## Contents

1. **Important notes on Dymola** ................................................................. 5
2. **About this booklet** ................................................................................. 5
3. **Dymola 2019** ......................................................................................... 6
   3.1 **Introduction** ..................................................................................... 6
      3.1.1 Additions and improvements in Dymola ........................................... 6
      3.1.2 New and updated libraries ................................................................. 7
   3.2 **Developing a model** ......................................................................... 8
      3.2.1 Support for Modelica 2 in Dymola discontinued ........................... 8
      3.2.2 Capturing the parameter values and other modifiers of a component ......................................................................................................................... 8
      3.2.3 Filtering of inherited components in diagram ............................... 13
      3.2.4 Improved plotting in the matrix editor ........................................... 15
      3.2.5 Presenting display units in the diagram layer ............................... 17
      3.2.6 Improved presentation of display units ........................................... 19
      3.2.7 Customizing the File > Libraries menu ....................................... 22
      3.2.8 Adding variables to bus declarations when connecting expandable connectors .......................................................... 24
      3.2.9 Minor improvements ....................................................................... 26
   3.3 **Simulating a model** ......................................................................... 33
      3.3.1 DAE Mode .................................................................................... 33
      3.3.2 Improvements in Variable Browser .............................................. 34
      3.3.3 Plot window .................................................................................. 38
      3.3.4 Scripting ..................................................................................... 40
      3.3.5 New features for debugging initialization failures ....................... 40
      3.3.6 Improved nonlinear solver diagnostics ...................................... 44
      3.3.7 Improved initial guesses for nonlinear equations in the model during simulation .......................................................... 47
      3.3.8 Minor improvements ....................................................................... 48
   3.4 **Installation** ..................................................................................... 54
      3.4.1 Dymola as a 32-bit application discontinued .................................. 54
      3.4.2 Scientific Data Format (SDF) packages for Python and Matlab available on GitHub .......................................................... 55
      3.4.3 Installation on Windows .................................................................. 55
   3.5 **Other Simulation Environments** ..................................................... 57
      3.5.1 Dymola – Matlab interface ............................................................ 57
      3.5.2 Real-time simulation ..................................................................... 57
      3.5.3 FMI Support in Dymola ................................................................. 58
      3.5.4 Source Code Generation ............................................................... 66
3.6 Modelica Standard Library and Modelica Language Specification ................................................................. 67
3.7 New libraries ...................................................................................................................................................... 67
  3.7.1 ClaRa DCS Library ................................................................................................................................... 67
  3.7.2 Electric Power Systems Library .............................................................................................................. 68
  3.7.3 Pneumatic Systems Library ................................................................................................................... 68
3.8 Updated libraries ............................................................................................................................................... 70
  3.8.1 Battery Library ...................................................................................................................................... 70
  3.8.2 ClaRa Grid Library ................................................................................................................................ 71
  3.8.3 ClaRa Plus Library ............................................................................................................................... 71
  3.8.4 Cooling Library ...................................................................................................................................... 72
  3.8.5 Dymola Commands Library ................................................................................................................ 72
  3.8.6 Electrified Powertrains Library .............................................................................................................. 72
  3.8.7 Fluid Dynamics Library ......................................................................................................................... 73
  3.8.8 Fluid Power Library ............................................................................................................................... 73
  3.8.9 FTire Interface Library .......................................................................................................................... 73
  3.8.10 Human Comfort Library ...................................................................................................................... 73
  3.8.11 HVAC (Heating, Ventilation, and Air Conditioning) Library .............................................................. 73
  3.8.12 Hydrogen Library ................................................................................................................................. 73
  3.8.13 Optimization Library .......................................................................................................................... 74
  3.8.14 Testing Library .................................................................................................................................... 74
  3.8.15 Thermal Systems Library ..................................................................................................................... 75
  3.8.16 VeSyMA (Vehicle Systems Modeling and Analysis) Library .............................................................. 75
  3.8.17 VeSyMA – Engines Library .................................................................................................................. 78
  3.8.18 VeSyMA – Powertrain Library ............................................................................................................ 81
  3.8.19 VeSyMA – Suspensions Library ........................................................................................................ 83
3.9 Documentation .................................................................................................................................................. 88
3.10 Appendix – Installation: Hardware and Software Requirements ........................................................................ 89
  3.10.1 Hardware requirements/recommendations ........................................................................................... 89
  3.10.2 Software requirements .......................................................................................................................... 89
1 Important notes on Dymola

Installation on Windows

To translate models on Windows, you must also install a supported compiler. The compiler is not distributed with Dymola. Note that administrator privileges are required for installation. Three types of compilers are supported on Windows in Dymola 2019:

Microsoft Visual Studio C++

This is the recommended compiler for professional users. Note that free Microsoft compiler versions earlier than Microsoft Visual Studio Express 2008 are not supported (concerning full versions, some earlier versions are supported). Refer to section “Compilers” on page 89 for more information.

Intel

Dymola 2019 has limited support for the Intel Parallel Studio XE compiler. For more information about this compiler, see section “Compilers” on page 89; the section about Intel compilers.

GCC

Dymola 2019 has limited support for the MinGW GCC compiler, 32-bit and 64-bit. For more information about GCC, see section “Compilers” on page 89; the section about GCC compilers.

Installation on Linux

To translate models, Linux relies on a GCC compiler, which is usually part of the Linux distribution. Refer to section “Supported Linux versions and compilers” on page 91 for more information.

2 About this booklet

This booklet covers Dymola 2019. The disposition is similar to the one in Dymola User Manual Volume 1 and 2; the same main headings are being used (except for, e.g., Libraries and Documentation).
3 Dymola 2019

3.1 Introduction

3.1.1 Additions and improvements in Dymola

A number of improvements and additions have been implemented in Dymola 2019. In particular, Dymola 2019 provides:

- DAE mode for more efficient simulation of large models (page 33)
- Improved initial guesses for equations in the model during simulation (page 47)
- New features for debugging initialization failures (page 40)
- Improved nonlinear solver diagnostics (page 44)
- Capturing parameter values and other modifiers by extending a component to a new model, or by creating a favorite model from a component (page 8)
- Filtering of inherited components in the diagram layer of editable models (page 13)
- Improved plotting of Boolean and short Integer arrays in the matrix editor (page 15)
- Presentation of display units in the diagram layer (page 17)
- Improved presentation of display units in the parameter dialog (page 19)
- Customization of the File > Libraries menu (page 22)
- Possibility to add variables to bus declarations when connecting expandable connectors (page 24)
- Variable browser improvements:
  - Selecting display units (page 34)
  - Improved filtering (page 36)
- Plot improvements (page 38)
  - Signal operators for curves plotted against independent variables other than time
  - Improved presentation of display units
- Support for Visual Studio 2017 compiler (page 55)
- Improved FMI support
  - General
    - Reorganization of simulation setup dialog for FMI (page 58)
  - FMU Export
    - Defining user-defined model identifiers for FMUs in GUI when exporting (page 61)
    - Sparse Jacobian handling when including source code supported (page 62)
Sparse solver support (page 63)
Export of models in DAE mode (page 63)
More GUI settings available as flags for scripting
  o FMU Import
    ▪ More GUI settings available as flags for scripting
  o FMI Kit for Simulink
    ▪ A new version 2.4.2 with support for Matlab R2017a and R2017b, and other improvements (page 66)
  ▪ Sparse Jacobian handling supported for Source Code Generation (page 66)
  ▪ New document: Dymola Referential (page 88)

3.1.2 New and updated libraries

New libraries
The following libraries are new in this Dymola version:

Commercial libraries
  • ClaRa DCS Library, version 1.1.0
  • Electric Power Systems Library, version 1.0
  • Pneumatic Systems Library, version 1.0

For more information about the new libraries, please see the section “New libraries” starting on page 67.

