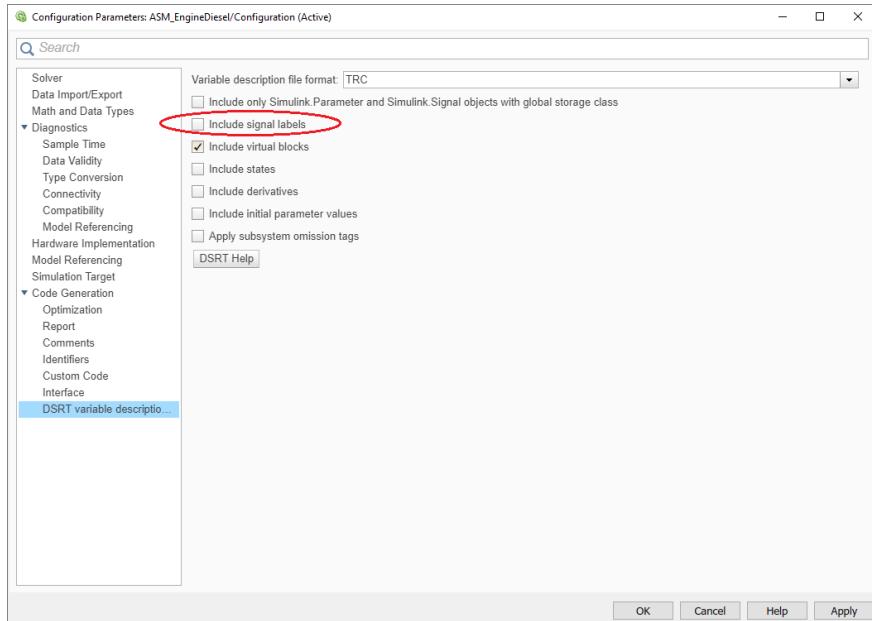
Problems can occur when working with ASM and MATLAB/Simulink.

**Out of memory**

When the real-time application is built, an Out-of-Memory error at MATLAB might occur, for example: `TLC Compiler encountered an OUT_OF_MEMORY`

**condition.** This error message results from an insufficient memory allocation of the MATLAB session. To avoid this error, one of the following actions can help:

- In the Configuration Parameters dialog, open the DSRT variable description file options page and disable the **Include signal labels** option.



- In some cases, it is a temporary solution to close MATLAB, and continue with a new MATLAB session.

#### Problems with restoring settings of the Simulink Scope block (starting from MATLAB Release R2015b)

Together with MATLAB Release R2015b, a new Scope Simulink block has been introduced. The Scope block contains new setting options compared to the former Scope block. However, using the `asm_replaceblocks` ASM function (which is triggered by the corresponding buttons in the ASM MDLUserInterface subsystems, for example) will only restore the following scope properties:

- 'Location'
- 'Open'
- 'NumInputPorts'
- 'TickLabels'
- 'ZoomMode'
- 'AxesTitles'
- 'Grid'
- 'TimeRange'
- 'YMin'
- 'YMax'
- 'SaveToWorkspace'
- 'SaveName'
- 'DataFormat'
- 'LimitDataPoints'

- 'MaxDataPoints'
- 'Decimation'
- 'SampleInput'
- 'SampleTime'

Other options will be restored to default value if a scope will be disabled and restored with this function.

#### Related topics

#### Basics

[Adapting the Generation of the Variable Description File \(Model Interface Package for Simulink - Modeling Guide\)](#)

## Troubleshooting after ControlDesk Project Migration

### Invalid look-up table data connections after ControlDesk project migration

In dSPACE Release 2016-B, the discontinued Simulink blocks Lookup and Lookup2D within the ASM library blocks have been updated to the new standard Simulink Lookup Table (n-D) block. This results in a change of the variable paths in ControlDesk from OutputValues to LookUpTableData. During migration of ControlDesk projects from Release 2016-A and earlier, the variable connections of the Table Editor become invalid.

## Troubleshooting with Measurement Function Definitions in the General Settings

### Introduction

ModelDesk's measurement functions feature lets you create pseudo measurements based on measured variables and parameters. These new variables are calculated with MATLAB scripts within ModelDesk Processing.

As of Release 2016-A, measurement functions in ModelDesk are part of the measurement data and can be accessed in the Processing Configuration pane. Before that, they were defined in the general settings file of the Processing Configuration of ModelDesk (`Settings.CalculatedMeasurement` field).

When measurement functions are defined in the measurement data, the results are stored in the measurement data and provided to later processing executions. Whereas measurement functions inside the general settings are always reexecuted for each processing execution. Therefore, it is recommended to use the measurement data to define these functions.

When measurement functions are defined in the general settings file, a warning occurs during the execution of ModelDesk Processing.

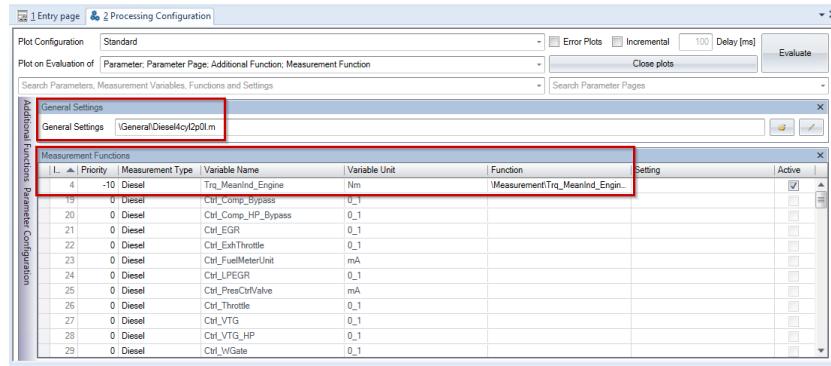
*The measurement function definitions inside the general settings have been replaced by definition of measurement functions in the measurement data.*

For more information on the general settings file, refer to [General Settings File](#) on page 247.

For more information on the Processing Configuration pane, refer to [Working with the Processing Configuration Pane \(ModelDesk Processing\)](#).

## Workaround

To eliminate the warning, you must transfer the definition of the measurement functions from the general settings file to the measurement data.



For more information on the Processing Configuration pane, refer to [Working with the Processing Configuration Pane \(ModelDesk Processing\)](#).