Updated libraries
The following libraries have been updated:
  • Battery Library, version 2.1
  • ClaRa Grid Library, version 1.1.0
  • ClaRa Plus Library, version 1.1.0
  • Cooling Library, version 1.2.1
  • Dymola Commands Library, version 1.5
  • Electrified Powertrains Library, version 1.2.1
  • Fluid Dynamics Library, version 2.5.0
  • Fluid Power Library, version 2018.1
  • FTire Interface Library, version 1.0.1
  • Human Comfort Library, version 2.5.0
  • HVAC (Heating, Ventilation, and Air Conditioning) Library, version 2.5.0
  • Hydrogen Library, version 1.1
  • Optimization Library, version 2.2.3
  • Testing Library, version 1.1
• Thermal Systems Library, version 1.2.0
• VeSyMA (Vehicle Systems Modeling and Analysis) Library, version 2018.1
• VeSyMA - Engines Library, version 2018.1
• VeSyMA - Powertrain Library, version 2018.1
• VeSyMA - Suspensions Library, version 2018.1

For more information about the updated libraries, please see the section “Updated libraries” starting on page 70.

3.2 Developing a model

3.2.1 Support for Modelica 2 in Dymola discontinued

Modelica 3.0 was standardized in September 2007 and Modelica Standard Library 3.0 was introduced in March 1, 2008. Modelica 2 traces its origins back to 2004.

The support for Modelica 2 in Dymola is discontinued from Dymola 2018 FD01. Neither simulation nor editing of Modelica 2 models is supported. Modelica Standard Library (MSL) 2.x is not available.

Conversion from Modelica 2 to Modelica 3 is still possible in this version, but it is easier to verify the conversion in a previous version of Dymola, e.g. Dymola 2018, where Modelica 2 is still supported. For more information and tips about conversion, see MigrationModelica2.pdf.

3.2.2 Capturing the parameter values and other modifiers of a component

In Dymola 2019 two new commands for capturing parameter values and other modifiers of a component are available, Capture Parameters > As Model and Capture Parameters > As Favorite. The commands are available as context commands for components in the diagram layer and component browser. The context menu in the diagram layer:
The context menu in the component browser:
Capturing the parameter values of a component by extending to a new model

If you select a component in the diagram layer or component browser, right-click and select Capture Parameters > As Model, the component’s model will be extended to create a new model. Parameter values and other modifiers are included in the extension.

As a minor example, consider the following model MyModel with just one parameter MyParameter with no default value, however given the value 33 by using the parameter dialog:
Right-clicking it in the diagram or component browser and selecting **Capture Parameters > As Model**, gives the dialog for extending the component’s model. Giving the new model the name `MyModelCapturedParameters` in the dialog:

![Extend From MyPackage.MyModel dialog](image)

The code of the new model will be:

```plaintext
model MyModelCapturedParameters
  extends MyModel(MyParameter=33);
end MyModelCapturedParameters;
```

Instantiating the new model in the diagram and looking at the parameter dialog gives:
Capturing the parameter values of a component by adding a new model to favorites

If you select a component in the diagram layer or component browser, right-click and select **Capture Parameters > As Favorite**, the component will added as a new model to favorites. Parameter values and other modifiers are included in the new favorite model.

As a minor example, consider the starting point from the example above, the model MyModel. Now let’s make this model a favorite instead by right-clicking it in the diagram or in the component browser and select **Capture Parameters > As Favorite**. The dialog that appears (if the package to add the favorite model to is selected to be Favorites):

The code for this favorite model will be:

```java
model MyModel = MyPackage.MyModel (MyParameter=33) ;
```

Instantiating this favorite model in the diagram and looking at the parameter dialog gives:
3.2.3 Filtering of inherited components in diagram

The Diagram Filter command has been extended to also include filtering of inherited components and connections for editable models.

As an example, using the command File > Demos > Motor Drive to open the motor drive demo, and then using the command New > Duplicate Class... to create an editable model, you can use the command Edit > Diagram Filter to activate the diagram filter. Now, you can, in the diagram filter, click Inherited to filter the inherited components of the model:
Notes:

- The model must be editable to be able to do this filtering.
- The Inherited button works in toggle mode.
- If other filtering has been implied in the diagram filter, that filtering is deactivated (and grayed) while the filtering of inherited components is active. Once the filtering of inherited components has been deactivated, the previous filtering is active again.
- The opacity level can be changed also for the filtering of inherited components.
- The previous setting Highlight inherited components in the Edit > Options... dialog has been removed.
3.2.4 Improved plotting in the matrix editor

Support for Boolean and short integer arrays

Boolean and short integer arrays are now plotted as expected; before they were also plotted, but displayed as real values.

A short integer array is by default defined as an array containing integers where the span between the minimum value and the maximum value is less than 10. This span can be changed by changing the value of the flag `Advanced.ShortIntegerSpan`. (The flag is by default 10.)

An example of plotting the following short integer array in the matrix editor:

```
Integer i[:]={1, 2, 0, 4, 8, 8, 7, 4, 3, 3, 6}
```

The curve is by default displayed using markers, marker style Filled Circle, and only the original points are marked, not the ones inserted to create the steps.

An example of plotting the following Boolean array in the matrix editor:

```
Boolean b[:]={true, false, true, true, false}
```
Support for Line Style and Marker Style for one curve

Plotting in the matrix editor now has support for line style and marker style for the curve in a plot containing only one curve. You can select using the context menu of the plot:
Note that for Boolean and short integer arrays that marker (Filled Circle) is the default.

**Support for toggling the grid**

The grid in a matrix editor plot can be hidden or displayed the same way as in the common plot window by using the context command **Toggle Grid**.

### 3.2.5 Presenting display units in the diagram layer

In Dymola 2019 display units are by default presented in the diagram layer. For example, the diagram layer of the demo Coupled Clutches now by default looks the following:

![Diagram of Coupled Clutches](image)

**Note:** The improved presentation of display units described in next section is valid also for the display units presented in the diagram.

The previous behavior of not presenting the display units in the diagram can be restored by unticking the setting **Use display unit for presenting parameter values**, reached by the command **Edit > Options...**, the **Graphical Editor** tab:
This setting is by default ticked. The setting corresponds to the flag 

`Advanced.UseDisplayUnitInDiagram=true`

If unticking the setting, the diagram layer of the demo Couple Clutches will look like in previous Dymola versions:
3.2.6 Improved presentation of display units

Parameter dialog improvements

Using \( \mu \), \( \Omega \), and °C instead of \( u \), Ohm, and degC

The characters \( \mu \), \( \Omega \), and °C are now presented instead of \( u \), Ohm, and degC in the display unit column. An example of using \( \Omega \) and °C is the parameter dialog of an ordinary resistor:

![Parameter dialog example](image)

Improved display of e.g. \( m^2 \) and \( m^3 \)

Power 2 and 3 of units, for example \( m^2 \) and \( m^3 \) are now presented in this way, using Superscript, instead of the previous way presenting m2 and m3.


**Entering values with prefixed units**

It is possible to enter values with prefixed units, also if that prefix is currently not selectable from the display unit after the value in the dialog. As an example, consider the parameter dialog of an ordinary constant current component (Modelica.Electrical.Analog.Sources.ConstantCurrent):

Default parameter dialog:

![Parameter dialog screenshot](image)

Default display unit selection:
Entering 2 uA (note that you must enter the blank between the value and the unit):

Clicking **OK** and displaying the parameter dialog again now gives:

The display unit selection has also been updated:

The display unit selection for Ampere will be updated for all display unit selections of Ampere.

Notes:
- It is possible to enter values with prefixed units also for parameters that have no alternative display unit (gray background in display unit column).
- The prefixed display unit update is not saved between sessions (the display unit will still be µA but this selection will not be displayed in the menu for selecting display unit, that is, that one will be reverted to the default one, until a change is being made of a µA value).
3.2.7 Customizing the File > Libraries menu

In Dymola 2019 you can customize the File > Libraries menu. The new command File > Library Management has been introduced:

This command contains two tabs. The Libraries tab lets you decide what libraries should be present in the File > Libraries menu, and the order of the libraries:
You can:

- Include or exclude all libraries by using the checkbox in the heading
- Include or exclude any library in the list by the checkbox in front of it
- Move the selected library upwards or downwards using the arrow buttons.

Any action is immediately implemented when clicking **OK**.

The setup is saved between sessions.

To restore the default library configuration, you can click **Reset** in the menu above.

The second tab **Modelica Path** is the previous command **File > Modelica Path…** that has been included in the **Library Management** command.

Note. If you change the Modelica Path, by for example adding a directory, the libraries to select from in the **Libraries** tab may change when you click **OK** in the **Modelica Path** tab.
3.2.8 Adding variables to bus declarations when connecting expandable connectors

It is now possible to add a new variable to the bus declaration when connecting expandable connectors. This is done by ticking the check box **Add variable to bus declaration** when adding a new variable by clicking **<Add Variable>** in the Create Connection menu. An example:

The next time this dialog is used, for example when connecting to another instance, the new variable is presented in the bus:
Note that the check box for adding a variable to the bus declaration is present also for variables that are visible under <Add Variable> when clicking it; ticking the check box for such a variable will “push” it above the <Add Variable> entry in the pane and add it to the bus declaration.

By default the variables added to bus declarations this way are stored between sessions, this is controlled by the selection Added variables in bus declarations that can be reached by the command Edit > Options..., the Settings tab:
If this setting is unticked, the variables added to bus declarations this way will not be present in the bus declaration dialogs when Dymola is restarted. (The added variables will still be present in the bus locally for the components where they have been added.)

3.2.9 Minor improvements

Base classes supported in instance hierarchy

Base classes are now supported in the instance hierarchy. As an example, opening the demo Motor Drive (by File > Demos > Motor Drive) and then right-clicking the gearbox in the diagram layer and selecting the command Show Component, followed by right-clicking flange_a and selecting the command Show Component now gives:
The base classes are shown with the same icon as in the component browser; the name of a base class can be shown by the tooltip (as above). Just like normal components, base classes can be navigated to by clicking them in the instance hierarchy.

(To compare, performing the same commands as above in a previous version of Dymola results in:}

name
Shift+double-click on a component in the diagram layer

Previously only double-click was supported on a component in the diagram layer. The action performed when double-clicking a component in the diagram layer can be configured using the Edit > Options… command, the Graphical Editor tab. The default action is to open the parameter dialog of the component.

Now Shift+double-click is also available for a component in the diagram layer. If double-clicking results in opening the parameter dialog, Shift+double-click on the component will perform the Show Component command. If double-clicking is configured to do something else than opening the parameter dialog, Shift+double-click on the component will open the parameter dialog.

Smart rename and smart delete available as GUI settings

How to update other classes when renaming or deleting a class or a component was previously available only as flags. Now they are also available in the GUI using the command Edit > Options…, the Package Browser tab (note than changing any of these settings does not require a new translation of the model).
The alternatives for renaming class and renaming components are the same. As example, looking at renaming classes:

The alternatives correspond to the alternatives of the flags
Advanced.ActivateSmartRenameClass (for renaming classes), and
Advanced.ActivateSmartRename (for components):
<table>
<thead>
<tr>
<th>GUI alternative</th>
<th>Flag value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>Ask in dialog</td>
<td>1</td>
</tr>
<tr>
<td>Yes [default]</td>
<td>2 [default]</td>
</tr>
<tr>
<td>Yes, also conversion scripts</td>
<td>3</td>
</tr>
</tbody>
</table>

Notes:
- In the dialog you get when having selected Ask in dialog, you can change the setting to Yes for future renaming by ticking Always update without confirmation and then clicking Update.
- In the last alternative, “also conversion scripts” means that the renaming operations are accumulated in a conversion-script, to enable conversion of libraries using the library that were not loaded during the conversion.

The alternatives for deleting classes or component correspond to the alternatives of the flag Advanced.ActivateSmartDelete:

The General tab and the Package Browser tab now looks the following:

<table>
<thead>
<tr>
<th>GUI alternative</th>
<th>Flag value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>Ask in dialog [default]</td>
<td>1 [default]</td>
</tr>
<tr>
<td>Yes, if possible</td>
<td>2</td>
</tr>
</tbody>
</table>

Note that in the dialog you get when having selected Ask in dialog, you can change the setting to Yes for future renaming by ticking Always update without confirmation and then clicking Update.

**The Edit > Options... dialog contains a new tab**

The dialog from the command Edit > Options... now contains a new tab Package Browser. The tab contains settings related to the package browser previously located in the General tab, as well as new settings related to updates of classes when, for example, renaming a class. See previous section for details about these new settings.

The General tab and the Package Browser tab now looks the following:
**Shortcuts for navigating model tabs**

Having a number of model tabs, **Ctrl+Tab** activates the tab to the right of the currently active tab in the model tabs toolbar; corresponding to moving down in the list of tabs (available to the right in the model tabs toolbar). **Ctrl+Shift+Tab** activates the tab to the left of the currently active one, corresponding to moving up in the list of tabs.

**Simplified insertion and editing of links in the documentation layer**

The insertion and editing of links in documentation has been simplified:

- When creating a link using the toolbar button **Link**, from a text that is a web address, the dialog for creation is prepopulated:
• If you have entered something that can be identified as a link in the text in the documentation layer and if you continue with pressing press Space, Enter or Tab, a link is automatically created. This feature can be disabled by setting the flag Advanced.AutoLinksDoc = false.

• To edit an existing link, you can place the cursor anywhere on the link, click the Link button in the toolbar, and edit the link or remove it by ticking Remove link.

MathML supported for inserting and editing equations and expressions in the documentation layer

You can now also use MathML format when inserting and editing equations and expressions in the documentation layer using the toolbar button Equation $\equiv$. The MathML expression needs to be surrounded with $<math>...</math>$ or $<mathml>...</mathml>$. An example:

Opacity in diagram filter saved between sessions

The diagram filter, activated by, for example, Edit > Diagram Filter, includes a slider to set the opacity of the components being filtered out. This setting can now also be controlled by the flag Advanced.DiagramFilterOpacity. The default value is 0.15. The value of the flag is saved between sessions.
**File > Modelica Path… integrated in File > Library Management**

The previous command **File > Modelica Path…** is now integrated in the new command **File > Library Management**, as the second tab **Modelica Path**. No other changes have been made for this command.

### 3.3 Simulating a model

#### 3.3.1 DAE Mode

Dymola 2019 supports DAE (Differential-Algebraic Equations) mode. DAE mode is activated by setting the flag

```
Advanced.Define.DAEsolver = true
```

DAE mode is supported by the integrators Dassl, Radau IIa, Esdirk23a, Esdirk34a, Esdirk45a, and Sdirk34hw. (The flag is by default false.)

A translated model may contain several nonlinear equation systems. (These are listed in the translation log under Statistics.) When DAE solver is not enabled these equations are solved numerically whenever the integrator calls the model. When DAE solver is enabled the equations in the output and dynamics section of the model are not solved during calls to the model. They are instead handled by the integrator as part of the nonlinear system of equations that the integrator solves each step. If the translated model contains several or large nonlinear equation systems, then DAE solvers may be more efficient since fewer nonlinear systems are solved.

When a simulation is run with the DAE flag enabled and one of the above solvers are used a message is printed in the simulation log. The message

```
The translated model is a DAE. Integrating using DAE solver.
```

means that there are nonlinear equations in the output and/or dynamics section of the translated model and that these are handled by the integrator as described above. The message

```
The translated model is an ODE. Integrating using DAE solver.
```

means that there are no nonlinear equations in the output and dynamics section of the translated model. However, the DAE solver formulation is still used. In this case, no performance differences should be expected compared to simulation without the flag enabled.

When trying to run a simulation in DAE mode with a solver that is not supporting DAE mode, the simulation is terminated and an error message is printed.

It is allowed to export a model where the DAE flag is set but the selected solver does not support DAE mode using Binary Model Export or Source Code Generation, but the model will generate error messages when trying to simulate it.
However, it is not allowed to export such a model as an FMU, neither as an FMU using model exchange or as an FMU using co-simulation. Trying to perform such an export, an error message is given.