The following is an example of how to transfer the definition of the measurement function of the Trq\_MeanInd\_Engine variable of the ASM EngineDiesel project.

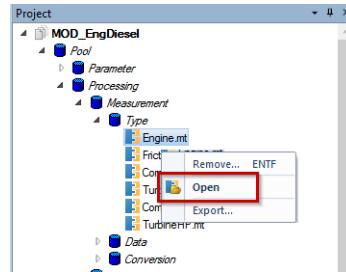
1. Open the active general settings file and identify the definition of the measurement function in the `Settings.CalculatedMeasurement` field.
  2. Identify the measurement function of the 'Trq\_MeanInd\_Engine' variable, including the priority of the function and the measurement type of the variable.
- Each processing function takes one row in the cell array. With the cell content **Priority**, **Measurement Type Name**, **Measurement Variable Name**, **Measurement function path**.

```
Settings.CalculatedMeasurement.v = { ...
    -10,'Engine','Trq_MeanInd_Engine',...
    fullfile(ProcPool,'Function','Measurement','Trq_MeanInd_Engine.m'));
```

**Note**

- `ProcPool` is the folder of the Processing pool in the ModelDesk project folder:  
`<ASMPProject>\Parameterization\MOD_EngDiesel\Pool\Processing`
- The folder of the function in this example is:  
`<ASMPProject>\Parameterization\MOD_EngDiesel\Pool\Processing\Function\Measurement`
- The *priority* of the function is the execution order of the processing functions. For more information on the execution order, refer to [How to Specify the Execution Sequence of Function Files \(ModelDesk Processing\)](#).
- `Engine` is the corresponding measurement type of the `Trq_MeanInd_Engine` variable'. For information on measurement types, refer to [Raw Data, Measurement Types, and Measurement Data \(ModelDesk Processing\)](#).

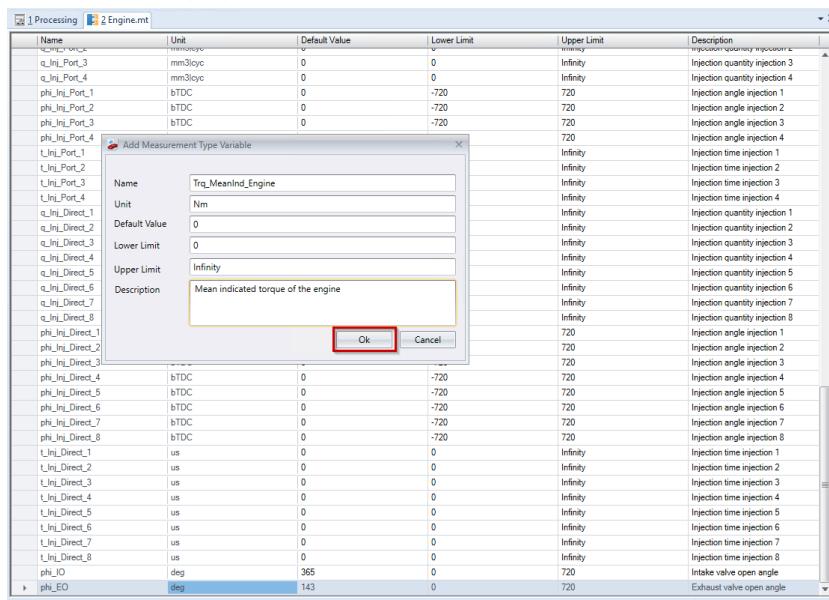
3. Extend the affected measurement type `Engine` by the missing '`Trq_MeanInd_Engine`' variable. The measurement type is a list of variables used in ModelDesk Processing functions for a measurement.
  1. In the Project Navigator, go to Pool - Processing - Measurement - Type and open the context menu of the corresponding measurement type. Click Open to open it from the ModelDesk Pool.



2. In the list of variables, open the context menu and click Append to add a new variable 'Trq\_MeanInd\_Engine' to the list.

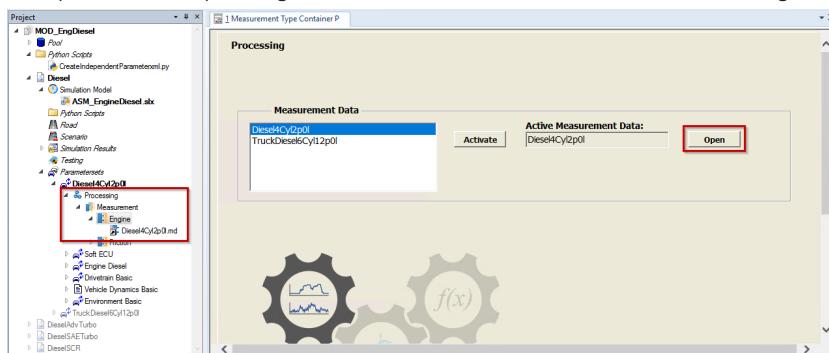
A dialog opens.

3. Define the properties of the variable and click Ok.

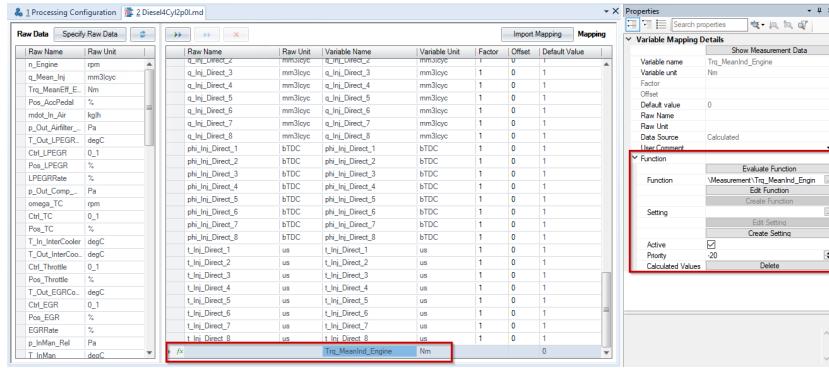


4. Save and close the measurement type.

4. Open the corresponding measurement data of the ModelDesk Processing.



5. Transfer the priority and location of the M-function from the general settings file to the Measurement Data.



6. Remove the definition of measurement functions from the general settings file.

#### Note

The general settings are evaluated each time ModelDesk Processing is executed. Thus, the measurement functions are automatically calculated for each new parameter set. After transferring the definition of the measurement functions from the general settings file to the measurement data, you must activate the measurement functions for each affected measurement data.