### 3.3.2 Improvements in Variable Browser

**Selection of display units**

In Dymola 2019, you can select display units for signals in the variable browser, if display units are defined for the signals, and if the signals are editable. Click on the unit to get a selection like in the parameter dialog:

> ![Variable browser screenshot]

When you have selected the new display unit the value is converted.

When you plot the signal, the selected display unit is used in the plot.

**Context command to close all results in the variable browser**

For the top-level nodes in the variable browser, a new context command `Close All Results` can be given.
The command closes all results; however the initial values of the current model are still present. This facilitates giving new initial values to be able to make a re-simulation with better parameter settings.
If a new model is opened now, also the initial values will be closed, since they do not apply to the new model.

The behavior is the same as for the built-in function `removeResults()`. This function is available also in previous versions.

**Option to decide if filtering should be case sensitive or not, and changed default value**

In Dymola 2019 it is possible to decide if the filtering using the line at the bottom of the variable browser should be case sensitive of not. This can be changed by the command `Case Sensitive` in the context menu of the filter line:
By default the filtering is *not* case sensitive – this is changed from previous versions of Dymola where the filtering was case sensitive. Clicking the command gives (the command works in toggle mode):
When starting a new session, the filtering is always not case sensitive.

**The results of a failed initialization can be made available in the variable browser**

The results of a failed initialization can be made available in the variable browser. See “The results of the failed initialization can be plotted using the Variable Browser” on page 42.

### 3.3.3 Plot window

**Signal operators for curves plotted against independent variables other than time**

You can now apply signal operators on curves plotted against independent variables other than time. All signal operators are supported, except First Harmonic, Total Harmonic Distortion (THD), and FFT (Fast Fourier Transform).

An example:

![Plot window](image)

Only independent variables with values rising over time are supported. Trying to apply a signal operator on an independent variable not fulfilling this will produce an error message.

Creating the supported signal operators by scripting, using `plotSignalOperator` or `plotSignalDifference` also works.
Command to close all plot windows

The new command **Plot > Close All Plot Windows** closes all plot windows in the current Dymola window.

Note that if you have opened a second Dymola window from the current one (by the command **Window > New Dymola Window**) only the plots in the window where the new window command was issued will be closed.

Display unit presentation in plots improved

The improved display unit presentation in the parameter dialog is also implemented in the plot. The improvements consist of:

- Displaying µ, Ω, and °C instead of u, Ohm, and degC
- Displaying power 2 and 3 using Superscript e.g. presenting m² and m³ instead of m2 and m3
- Displaying units with new prefix that have been used to enter the values, for example, having entered 10 kOhm instead of 10000 for a variable with unit Ohm, this value will be displayed as 10 kΩ or 10 kOhm.

These improvements are implemented for the curve and parameter tooltips, the legend, axis headings, and the curve context command **Display Unit**. For details on the improvements, see relevant parts of “Parameter dialog improvements” on page 19.

**Note**! Presently the variable browser is still unchanged, entering and presenting values has to be done as in previous versions.
3.3.4 Scripting

New built-in function GetDymolaCompiler (Windows)

A new built-in function GetDymolaCompiler has been added. The function extracts the compiler settings from the Compiler tab of the simulation setup (except the Customer options settings) to two output text parameters, compiler that is the compiler type, and settings[:] that contains the other settings. An example of a call and the output of that call:

GetDymolaCompiler()
    = "vs", ["CCompiler=MSVC", "IntelDir=C:/Program Files (x86)/IntelSWTools/compilers_and_libraries_2016.1.146", "IntelVsVersion=", "GCC32Path=C:/MinGW/bin/gcc.exe", "GCC64Path=C:/MinGW64/mingw64/bin/gcc.exe", "MSVCDir=C:/Program Files (x86)/Microsoft Visual Studio 14.0/Vc", "gcc32Disabled=0", "gcc64Disabled=0", "DDE=0", "DLL=0", "OPC=0"]

(The Customer options settings in the Compiler tab cannot be extracted by this function; they are handled by the string flags Advanced.LinkerOptions and Advanced.CompilerOptions, respectively.)

Note that the function extracts all settings except the above, not only the ones used.

The function can be used, for example, to test if an application relying on a specific compiler setup is supported on a certain machine by defining and populating the variables compiler and settings by the call (compiler,settings)=GetDymolaCompiler() and then comparing with demands.

The function is not supported on Linux; in Linux it returns just an empty list.

The function is available in the DymolaCommands library.

Note the related built-in function SetDymolaCompiler that can be used to define the compiler settings.

Improved built-in functions for signal operators

The built-in functions plotSignalOperator and plotSignalDifference have been extended to also work creating signal operators for curves plotted against independent variables other than time; with certain limitations. See “Signal operators for curves plotted against independent variables other than time” on page 38 for the limitations.

3.3.5 New features for debugging initialization failures

Activating the features

To activate the features below, you must, before the simulation, enable the new setting Store variables after failed initialization in the simulation setup. You can reach this setting by the command Simulation > Setup..., the Debug tab:
The setting is by default not enabled. The setting is saved between sessions; see also “Minor changes in the Debug tab in simulation setup, and more settings saved between sessions” on page 53.

**Plot Dependencies available also after failed initializations**

You can now use the feature of plotting dependencies also after failed initializations, if you have activated the setting in previous section before simulation. The feature is available in the same way as usual, note however that the presented dependencies are for the simulation problem, the dependencies for the initialization problem may differ, cf. dsmodel.mof.
The results of the failed initialization can be plotted using the Variable Browser

If you have activated the setting in the first subsection above before simulation, after a failed initialization all variables are now available in the Variable Browser, ready for plotting. The value of each variable depends on how far the initialization process came before failing. If the variable was initialized the result of that initialization is presented. If the variable was not initialized its start value is instead presented. The simulation log contains information on what made the initialization fail. The execution order can be found in e.g. dsmodel.mof (to generate this file, activate the setting Generate listing of translated Modelica code in dsmodel.mof before translation, the setting can be reached by the command Simulation > Setup..., the Translation tab).

Consider the example model:

```model InitializationFailure
  Real x(start=-0.5, fixed=true);
  Real dx(start=-0.9);
  input Real u;
  output Real y1(start=-1);
  output Real y2(start=-1);
  equation
    der(x) = dx;
    u*dx + exp(dx) = x;
    y1 = x^3;
    y2 = der(x)^3;
end InitializationFailure;
```

To compute the variable der(x) = dx Dymola will, during translation, generate a nonlinear equation in this variable. When simulating the model Dymola complains that it cannot solve this nonlinear equation and the initialization fails. Plotting the iteration variable der(x) = dx gives the last attempted value in the Newton iteration.
By a closer look at the model one can see that the nonlinear equation depends on the variables u and x. (This information is also readily available in the translation log if you enable the setting **List non-linear iteration variables** before translation, the setting can be reached by the command Simulation > Setup..., the Translation tab) or in dsmodel.mof (see above how to generate this file.) Plotting these variables reveals that u = 0 and x = -0.5.
Thus, the nonlinear equation reduces to \( \exp(dx) = -0.5 \), which lacks solution. The source of the initialization problem has been isolated.

Also note that a look into dsmodel.mof reveals that \( y1 \) is computed before the attempt to solve the nonlinear equation. The variable \( y2 \) is computed after. This agrees with the values presented when plotting these variables: \( y1 = -0.125 \) and \( y2 = -1 \). The former variable has its computed value \( x^3 \), whereas the latter variable has its start value \(-1\).