#### Note

If you use multiple parameter sets in a ModelDesk project and they are based on different measurements but use the same general settings file, activate the corresponding measurement functions for each measurement data. You have to do this for each measurement data that uses the general settings file from which the definition of the measurement functions was removed.

## Result

You transferred the definition of the measurement functions from the general settings file to the measurement data of ModelDesk.

---

<b>Related topics</b>	<b>Basics</b>
	<p>General Settings File..... 247 Preparing Measurement Data for Processing (ModelDesk Processing ) Raw Data, Measurement Types, and Measurement Data (ModelDesk Processing ) Working with the Processing Configuration Pane (ModelDesk Processing )</p>
	<b>HowTos</b>
	<p>How to Specify the Execution Sequence of Function Files (ModelDesk Processing )</p>

## Troubleshooting with Parameters in ModelDesk Processing After Migrating ModelDesk Projects

### Introduction

After the migration of an ASM project, it is possible that a parameter exists with a different address when ModelDesk Processing is executed. This happens if a parameter or component has been renamed or a former version for a block has been introduced, because this changes its address.

A warning of one of the following types is displayed:

- *For the requested subcomponent: 'ENGINE\_SETUP' only a former version match exists: 'engine\_setup\_4\_0'.*
- *The parameter `<old Maincomponent>.<old Subcomponent>.<old Name>` has been redirected to `<new Maincomponent>.<new Subcomponent>.<new Name>`.*
- *The existence request of `<old Maincomponent>.<old Subcomponent>.<old Name>` has been redirected to `<new Maincomponent>.<new Subcomponent>.<new Name>`.*

### Description

The address of a ModelDesk parameter consists of names of the main component (e.g., EngineDiesel or Environment), the subcomponent (ASM block), and the parameter. In this way, the parameter is precisely defined and you can access the parameter outside of ModelDesk.

For example, the address of the Const\_num\_Cyl parameter of the ENGINE\_SETUP block of the EngineDiesel component is:  
`EngineDiesel.ENGINE_SETUP.Const_num_Cyl`.

This way, MATLAB functions of ModelDesk Processing read parameters from different parameter pages and calculate other parameters. The following is an example of such a call:

```
Const_num_Cyl = asm_proc('get_parameter',...
    'MainComp','EngineDiesel','SubComp','ENGINE_SETUP','ParaName','Const_...
    num_Cyl','InstID',InstID);
```

Problems can occur when the address of a ModelDesk parameter is renamed with a former version of the ASM block. For more information on the migration of ASM models, refer to [Migrating ASM Models](#) on page 67.

The `asm_proc` function automatically redirects parameter requests to the related former version or according to an internally stored redirection list if a parameter is not found. In rare cases (e.g., further development on migrated models), this can lead to unexpected behavior. Therefore, it is recommended to update these parameter requests in the related processing function.

## Workaround

To avoid the warning, manually adjust the address of the ModelDesk parameter in the affected processing functions.

For example, change the following address:

```
Const_num_Cyl = asm_proc('get_parameter',...
    'MainComp','EngineDiesel','SubComp','ENGINE_SETUP','ParaName','Const_...
    num_Cyl','InstID',InstID);
```

Change it to:

```
Const_num_Cyl = asm_proc('get_parameter',...
    'MainComp','EngineDiesel','SubComp','engine_setup_4_0','ParaName','Co...
    nst_num_Cyl','InstID',InstID);
```

## Related topics

### Basics

[Migrating ASM Models](#).....67

# Troubleshooting with Dimension Parameters Error Messages

## Introduction

When parameter sets are downloaded to a platform, differences between dimension parameters can trigger error messages similar to the following one:

Error: The maximum number of cylinders has been changed in ModelDesk Subcomponent: 'ENGINE\_SETUP' ASM Instance ID: '1'. The current parameterized value is '8' whereas the compiled code expects a value of '6'.  
For Details refer to [ASM\\_UserGuide Troubleshooting](#).

For information on dimension parameters, refer to [Dimension Parameters](#) on page 57.

## Workaround

There are two possible workarounds for this issue:

**Maximum number parameters** For the *number of cylinders*, *number of direct injections*, *number of port injections*, and *number of ignitions* there are two parameters each.

- One for the maximum number (supported by the compiled code).
- One for the used number.

Check if the maximum number can be set to the maximum number of the compiled code running on the platform. This is typically the case if the used number is equal to or smaller than the maximum number of the compiled code. If possible, specify the maximum number and run the processing again to ensure that all mapping tables fit. If the generated code does not support the desired used number, new code has to be built.

### Note

To reduce the amount of code generation runs and handling of different code versions, define the maximum number of cylinders and injections at the beginning of a project and use them in all configurations and parameterizations.

**Number parameters** For the number parameters (e.g., the *number of intake manifolds*), the workaround is similar.

Estimate the required maximum value for the generated code. If the current real-time code does not support the desired used number, new code has to be built. If the dimension of the real-time code is greater than in the current parameter set, increase the value in the parameter set and check the related mapping matrixes. You can then use this code for all variants of number parameters.

### Note

Increasing the dimension parameter influences the turnaround time. Therefore, you have to specify the dimension parameter accordingly: as few as possible, as many as required.

---

## Related topics

### Basics

Dimension Parameters..... 57

# Troubleshooting for Task Overruns with a MotionDesk/Animation Interface on SCALEXIO Platforms

## Introduction

To connect the ASM model to MotionDesk, the ASM model can contain the MotionDesk Interface or the Animation Interface.

In ASM demo models, the execution of the ASM model and the execution of the MotionDesk Interface or Animation Interface are both assigned to the same timer task by default. This can lead to high calculation loads for the timer task or even task overruns when simulating the model on a SCALEXIO platform.

To avoid this, you can assign the execution of the MotionDesk Interface or Animation Interface to an additional task with a lower priority than the timer task.