### 3.3.6 Improved nonlinear solver diagnostics

The message in the simulation log when there is a failure to solve nonlinear equations has been improved:

**More general information**

When there is a failure to solve nonlinear equations, the general message contains:

- More information about common causes for failures
- Extended list of settings that can be used to get more information, for example, the new settings in section “New features for debugging initialization failures” starting on page 40.
- A nonlinear solver summary, if the below option is activated.
Option to print a nonlinear solution summary

A new setting **Nonlinear solution summary** is available in the simulation setup. This setting is reached by the command **Simulation > Setup...**, the **Debug** tab:

The setting is by default not activated. If activated, a summary of the nonlinear solution is generated in the simulation log. An example:
If the above setting is active, the nonlinear solution summary is also included in the message given in the simulation log in the following cases:

- When solving a nonlinear equation fails
- When initialization fails
- When using profiling – note the tip about setting the flag `Advanced.GenerateBlockTimers` to activate profiling to log CPU times, see also “Improved profiling information” on page 51.

**More information when using the setting Nonlinear solution in the simulation setup**

If the setting **Nonlinear solution** in the simulation setup has been activated before translation, the information has been enhanced with:

- How many times each nonlinear equation is called
- A nonlinear solver summary, displaying accumulated results, if the setting **Nonlinear solution summary** has been activated.

(The simulation setup can be reached by the command **Simulation > Setup...**, the **Debug** tab.)
3.3.7 Improved initial guesses for nonlinear equations in the model during simulation

A translated Modelica model may contain nonlinear equations which are solved several times during integration. To enable fast and robust solution Dymola integrators provide initial guesses for these nonlinear equations. The guesses are dynamically updated during integration to incorporate the most recent values of the iteration variables of the nonlinear equation system.

In Dymola 2019 these initial guesses have been further improved, in particular for nonlinear equations in the accepted sections (Accepted Section and Conditionally Accepted Section) of the model equations.

(To display the output of manipulated equations in readable form to see the different equations sections, you must generate the output of manipulated equations in Modelica format and investigate the file dsmodel.mof. For more information about this, and examples, see Dymola User Manual Volume 1, section 5.5.6 “Output of manipulated equations in Modelica format”.)

The inter-/extrapolation method to be used to compute initial guesses can be controlled by using the flag

```
Advanced.Define.InitialGuessInterpolation
```

The flag can have any of the following values:

- 0 results in constant inter-/extrapolation in all sections
- 1 (default) meaning linear inter-/extrapolation is being used to define initial guesses in all sections except accepted sections
- 2 results in linear inter-/extrapolation only in accepted sections
- 3 results in linear inter-/extrapolation being used to define initial guesses in all sections.

Linear extrapolation typically performs better if the iteration variables vary smoothly as functions of time, otherwise constant extrapolation may perform better.

**Note** that only some solvers support linear inter/extrapolation. The solvers supporting it are

- Dassl
- Radau IIa
- Esdirk23a, 34a, and 45a
- Dopri45 and 853
- Sdirk34hw
- Cerk23, 34, and 45

Enabling the flag

```
Advanced.MoveEquationsToDynamics
```

moves all nonlinear equations in the accepted sections to the dynamics section. This may be useful if problems with nonlinear equations in the accepted sections are experienced,
especially if few output points are used. It forces the integrator to solve the nonlinear
equations each integrator step and thereby it also updates the initial guesses more often.

3.3.8 Minor improvements

Alias elimination of parameters

In Dymola 2019 alias elimination of parameters can be obtained, by setting the flag

```plaintext
Advanced.AllowParameterAlias=true
```

This eliminates parameters equal to other parameters – in the same way as alias elimination
of time-varying variables. This is primarily intended to reduce the size of the generated C-
code, and the only down-side is that such parameters are treated a bit specially when
exporting the model to FMI.

Efficient code generation for large tables

Large tables in Modelica models will by default yield a large amount of extra C code to be
compiled and fills up the variable browser with table data which may be of limited interest.
In Dymola 2019 it is possible to avoid this overhead by setting the annotations
Evaluate=true and HideArray=true on the table, combined with the new (see above
section) global setting Advanced.AllowParameterAlias=true. For example,

```plaintext
model HideArray
  parameter Real myTable[:,2]=[linspace(0, 30, 50), sqrt(linspace(0.0, 100, 50))];
  annotation (Evaluate=true, HideArray=true);
end HideArray;
```

Setting to use analytic ODE Jacobians available in GUI

In previous versions, to use analytic ODE Jacobians during simulation and also for
linearization, you could set the flag Advanced.GenerateAnalyticJacobian to true.

Now there is a corresponding setting Generate Analytic Jacobian for the OED problem
available in the GUI, in the simulation setup, the Translation tab:
The simulation setup can be reached by the command \texttt{Simulation > Setup…}.

The setting is by default not activated.

The setting is intended to improve the simulation speed, by computing the Jacobians more efficiently, but it generates a large amount of C code that needs to be compiled. Note that the speed improvement may not be as large as assumed since that case when the setting is not activated has also been optimized.

**More efficient event handling activated by default**

Already in previous Dymola versions a more efficient event handling of “minor” events could be activated by setting the flag \texttt{Advanced.EfficientMinorEvents = true}. In
Dymola 2019, this is the default value, since generally having this feature activated is a gain.

**Extended information about nonlinear systems of equations in the translation log**

In Dymola 2019, if activating the setting **List non-linear iteration variables** in the simulation setup (reached by the command **Simulation > Setup…**, the **Translation** tab) the following information is now also listed in the translation log when translating the model:

- Initial equations (previously only simulation equations were listed)
- Time-varying variables that the equations depend on (previously only the iteration variables were listed)
- Each system of equations is more precisely tagged

As an example, please compare the last part of the translation log for the model Modelica.Mechanics.Multibody.Examples.Loops.EngineV6 for Dymola 2019 (to the left) and Dymola 2018 FD01 (to the right):

**Improved logic for concealing parts of model.mof and dsmodel.mof**

The logic for concealing parts of **model.mof** and **dsmodel.mof** depending on encryption and the annotation **showDiagnostics** has been improved.
**Improved profiling information**

In the profiling information, given in the simulation log when the flag `Advanced.GenerateBlockTimers` is set to `true` before simulation, the tags for linear and non-linear equation systems are now used.

**Changed usage of compiled functions**

In Dymola 2019 expressions using compiled functions are not evaluated unless needed, for example, if they contain any parameter with the annotation `Evaluate=true`.

For compatibility reasons, if models have been constructed to rely on all expressions using compiled functions being evaluated, previous behavior can be preserved by setting the flag

```
Advanced.RelyOnCompiledFunctions=true
```

The flag is `false` by default.

**Several result files can be opened by multiselection**

The plot and animation commands for opening result files (`Plot > Open Result…` and `Animation > Open Result…`) now supports opening several result files by multiselection using `Shift` or `Ctrl` keys the usual way.

Note that the opened result files will be displayed in alphabetical order in the variable browser.

**Shortcuts for navigating recent windows**

Having a number of subwindows in a Dymola window (plot, animation, diagram, visualizer window), the already previously available command button `Recent Windows` can display a list of these by clicking the arrow to the right of the button. Now shortcuts are available for navigation in that list: `Ctrl+Tab` activates the subwindow below the currently active one in the list, while `Ctrl+Shift+Tab` activates the subwindow above the currently active one in the list.

**Progress bar in the taskbar icon for Dymola (Windows)**

When you simulate, you now get a progress bar in the taskbar icon for Dymola. During initialization, you get a green blob that moves back and forth. During simulation you get a progress bar based on the current simulation time. In the example below the max time was set to 2000 s.
Note that this progress bar is currently only available in Windows.

**Improved editing in the command window**

**Option to hide the toolbar of the command window**

In Dymola 2019 a new context command **Show Toolbar** makes it possible to display/hide the toolbar of the command window to save screen space:

![Show Toolbar button](image)

The toolbar is shown by default. The command works in toggle mode. If **Show Toolbar** is not activated, the command window looks like:

![Command window without toolbar](image)

The setting is saved between sessions.

**Command button to enable/disable mathematical notation in the command log**

A new command button in the command window toolbar can be used to enable/disable mathematical notation in the command log. The button works in toggle mode.

![Mathematical notation button](image)
The mathematical notation includes formulas and equations, Greek characters, and index rendering.