### Note

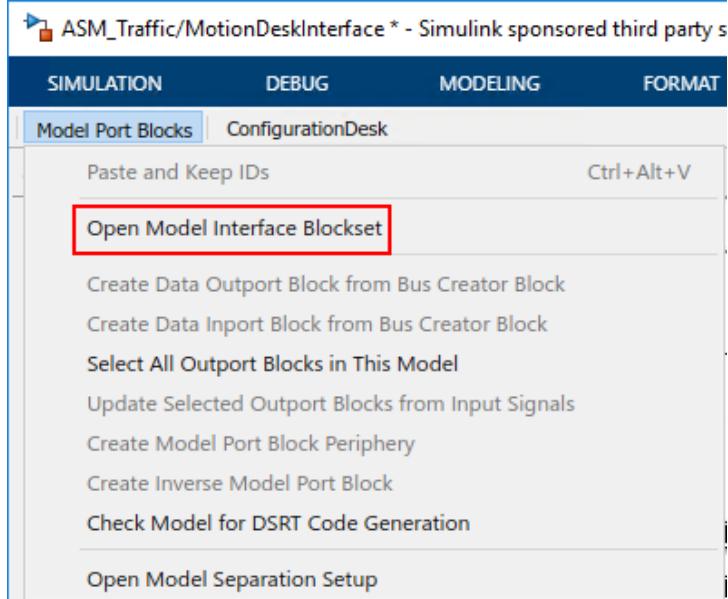
The following instructions describe the steps to assign the MotionDesk Interface to a low priority task. The description is also valid for the Animation Interface.

## Method

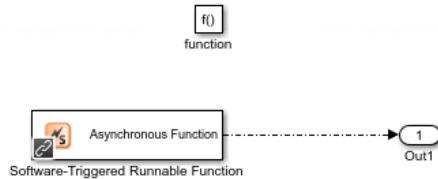
### To assign the MotionDesk Interface to a low priority task for SCALEXIO platforms

- 1 In MATLAB, switch to <Projectfolder>\Simulation and call `go`.  
The demo model opens with the initial default parameters specified in the `go` file.
- 2 Navigate to the MotionDeskInterface subsystem and open it.
- 3 Place a function-call subsystem from the Simulink library into the MotionDeskInterface subsystem.
- 4 Open the function-call subsystem and delete the import.

- 5 Open the Model Interface Blockset.



- 6 Insert a Software-Triggered Runnable Function block in the function-call subsystem. Connect the outport of the Software-Triggered Runnable Function block to the outport of the function-call subsystem.



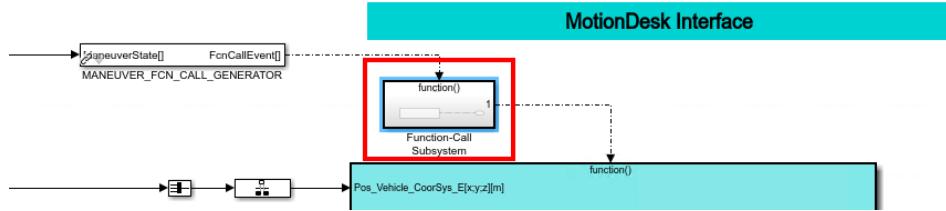
- 7 Open the block dialog of the Software-Triggered Runnable Function block.

Set the Required priority parameter to a value higher than the priority value for the timer task that triggers the ASM model execution.

**Tip**

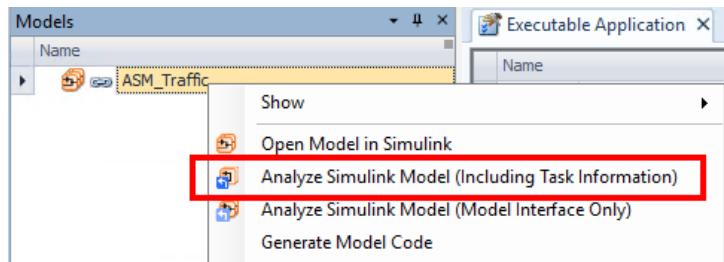
Typically, a *required priority* value of 50 is sufficient.

- 8** Place the function-call subsystem between the MANEUVER\_FCN\_CALL\_GENERATOR block and the triggered subsystem MOTION\_DESK\_INTERFACE.



Save the model.

- 9** Open the ConfigurationDesk project and execute the Analyze Simulink Model (Including Task Information) command.



- 10** Check the task configuration. Make sure that the priority value for the asynchronous function is greater than the priority value for the periodic task that triggers the ASM model execution.

- 11** Start the build process in ConfigurationDesk.

## Result

You have assigned the execution of the MotionDesk Interface or Animation Interface to a lower priority task to avoid task overruns.

# Troubleshooting for Task Overruns with a MotionDesk/Animation Interface on PHS Bus-based Platforms

## Introduction

To connect the ASM model to MotionDesk, the ASM model can contain the MotionDesk Interface or the Animation Interface.

In ASM demo models, the execution of the ASM model and the execution of the MotionDesk Interface or Animation Interface are both assigned to the same timer task by default. This can lead to high calculation loads for the timer task or even task overruns when simulating the model on a PHS bus-based platform.

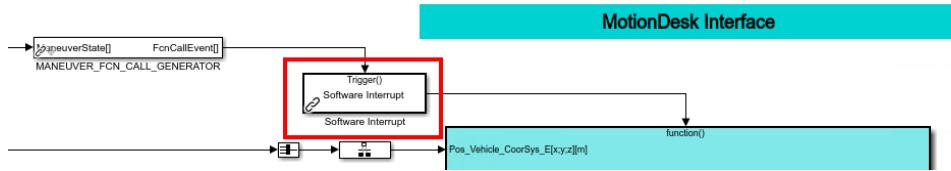
To avoid this, you can assign the execution of the MotionDesk Interface or Animation Interface to an additional task with a lower priority than the timer task.

**Note**

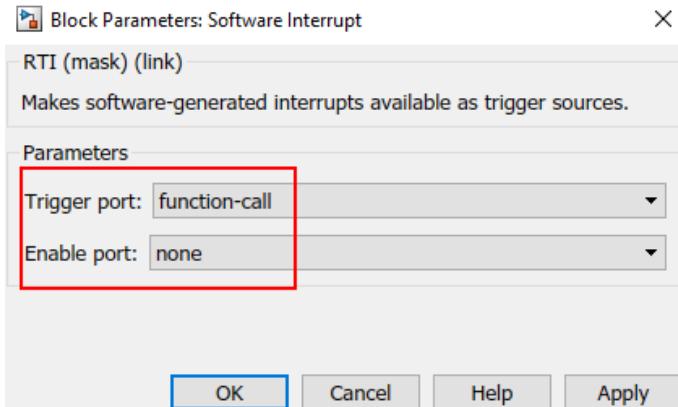
The following instructions describe the steps to assign the MotionDesk Interface to a low priority task. The description is also valid for the Animation Interface.