In the example above, the expression with beta has been entered with mathematical notation disabled, while the expression with gamma has been entered with mathematical notation enabled. Note that the flag controlling Greek character rendering (Advanced.RenderGreekLetters) is simply overrun when the mathematical notation is disabled; the value of the flag does not matter here.

By default the mathematical notation is disabled. This setting corresponds to the flag (which was available also in previous versions):

```plaintext
Advanced.MathematicalNotation.InCommandLog = false
```

The setting is saved between sessions.

**Simplified insertion and editing of links in the command log**

The insertion and editing of links in the command log using the command button **Link** in documentation editor has been simplified in the same way as when doing this in the documentation layer. See section “Simplified insertion and editing of links in the documentation layer” starting on page 31 for details.

**MathML supported for inserting and editing equations and expressions in the command log**

You can now also use MathML format when inserting and editing equations and expressions in the command log using the command button **Equation**, in the same way as doing this in the documentation layer. See section “MathML supported for inserting and editing equations and expressions in the documentation layer” starting on page 32.

**Editing of inserted equations and expressions harmonized with the corresponding command in the documentation layer**

In Dymola 2019, to edit an inserted equation/expression that has been inserted using the toolbar command button **Equation**, put the cursor directly in front of, or directly after the equation/expression and then click the command button **Equation**; this will bring up the edit dialog again - this is the same way to do it as in the documentation layer.

**Minor changes in the Debug tab in simulation setup, and more settings saved between sessions**

The settings in the **Debug** tab in the simulation setup (reached by **Simulation > Setup**...), the **Debug** tab) have been slightly rearranged, and all settings in the General group are now saved between sessions.
3.4 Installation

For the current list of hardware and software requirements, please see chapter “Appendix – Installation: Hardware and Software Requirements” starting on page 89.

3.4.1 Dymola as a 32-bit application discontinued

Dymola 2018 was the last version to support Dymola as a 32 bit application. Dymola 2018 FD01 and later versions are only available as 64-bit applications, on Windows and Linux.
Note however that only the Dymola 32-bit application is discontinued, the executable dymosim.exe, for example, is still available as a 32-bit executable.

### 3.4.2 Scientific Data Format (SDF) packages for Python and Matlab available on GitHub

The Scientific Data Format (SDF) packages for Python and Matlab have been removed from the Dymola distribution; they are now instead available on GitHub:

<table>
<thead>
<tr>
<th>Removed package</th>
<th>Now available on</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin\external\Python</td>
<td><a href="https://github.com/ScientificDataFormat/SDF-Python">https://github.com/ScientificDataFormat/SDF-Python</a></td>
</tr>
<tr>
<td>Mfiles+SDF</td>
<td><a href="https://github.com/ScientificDataFormat/SDF-MATLAB">https://github.com/ScientificDataFormat/SDF-MATLAB</a></td>
</tr>
</tbody>
</table>

### 3.4.3 Installation on Windows

For the full list of supported compilers, see “Software requirements” starting on page 89.

**Support for Visual Studio 2017 compiler**

Dymola 2019 supports the Visual Studio 2017 compiler, the following editions:

- Visual Studio 2017 Desktop Express **Note!** This compiler only supports compiling to Windows 32-bit executables.
- Visual Studio Community 2017
- Visual Studio Enterprise 2017
- Visual Studio Professional 2017

The recommended free compiler to use is the Community edition due to the limitation in the Desktop Express edition.

The compiler is selected in the simulation setup, reached by the command **Simulation > Setup...**, the **Compiler** tab:
**Simulation Setup**

- **C compiler**
  - Visual Studio 2005 Professional (8.0)
  - Visual Studio 2010/Visual C++ 2010 Express Edition (10.0)
  - Visual Studio Custom

- **Embedded server (Requires Visual Studio as C compiler)**
  - None
  - DOE server
  - OPC server (Requires administrator rights)

- **Export DLL**
  - Export model as DLL with API

- **Custom options**
  - Compiler
  - Linker

**Store in Model**

- **OK**
- **Cancel**
3.5 Other Simulation Environments

3.5.1 Dymola – Matlab interface

Compatibility

The Dymola – Simulink interface now supports Matlab releases from R2013a (ver. 8.1) up to R2017b (ver. 9.3). Only Visual Studio C++ compilers are supported to generate the DymolaBlock S-function. The LCC compiler is not supported.

3.5.2 Real-time simulation

Compatibility – dSPACE

Dymola 2019 officially supports the DS1005, DS1006, MicroLabBox, and SCALEXIO systems for HIL applications. For these systems, Dymola 2019 generated code has been verified for compatibility with the following combinations of dSPACE and Matlab releases:

- dSPACE Release 2013-A with Matlab R2013a
- dSPACE Release 2013-B with Matlab R2013b
- dSPACE Release 2014-A with Matlab R2014a
- dSPACE Release 2014-B with Matlab R2014b
- dSPACE Release 2015-A with Matlab R2015a
- dSPACE Release 2015-B with Matlab R2015b
- dSPACE Release 2016-A with Matlab R2016a
- dSPACE Release 2016-B with Matlab R2016b
- dSPACE Release 2017-A with Matlab R2016b and R2017a
- dSPACE Release 2017-B with Matlab R2016b, R2017a, and R2017b

The selection of supported dSPACE releases focuses on releases that introduce support for a new Matlab release and dSPACE releases that introduce a new version of a cross-compiler tool. In addition, Dymola always support the three latest dSPACE releases with the three latest Matlab releases. Although not officially supported, it is likely that other combinations should work as well.

Note on dym_rti_build and dSPACE Release 2017-A and later

The function rti_usrtrcmerge is no longer available in dSPACE Release 2017-A and later. As a consequence, it is required to run the standard rti_build function (with the ‘CM’ command) after dym_rti_build to get your _usr.trc content added to the main .trc file. For example:

```matlab
>> dym_rti_build('myModel', 'CM')
>> rti_build('myModel', 'Command', 'CM')
```
Compatibility – Simulink Real-Time (formerly xPC Target)

Compatibility with Simulink Real-Time has been verified for all Matlab releases that are supported by the Dymola – Simulink interface, which means R2013a (xPC Target ver. 5.4) to R2017b (Simulink Real-Time ver. 6.7). Only Microsoft Visual C compilers have been tested.

3.5.3 FMI Support in Dymola

Unless otherwise stated, features are available both for FMI version 1.0 and version 2.0.

General

Reorganization of simulation setup dialog for FMI

In previous versions of Dymola, the settings for FMU import and FMU export were available in the simulation setup, reached by e.g. the command Simulation > Setup…, in the FMI tab.

In Dymola 2019 this tab has been split into two tabs, FMI Export and FMI Import:

The FMI Export tab:
The **FMI Export** tab has been somewhat reorganized. This reorganization is also applied on the dialog that by default appears when exporting an FMU:
Note the new information message for FMU export if the setting **Evaluate parameters to reduce models (improves simulation speed)** is not ticked in the **Translation** tab of the simulation setup.

The **FMI Import** tab:
**FMU Export**

**Defining a user-specified model identifier of an FMU in GUI when exporting**

In earlier versions of Dymola you could yourself specify the model identifier of an FMU when exporting an FMU by using the input parameter `modelName` in the built-in function `translateModelFMU`.

In Dymola 2019, this setting is also available in the user dialog when exporting the FMU; you can type your own model identifier for the FMU in the new **Model identifier** field:
Note that there is a default value in the field, the name of the model to be exported. Please go back three images to compare with the same dialog where the default value (for the Coupled Clutches demo) is still present.

The model identifier will also be the file name of the exported FMU.

**Using sparse Jacobians when exporting FMUs with source code included**

FMU source code export includes code for building the CVode solver. With the new feature CVode can be enhanced with sparse solver capabilities. If the flag `Advanced.SparseActivate` is set to `true` before the export additional files are copied to the "sources" subdirectory of the generated FMU directory.