**Method****To assign the MotionDesk Interface to a low priority task for PHS bus-based platforms**

- 1 In MATLAB, switch to <Projectfolder>\Simulation and call go.  
The demo model opens with the initial default parameters specified in the go file.
- 2 Navigate to the MotionDeskInterface subsystem and open it.
- 3 In the MATLAB Command Window, type `rtitasklib` to open the RTI TaskLib.
- 4 From the library, drag a Software Interrupt block to the MotionDeskInterface subsystem.  
Place the block between the MANEUVER\_FCN\_CALL\_GENERATOR block and the triggered subsystem MOTION\_DESK\_INTERFACE.



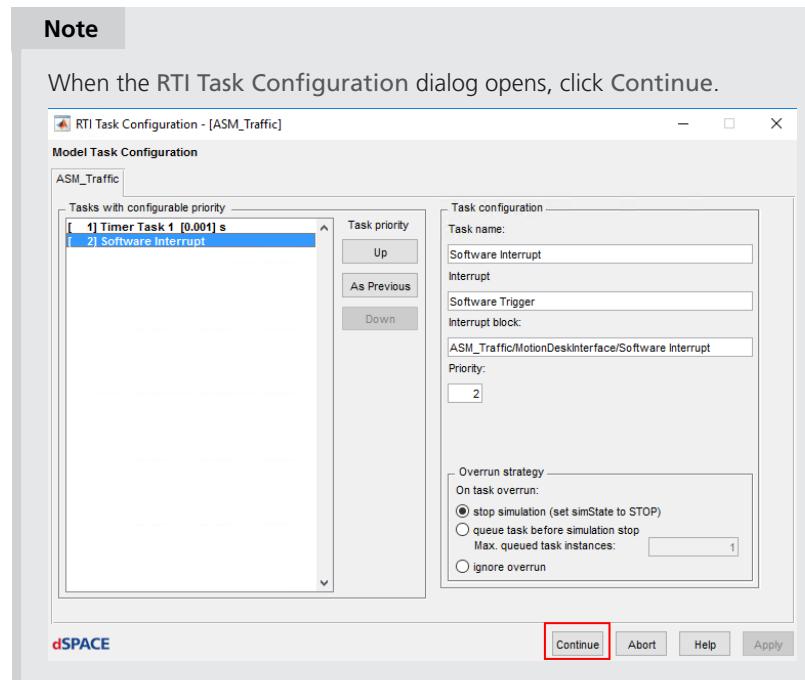
- 5 Double-click the Software Interrupt block. In the block dialog, make the following settings:



Click OK and save the model.

**6** Build the model.

RTI now automatically assigns a lower priority value (2) to the software interrupt than to timer task 1 (1) which gets the higher priority value.

**Result**

You have assigned the execution of the MotionDesk Interface or Animation Interface to a lower priority task to avoid task overruns.



# Limitations

## Limitations of All ASM Blocksets

<b>Introduction</b>	Some limitations apply to all the ASM blocksets.
<b>Rapid accelerator simulation mode</b>	ASM does not support the Simulink rapid accelerator simulation mode.
<b>Row major code generation</b>	ASM does not support the Simulink row major code generation. If a model contains ASM blocks and row major code generation is selected, the build stops with an error.



# New Features/Migration History of the ASM Blocksets

## Introduction

The following topics provide an overview of the changes to the ASM products in the previous Releases.

For an overview of the new features and migration of the current Release, refer to [Automotive Simulation Models \(ASM\) \(New Features and Migration\)](#).

## Where to go from here

## Information in this section

<a href="#">Changes to all ASM Products.....</a>	295
--	-----

Provides an overview of all the new features and migration of the ASM Blocksets in the previous Releases.

<a href="#">History of the ASM Utils Library.....</a>	300
---	-----

Provides an overview of the new features and migration of the ASM blockset in the previous Releases.

## Changes to all ASM Products

### Release 2020-B

All ASM demo models are now designed in a new, more modular structure to meet the requirements of today's model-based development. For detailed information, refer to [ASM Model Structure](#) on page 15.

### Release 2019-B

**ASM Engine Testbench** ASM Engine Testbench has new functionalities:

- A list for defining parameters and values has been added to the Online Settings page. These parameters are set to the specified values when the online test bench simulation is started on a dSPACE platform.

- The Plot Selected button has been added to the Execute page to only show the selected signals.

---

**Release 2019-A**

**VEOS code generation** From this Release on, ASM supports only the 64 bit version of VEOS. Before building the code, make sure HostPC64 is set as the simulation target on the Build Options page during import to the VEOS Player.

---

**Release 2018-A**

**Migration** The ASM migration supports the migration from the last ten Releases to dSPACE Release 2018-A. If you want to migrate from an older Release than the supported versions, the migration may fail. In this case, migrate to an intermediate Release and afterwards to the current Release. For more information, refer to [Migrating ASM Models](#) on page 67.

**Discontinuation of Rapid Accelerator support for ASM products** The support of Simulink's Rapid Accelerator Simulation Mode has been discontinued for all ASM products.

---

**Release 2017-A**

**ASM demo models** The standard platform in the model startup go files is changed from RTI platforms to SCALEXIO.  
As a consequence, if you use RTI platforms (e.g., DS1005, DS1006), you need to start the model with `go('platform', 'RTI')` before starting code generation to choose proper compiler options.  
If you use SCALEXIO, you can use the model immediately to trigger the code generation from ConfigurationDesk.  
You can find more information on the input arguments of the go file and the model startup procedure in the relevant file header and the tutorials in the ASM Model Descriptions.

---

**Release 2016-B**

**Look-up table migration** The discontinued Simulink blocks Lookup and Lookup2D within the ASM library blocks are updated to the new standard Simulink Lookup Table (n-D) block.  
For a list of the updated blocks of each ASM blockset, refer to the migration information of the respective ASM blockset.  
The migration of the look-up table settings is performed along with the recommended migration steps in MATLAB. However, as described in the release notes of MATLAB R2011a, special handling in case of repeated breakpoints is required. Repeated breakpoints were not prohibited in the former look-up table blocks. The ASM migration only handles issues with repeated breakpoints for parameters set in the provided ASM demo models. These blocks consequently get a special migration treatment. Custom parameterizations with repeated breakpoints are not treated by this.  
To also ensure the expected simulation behavior for these cases, you have to manually migrate them. More information along with a Python script, which helps to re-establish the ControlDesk connections, can be found here:

<https://www.dspace.com/en/pub/home/support/kb/dsutil/utilasm/utilasmlookuptblconn.cfm>

**Discontinuation of ASM\_UTILS license** The ASM\_UTILS license is discontinued. The license check now works with any other ASM license. For a list of the updated blocks of each ASM blockset, refer to the migration information of the respective ASM blockset.

#### Release 2016-A

**Scope handling** During migration, all disabled scopes within the model need to be restored to ensure that the model works properly, including the new scope handling procedure. You might have to disable these scopes again after migration. You can easily disable scopes via the Scope Handling GUI button provided by ASM. This makes direct use of the commenting feature in Simulink.

#### Release 2015-B

**Model start script go.m file** The standard start script `go.m` file for the ASM demo models is adapted to take the changes of the supported simulation platforms into account.

The revised `go.m` file now contains the following platform options:

- 'RTI': DS1005, DS1006, DS1007, MicroLabBox
- 'SCALEXIO': SCALEXIO hardware
- 'VEOS': VEOS (offline simulation platform)

For more information on the calling procedures of the `go.m` file, refer to the corresponding file header or [Model Initialization](#) on page 35.

The start script `go.m` file also takes the changes to the VEOS target into account. With Release 2015-B, VEOS uses the DSRT target instead of the DSOFFSim target (refer to [Migrating from VEOS 3.5 to 3.6 \(VEOS Manual\)](#)). The `go.m` file in the ASM demo projects is adapted to trigger the correct compiler settings.

#### Release 2015-A

For rapid accelerator code generation the Microsoft Windows SDK 7.1 compiler must be used for the 32-bit version also. Code generation for Rapid Accelerator Mode with another compiler will fail (e.g., with the *don't know how to make.../cc.lib* error message for the LCC compiler).

#### Release 2014-B

**ModelDesk Processing** ModelDesk provides the *Processing* functionality. It supports the execution of calculation functions together with the administration of measurement information.

With this new feature, ModelDesk supports the ASMPParameterization Tool functionalities, among others. As of this release, new users of mean value engine models are required to start an engine parameterization project in ModelDesk, because no demo engine parameterizations are provided for mean value engine models. Nevertheless, the ASMPParameterization Tool is still provided in the dSPACE installation, allowing the migration from previous releases. In this way,

users are able to use existing parameterization projects also in the current release.

**ModelDesk export** The Parameter Export to ModelDesk page is no longer displayed. The user is requested to use an import functionality provided with the new ModelDesk *Processing* feature.

Importing parameters in ModelDesk is supported by the GeneralSettings file with the following content:

```
****  
function Settings = MySettings()  
Settings.PreFcn = '<Filepath with filename and extension of  
ImportMDLfromWorkspace.m>';  
***
```

Make sure to copy the ImportMDLfromWorkspace.m file from the installed engine parameterization demo ZIP file beforehand:

```
<dSPACE_Root>\Demos\ASM\<EnginePackage>\Parameterization\<Model  
Desk-  
Project.zip>\Pool\Processing\Function\PreFdns\ImportMDLfromWork  
space.m.
```

When you evaluate the processing configuration in ModelDesk, the command reads the current MATLAB workspace structure and imports the existing parameter to the ModelDesk parameter set of the simulation model used in the ModelDesk experiment. Evaluation of the processing configuration is done via the corresponding button on the Processing Configuration page in ModelDesk.

---

#### Release 2013-B

**User documentation renamed** To facilitate finding user documentation in the **Print** folder of dSPACE Help, the PDF file names are changed to match the titles of the documents. The modifications are also applied to the related CHM files in the **Online** folder.

For ASM, the following documents are relevant:

Document	Old Name	New Name
ASMDieselEngineModelDescription	ASMMODELDESCRIPTIONDiesel.pdf	ASMDieselEngineModelDescription.pdf
ASMDieselEngineInCylinderModelDescription	ASMMODELDESCRIPTIONDieselInCylinder.pdf	ASMDieselEngineInCylinderModelDescription.pdf
ASMGasolineEngineBasicModelDescription	ASMMODELDESCRIPTIONGasolineBasic.pdf	ASMGasolineEngineBasicModelDescription.pdf
ASMGasolineEngineModelDescription	ASMMODELDESCRIPTIONGasoline.pdf	ASMGasolineEngineModelDescription.pdf
ASMGasolineEngineInCylinderModelDescription	ASMMODELDESCRIPTIONGasolineInCylinder.pdf	ASMGasolineEngineInCylinderModelDescription.pdf
ASMTrailerModelDescription	ASMMODELDESCRIPTIONTrailer.pdf	ASMTrailerModelDescription.pdf
ASMVehicleDynamicsModelDescription	ASMMODELDESCRIPTIONVehicleDynamics.pdf	ASMVehicleDynamicsModelDescription.pdf

Document	Old Name	New Name
ASMVehicleDynamicsVEOSModelDescription	ASMMODELDESCRIPTION VehicleDynamicsVEOS.pdf	ASMVehicleDynamicsVEOSModel Description.pdf

---

**Release 2013-A**      **Support of MATLAB 64 bit**      ASM supports MATLAB 64 bit as modeling and simulation platform.

---

**Release 7.4**      **Support of new Simulink SLX model format**      ASM and ModelDesk support the new Simulink SLX model format. With MATLAB R2012b, models containing ASM blocks can be opened or saved in MDL or SLX format. Both formats can be used in ModelDesk projects and for custom components libraries.

**Problems using ASM with MATLAB R2012a and R2012b**      Due to performance problems with in MATLAB R2012a and R2012b, it is recommended to install the following bugfix from the MathWorks® website before using ASM with MATLAB R2012a or 2012b:  
[http://www.mathworks.de/support/search\\_results.html?q=827771](http://www.mathworks.de/support/search_results.html?q=827771)

---

**Release 7.3**      **Multi-instance support**      It is now possible to use the same ASM block of an ASM library several times in one model. The multi-instance feature is used to get ModelDesk support for each of these blocks. To get an explicit description of duplicate blocks, each ASM block has its own instance ID and instance name. The instance ID must be different for duplicate blocks of the same type. It is used together with the instance name to handle and show each block in ModelDesk. For details, refer to [Multi-Instance](#) on page 53.