The details on how to enable the code for sparse solvers are found in the documentation provided in the FMU directory, in the `index.html` file, located in the generated FMU. (To display the directory, use any zip tool to reveal the structure of the generated FMU.) This description looks the following:
Enabling sparse solver

Additional source files are needed to enable sparse solution of linear systems during simulation. The required CVode and SUNDIALS files are located in the "source" subdirectory of the Dymola installation directory. If the flag

```
Advanced.SparseActivate = true
```

is enabled during FMU source code export these files are copied to the "sources" subdirectory of the FMU. To enable the code for the sparse solver define the preprocessor macro

```
#define FMU_SOURCE_CODE_EXPORT_SPARSE
```

Furthermore, the following three files must be compiled in addition to those listed above

```
cvode_sparse.c
cvode_superslu.c
sundials_sparse.c
```

For all.c these are incorporated automatically when setting `FMU_SOURCE_CODE_EXPORT_SPARSE`. These files implement the interface between CVode and the default sparse linear algebra library: SuperLU_MT Version 2.4. You must link with this library when building the executable. The source code can be found at [crd-legacy.lbl.gov/~xiaoye/SuperLU/](http://crd-legacy.lbl.gov/~xiaoye/SuperLU/).

(Note: You must also define `DYNSparseJacobian` if it is not already defined in `dsmodel.c`. Its value affects the number of cores that are used during sparse matrix factorization.)

Exporting a model in DAE mode

Exporting a model in DAE mode as an FMU is supported if the selected solver supports DAE mode. If this is not the case, the export is not allowed – an error message will be displayed. See “DAE Mode” on page 33 for details.

Controlling the result generation interval when storing result as a mat file

Previously the result generation interval when storing simulation result as a .mat file was arbitrary and depending on the solver. In Dymola 2019 a new setting `Interval` has been introduced as part of the setting `Store result in mat file`, in the `Store result` group in the `FMI Export` tab (see image above) It controls the result generation interval when storing simulation result as a .mat file. The default value of the setting is 0.001. A corresponding real flag is available, `Advanced.FMUSToreResultInterval`

Option to include “save total” of the model in the generated FMU

It is possible to include a “save total” of the model in the generated FMU by setting the flag
Advanced.FMI.IncludesSaveTotal = true

The default value of the flag is false.

More GUI settings available as flags for scripting

Looking at the FMI Export tab, the following marked settings are now also available as flags for scripting:

The flags are:

- **Advanced.FMI.FMUFMIType** This flag of type String controls the FMI type, the alternatives are (default is an empty string):
  - "me" Model exchange
  - "cs" Co-simulation using Cvode
  - "all" Model exchange, and Co-simulation using Cvode
  - "csSolver" Co-simulation using Dymola solvers

- **Advanced.FMI.FMUIncludeSource** This Boolean flag controls if source code should be included in the generated FMU. The default value is false.

- **Advanced.FMI.FMUStoreResult** This Boolean flag controls if the result should be stored as a .mat file. The default is false.
• **Advanced.FMI.FMUStoreResultInterval.** This flag of type Integer is used if the previous flag is set, to control the interval of result storage. This functionality is new, see “Controlling the result generation interval when storing result as a mat file” above.

**Updated simulation data for Cvode FMUs**

In previous Dymola versions, the simulation data was reset during events, and the f function evaluation was based on all calls to f instead of solver calls, for example, it also included calls to the function during Jacobian calculations. This is now corrected and explains differences in simulation data for Cvode FMUs generated by Dymola 2019 and generated by previous versions of Dymola.

**FMU Import**

**More GUI settings available as flags for scripting**

Looking at the **FMI Import** tab, the following marked settings are now also available as flags for scripting:

The flags are:

- **Advanced.FMI.Integrate** This Boolean flag has the alternatives:
  - *true* Model exchange [default]
  - *false* Co-simulation

- **Advanced.FMI.IncludeAllVariables** This Boolean flag controls if all variables should be included when importing the FMU. The default is *true*.

- **Advanced.FMI.PromptReplacement** This Boolean flag controls if user should be prompted when an imported FMU is replacing an existing Modelica model. The default is *false*.

**Using a standard FMU image when importing a FMU with no model image available**

When importing an FMU with no model image available, a standard FMU icon is now used when instantiating that FMU. An example of such an FMU instantiated:
FMI Kit for Simulink

A new version, 2.4.2, of the FMI Kit for Simulink is released and distributed with Dymola 2019. This version adds support for Matlab R2017a and R2017b.

In addition, the following improvements are included for the FMU export:

- Extended support for S-functions:
  - User-defined data types
  - Calls to ssSetStopRequested respected
  - ssPrintf directed to FMU logger also for C++ S-functions

- Indices of array variables in the XML description starting at 1 instead of 0 to comply with the FMI specifications

- Corrections to support the Visual Studio 2015 and 2017 compilers

3.5.4 Source Code Generation

Using sparse Jacobians

In Dymola 2019, using sparse Jacobians during source code generation is supported. The example project StandAloneDymosim now also includes a demo exemplifying how the information provided by dsmodel.c can be used to build sparse numeric model Jacobians. These Jacobians can then be used by sparse solvers for efficient simulation of large-scale models (not demonstrated here). Files implementing the demo:

- sparse_Jacobian.h, sparse_Jacobian.c: Exemplify how the structural sparsity pattern provided by dsmodel.c can be used to construct sparse numeric Jacobians.

To enable the demo include sparse_Jacobian.c in the project and define the flag DYN_SPARSE_JACOBIAN_TEST (as well as INCLUDE_EULER). When activated a call is
made in StandAloneDymosim.c to construct a sparse numeric Jacobian at the end of the explicit Euler simulation. The Jacobian is not used by explicit Euler, but the result of the simulation is used to exemplify sparse Jacobian construction.

### 3.6 Modelica Standard Library and Modelica Language Specification

The current version of the Modelica Standard Library is version 3.2.2. The current version of the Modelica Language Specification is 3.4.

Note that the Modelica Standard Library is compliant with the Modelica Language Specification 3.4. (So this version of the Modelica Language Specification should be used for reference, and not the Modelica Language Specification 3.2, Revision 2.)

### 3.7 New libraries

Below is a short description of new libraries. For a full description, please refer to the libraries documentation.

The libraries are presented in alphabetical order. If not stated as free, the library is commercial.

#### 3.7.1 ClaRa DCS Library

The ClaRa DCS Library is an add-on to the ClaRa Plus library, for handling Distributed Control Systems. The library provides state-of-the-art power plant controllers to couple them to power plant models created with ClaRaPlus.

This library provides (stopable) controllers, logical and nonlinear blocks very close to the hardware implementation of modern Decentralized Control Systems of power plants. Thus, the user is able to set up the power plant control of his model closely according to the control documentation. This reduces modeling effort, avoids mistakes and enables failure analysis and detailed tuning of the control system.

The library allows to:
- Directly convert existing control documentation to ClaRa DCS applications
- Implement new control concepts
- Use application diagrams as documentation being close to DCS suppliers nomenclature
- Switch over from controlled mode to manual operation mode
- Investigate start-up and shut down sequences
- Detect effects of DCS failures to the process (e.g. due to sensor failures)
• Detect effects of process failures to the DCS (e.g. high pressure/temperature above sensor limits of validity)
• Be coupled to Modelica Standard Library blocks

3.7.2 Electric Power Systems Library

The Electric Power Systems Library (EPSL) is intended to assist the process of designing electrical power systems. It enables modeling Alternate Current (AC) and Direct Current (DC) as well as combined AC/DC (hybrid) networks found e.g. in More Electrical Aircrafts. Due to the use of a phasor based description, it enables fast simulation of AC systems, with performance basically independent of the AC system's frequency.

The EPSL covers the key components of an electrical power system at different levels of detail. The architectural mode corresponds to "simple" phasor theory resulting in a quasi-static description of the system. The functional level of detail uses dynamic phasors to cover dynamic effects in the models. The modeling mode can currently be switched on a per-component basis via a parameter.

Additionally the EPSL enables the description of harmonics of the fundamental AC frequency, which can be either constant or variable. Currently the AC components are either single or three-phased, whereas other phase numbers would be straight-forward to be added due to the generic modeling approach of components.