**Problems using ASM with MATLAB R2012a**      Due to problems with signal buses in MATLAB R2012a, it is not recommended to use ASM with MATLAB R2012a.

---

**Release 6.6**      **Simulink Rapid Accelerator mode**      ASM supports the Simulink Rapid Accelerator mode which can increase simulation performance.  
 Limitation: The Simulink Rapid Accelerator mode cannot be used with an ASM operator version and the MotionDesk Blockset which is used for animation.

**ASM migration**      The migration process is redesigned. For details, refer to [Migrating ASM Models](#) on page 67.

---

**Release 6.5**      **3-D Object Library**      The textures of some objects in the 3-D library are updated with a new look.  
 The 3-D objects of the ASM MotionDesk demo experiments are automatically updated with the new object textures that have a more natural look.

**MotionDesk Blockset**      The MotionDesk Blockset is migrated from a previous version to version 2.0 automatically. You must only open the real-time model

and rebuild the real-time application. For detailed information on the blockset, refer to [MotionDesk Blockset \(MotionDesk Calculating and Streaming Motion Data\)](#).

---

**Release 6.4**

**ModelDesk support** The parameterization of ASM Engine Gasoline Basic, ASM Engine Gasoline and ASM Engine Diesel is supported by ModelDesk. Parameter sets generated in ASMPparameterization can be exported to ModelDesk projects, organized in experiments and downloaded to the simulation platforms. In ModelDesk, ASM models can be configured and parameters can be changed, saved, downloaded, etc. Parameters can also be exported back to MATLAB INI files.

## History of the ASM Utils Library

---

**Release 2020-B**

**ASMSignalInterface** The ModelDesk PlotSignalCollector has been renamed to *ASMSignalInterface*. It is now possible to use several instances of the block in the model. Each ASM module contains one separate *ASMSignalInterface* block. For more information, refer to [ASMSignalInterface](#) on page 259.  
After migration of the model, the MATLAB Command Window shows a message containing a link. Click the link to update the *ASMSignalInterface*.

**ASMSignalInterface Manager** The *ASMSignalInterface* Manager replaces the Toggle ModelDesk Plotting block to support the multi-instance capability of the *ASMSignalInterface*. For more information, refer to [ASMSignalInterface Manager](#) on page 261.

---

**Release 2020-A**

**'asm\_proc' function** ModelDesk increases parameter precision from 6 to 14 significant digits. Most applications do not require this precision. If you use ModelDesk Processing, e.g., in table generation, the *asm\_proc* function rounds to 6 significant digits by default.  
For information on changing this behavior, refer to the new *set\_round* method of the *asm\_proc* function.

---

**Release 2019-A**

**Customized variable time delay** A customized variable delay block based on an S-function has been introduced. The delayed signal is calculated in discrete time and no interpolation/extrapolation is performed for the values between successive simulation steps. The block delivers a precise result when the delay time is not continuously changed during the simulation. Therefore, for real-time applications where the delay time is not altered continuously or when small

errors resulting from such changes are negligible, the block shows a significant reduction in turnaround time.

**Unit Conversion block** The Unit Conversion block has been optimized. Depending on the selected conversion, only the required operations are inserted (multiply with a gain and/or add an offset).

**Compare Value block** The output of the Compare Value block has been renamed from Const\_match[0|1] to Const\_mismatch[0|1].

The behavior is unchanged. The output value is:

- 0 if the input matches the reference value
- 1 if the input does not match the reference value

**ModelDesk processing** The automatic generation of a measurement file has been discontinued. The existence of the `WriteMeasFile` struct field in the general settings now triggers a warning.

Until now, the measurement file had been used by ASM Engine Testbench and ASM Optimizer. Now, both can read the related data directly from the ModelDesk measurement data (MD) file, including the calculated measurement variables. Remove the `WriteMeasFile` field in the general settings and import the measurement data to your ASM Testbench and ASM Optimizer project.

In special cases (e.g., if measurement data is combined to one measurement file), you have to introduce an *additional function* in ModelDesk Processing.

This function is available with the current ModelDesk Engine demo file at:

```
Pool\Processing\Function\PostFcns\
WriteCombinedMeasurementFile.m
```

**ASM Engine Testbench** When you import ModelDesk measurement data files to the ASM Engine Testbench, the measurement data variables are used instead of the raw data variables. Therefore, variables that result from measurement functions in ModelDesk can be used in ASM Engine Testbench. In contrast to the raw data variables, the measurement data variables have a fixed unit defined by the measurement type. This reduces the risk of errors when handling raw data with different units.

#### Release 2018-B

**Multi-instance support for custom component library blocks** An ASM block of an ASM library can be used multiple times in a model. With ModelDesk's multi-instance feature, each of these blocks can be accessed. Now, the multi-instance feature also supports blocks from a ModelDesk Custom Component Library.

For more information, refer to [How to Migrate Blocks of Custom Component Libraries](#) on page 75.

#### Release 2018-A

**asm\_proc** The internal storing of parameters changed. Now, the results are rounded with the same precision which is used by ModelDesk (currently 6 digits).

The result of executing multiple functions at once or executing each function individually is now the same. The changes can lead to slightly different results compared to previous Releases if multiple functions are executed at once.

**asm\_tablegenerator** The internal map update handling changed. In previous Releases, the exact values set with the options `xv_set`, `yv_set`, and `xyv_set` are not always achieved if further extrapolation steps are performed. This issue was resolved.

The changes can lead to slightly different results compared to previous Releases.

---

#### Release 2016-B

The discontinued Simulink blocks `Lookup` and `Lookup2D` within the ASM library blocks have been updated to the new standard Simulink `Lookup Table (n-D)` block.

The `Utils` blocks in the following blocks within this library have been updated:

- `Forward Euler`
- `asm_1d_integral_recip_lut`
- `asm_1d_lookup`
- `asm_2d_lookup`
- `asm_shiftable_lookup`
- `compare_value`

The option of the `Bus2Vector` block to generate separate `Vector2Datatype` and `Datatype2Vector` blocks was discontinued. The conversion is now always included in the `Bus2Vector` and `Vector2Bus` blocks.

When the `Bus2Vector` block is updated, you have to delete the related `Vector2Datatype` and `Datatype2Vector` blocks manually.

# Appendix

## Bibliography

### List of literature

The following literature provides more details:

- [Foe94]** Föllinger, O.: Regelungstechnik: Einführung in die Methoden und ihre Anwendungen. Heidelberg: Hüthig, 1994.
- [Hyd03]** Hydro-Quebec TransEnergie Technologies: SimPowerSystems for Use with Simulink - User's Guide. Version 3, 2003.
- [Oga10]** Ogata, K: Modern Control Engineering. New Jersey, Pearson Education 5th edition, 2010.
- [Sch09]** Schröder, D.: Elektrische Antriebe - Regelung von Antriebssystemen. Berlin: Springer, 2009.



## Numerics

1-D look-up table  
variable structure 49  
1D\_PsiTable method  
basics 216  
function call 216  
2-D look-up table  
variable structure 50  
2D\_Table method 205  
basics 206  
execution order 209  
function call 206

## A

ActivateDeveloperVersion block 194  
ActivateOperatorVersion block 194  
additional functions 250  
ASM blocksets  
limitations 293  
ASM Controller Adaption 107  
ASM Engine Testbench 78  
basic concepts 78  
Execute page 79  
General Settings page 82  
Plots page 85  
starting 90  
Stimulus Signals page 83  
user interface 79  
ASM example model  
starting via MATLAB 268  
ASM installation folder 43  
ASM LabelInterface 137  
opening the library 138  
ASM library  
opening 266  
ASM Model Tools 27  
ASM models  
migrating 67  
ASM module 17  
ASM project folder 43  
ASM RoadConverter 93  
Conversion page 99  
folder structure 95  
functional description 94  
Main page 97  
Mapping page 98  
user interface 97  
ASM submodule 24  
ASM task submodule 25  
asm\_1d\_integral\_recip\_lut block 191  
asm\_1d\_lookup block 188  
asm\_2d\_lookup block 189  
asm\_eng\_drivingcycles 200  
asm\_multiinstance 201  
asm\_plot\_geosusp 204  
asm\_plot\_table 203  
asm\_proc  
calc method 230  
check\_measurement method 236

check\_parameter method 230  
get\_measurement method 237  
get\_paramchilds method 231  
get\_parameter method 232  
get\_plotconfig method 240  
set\_measurement method 238  
set\_measurementnumber method 235  
set\_parameter method 234  
write2log method 239  
asm\_proc function 228, 229  
asm\_shiftable\_lookup block 190  
asm\_tablegenerator 205  
parameters for extrapolation 211  
parameters for plotting 214  
asm\_traffic.mdl\_update\_num\_fellows 219  
ASM\_Utils.lib library 153  
ASMSignalBus 31  
ASMSignalInterface block 259  
ASMSignalInterface Manager block 261

## B

Bus2Vector block 140

## C

CALC\_CASTER\_LEFT block 177  
ChangeLabels block 151  
Common Program Data folder 12  
compare\_value block 182  
CreateIndependentParameterxml.py 220  
CROSS\_PRODUCT block 176  
CROSS\_PRODUCT\_2 block 176  
Custom Component block 263  
custom component library  
migrating 75  
Custom Component Migration block 264

## D

dimension parameters 57  
directory structure 42  
Documents folder 12  
Domain control 25  
DomainControl 25  
DOT\_PRODUCT block 177  
driving cycles  
loading 200

## E

engine parameterization test 91  
EngOP\_Selector block 185

## F

FirstOrderDynamics block 157  
Flush Denormals to zero block 185  
folder  
ASM installation 43  
ASM project 43  
folder structure 42  
ForIteratorDelay block 156

Forward\_Euler block 155

## G

general settings file 247  
GenerateVEOS block 196  
generating code 272  
geometrically described suspension  
visualizing 204  
go.m 35

## I

ImportParamfromWorkspace.m 246  
initialization files 253  
initializing the models 35

## L

limitations  
ASM blocksets 293  
loading driving cycles 200  
Local Program Data folder 12  
local subsystem oversampling 63

## M

MANEUVER\_START block 162  
MANEUVER\_STATE block 164  
MANEUVER\_STOP block 163  
matrix  
variable structure 49  
measurement functions 252  
MemoryWithReset block 159  
migrating  
ASM models 67  
custom component library 75  
migration variants 73  
model initialization 35  
MODEL\_DESK\_WRITE\_ACCESS block 167  
ModelDesk interface 160  
ModelDesk Parameter Group block 167  
ModelDesk\_MC\_ID\_EngineBasic block 168  
ModelDeskParaGroupOverview block 166  
multi-instance 53  
MultiInstanceOverview block 165

## N

number of fellows  
updating 219  
numerical aspects 63

## O

OpenExperiment block 193  
opening  
ASM library 266  
operator version 61  
output structure 208  
oversampling  
ASM 25

**P**

parameter files  
    renaming 220  
parameterization data  
    plotting 203  
plotset 78

scalar, vector, matrix 49  
vector  
    variable structure 49  
visualizing geometrically described suspension 204

**R**

RANDOM\_NUMBERS\_NORMAL  
    block 197  
real-time simulation 270  
renaming  
    parameter files 220  
ReplaceTireMF block 196  
ReplaceTireTMEasy block 195  
Reset block 164  
ROTATION\_MATRIX\_2\_ANGLES block 174  
ROTATION\_MATRIX\_ANGLES block 173  
ROTATION\_MATRIX\_ANGLES\_SMALL block 174

**S**

scalar  
    variable structure 49  
ScopeHandlingUser interface block 192  
SecondOrderDynamics block 158  
Select Maneuver block 186  
semi-implicit Euler stabilization 65  
simulation mode 36  
Simulink simulation 270  
SKEW\_SYMMETRIC\_MATRIX block 173  
starting  
    ASM Engine Testbench 90  
starting via MATLAB  
    ASM example model 268  
stimulus data 78  
structure of ASM folder 42  
submodule 24, 25

**T**

task submodule 25  
test cycle 184  
testing  
    engine parameterization 91  
Transform\_GammaAlphaBeta block 175  
TRANSPOSE block 172  
troubleshooting 277

**U**

UNIT\_VECTOR block 171  
UNIT\_VECTORS block 171  
UnitConversion block 169  
user interface  
    ASM Engine Testbench 79

**V**

variable structure 47  
    1-D look-up table 49  
    2-D look-up table 50