3.7.3 Pneumatic Systems Library

The new Pneumatic Systems Library (PSL) is intended to assist the development of pneumatics systems. PSL enables predicting the system behavior for a wide range of industrial applications such as industrial machines, pneumatic brakes or suspension systems as well as cooling and engine bleed air systems for aerospace.

In this first release, the following components are introduced:
• Gases: IdealAir model and ImportFromMSL so any media model extending from Modelica.Media.Interfaces.PartialMedium can be used for gas properties.
• Sources: ideal pressure, mass flow rate and exhaust, silencer and a vane compressor.
• Valves: directional, proportional and flow control valves.
• Actuators: linear, rotary and bellow drive.
• Reservoir.
• Piping: tube, bend and a custom pressure loss component.
• Sensors.

The library also includes a package modeling the main physical effects modeled: capacitance (energy storage), resistance (energy dissipation) and power transformation (mechanical and thermal). These effects serve as a basis for the development of the previously listed components.

The component icons are compliant with the standard ISO 1219 which is typically used for pneumatic schematics. Animation is provided to see the valve and cylinder displacement with respect to their command.
As an example, the following model represents two single-acting cylinders with spring feedback, piloted by a single directional control valve alternatively connecting to an ideal pressure source or exhaust. While the top cylinder does not include any cushioning device, the piston of the bottom cylinder will close the inlet port when it approaches its end stop (throttle effect).

This example covers the main physical effects involved into pneumatics: capacitance as the gas volume inside the cylinder, restriction as the inlet flow restriction into the cylinder and transformation as the cylinder converts pneumatic into mechanic power.

As the example uses a one-sided cylinder, cushioning only happens on the left side of the cylinder actuation. When the piston enters in the cushioning zone (below the green line), the cylinder with cushioning (red plot) receives less mass flow rate from the inlet port and thus its speed decreases regarding the top cylinder (blue plot).
3.8 Updated libraries

Below is a short description of updated libraries. For a full description, please refer to the libraries documentation.

3.8.1 Battery Library

A new version 2.1 has been released. Some examples of new features and improvements:

Improved cylindric cell thermal model

The surface heat port of the cylindric heat port can now be discretized along the perimeter of the cell. The surface heat port now has the size \([N_{\text{surface}}, N_{\text{verticalElements}}]\) where \(N_{\text{surface}}\) defines the number of heat ports on the perimeter and \(N_{\text{verticalElements}}\) defines the number of ports in height (\(z\)-direction). This change allows ideal heat transport in the pack models using cylindrical cells.
**New nested pack model**

A pack model of nested cylindrical cells is now available in the library. The pack includes 4 different nesting types. The pack models are structured into three replaceable submodels:

- **CellConfiguration**: Defines the used cell model and the electric and thermal connection of the cell pins.
- **ThermalSeparation**: Defines the heat transfer between the cells. Models with ideal, static and dynamic heat transfer and direct and material based parameterization are available.
- **Housing**: Defines the thermal properties of the housing. Models with static and dynamic heat transfer and direct and material based parameterization are available.

**Thermal boundaries for pack models**

Thermal boundaries models that enable a quick setup for test models are now available.

**Other features and improvements**

- The assertion warnings raised by the models of the library are improved.
- Copper is added to the material records.
- Variables for core temperatures and heat flows for each thermal connector are introduced in the partial thermal models.

### 3.8.2 ClaRa Grid Library

A new version 1.1.0 has been released. This version contains minor updates of examples due to the new version of ClaRa Plus Library.

### 3.8.3 ClaRa Plus Library

A new version 1.1.0 has been released. Some examples of new features and improvements:

#### New Features

**General**

- FluidDissipation is omitted. All applications of the library are transferred directly to the replaceable models. This is advantageous with respect to CPU time and robustness.
- New fuel connectors which are less error prone and more flexible.
- ClaRaPlus.Components.Furnace now has optional pressure loss models.
- New Examples which are more declarative, better represent the power of ClaRa and describe relevant transient scenarios

**Components**

- Improved numerical behavior of all NTU based models, including heat exchangers
- Major pump revision
New Models

- ClaRaPlus.Components.Electrical.AsynchronousMotor_L2_VFD - a motor which can be speed controlled and can be attached to ClaRa_Grid based electrical grids.
- ClaRaPlus.Basics.ControlVolumes.FluidVolumes.VolumeVLE_L4 was improved to account optionally for (slight) mechanical non-equilibrium of liquid and gas phase.
- Fuel Sensors and fuel splits.
- Static Cycle Dispatcher for calculation of required fuel mass flow rate.

3.8.4 Cooling Library

A new version 1.2.1 has been released. Version 1.2.1 contains minor additions and improvements.

Examples of additions/improvements are, for the individual sub-packages:

- Cooling.Common: Add summary record for tank models
- Cooling.Media: Improved code inlining for incompressible media (DassaultSystemes library)
- Cooling.Sources: Improved graphical representation of vectorized ports
- Cooling.Pipes: Fix parameter propagation to wall model
- Cooling.Reservoirs: Add specific summary record to OpenLiquidTank

3.8.5 Dymola Commands Library

A new version 1.5 has been released. The updates reflect the additions and changes of built-in functions in, for example, “Scripting” starting on page 40.

3.8.6 Electrified Powertrains Library

A new version 1.2.1 has been released. The release is a maintenance release, containing, for example, the following additions and improvements:

Appearance

- New library icon
- New colors in documentation

Functionality

- Improved default value for outputfilename in Rainflow counting
- Removed duplicated values in Battery Data, functionality is covered by the "hold" extrapolation
- Deactivated REX in BEV example
3.8.7 Fluid Dynamics Library
A new version 2.5.0 has been released. Two new items have been added:

3.8.8 Fluid Power Library
A new version 2018.1 has been released.
In this release the concepts from FluidPower and from Claytex.Fluid have been unified in a new package ClaytexFluid, on which FluidPower now depends.
It is possible to utilise components from ClaytexFluid together with components from FluidPower. In order to do this the medium in the ClaytexFluid component must be set to the media model FluidPower.Fluids.HydraulicOil. This will connect the ClaytexFluid component to the FluidPower fluid representation. An example of this can be seen in the FluidPower.Piping.Pipes.CurvedBend pipe model.

3.8.9 FTire Interface Library
A new version 1.0.1 has been released. The release is a maintenance release focusing on consistent layout.

3.8.10 Human Comfort Library
A new version 2.5.0 has been released. The following new models have been added:

3.8.11 HVAC (Heating, Ventilation, and Air Conditioning) Library
A new version 2.5.0 has been released. New adaptor models for coupling HVAC liquid side models with models from the Modelica.Fluid library have been introduced.

3.8.12 Hydrogen Library
A new version 1.1 has been released. The main new features are:
- Control of moisture of media: humidifier and water separator
- Three new media models: two for moist hydrogen and methane reformate gas
- Control based Methane reformation model for PEM
- Heat exchanger
- Generic volume and pressure loss for easy addition of user specific components

3.8.13 Optimization Library

A new version 2.2.3 has been released. An output value of the optimization runs to indicate if the run has been canceled has been introduced.

3.8.14 Testing Library

A new version 1.1 has been released. This version brings improvements to the test runners, changes the visual look of the library and introduces operator records.

Main features

The main features are:

- The test runner runTests has been improved:
  - Simulation output settings are considered for simulation if they are stored in the annotation of a test model, this is necessary because the setting equidistant time grid can influence the simulation.
  - Instead of packages single test class can be selected also
  - The required time for the execution of a test is printed
  - Test models with setup.mode set on skip_checks are not simulated
- Pedantic check can be turned on in checkPackage, previously the current Dymola setting was used.
- Visual update: the library icon has been updated and the colors of all blocks in the library have been changed to match the new library icon.
- All records were converted to operator records; they contain the corresponding functions now.
- Absolute and relative tolerances can be set in Checks.ArrayReal
- When references are created with createRefs, it is now possible to specify the simulations settings instead of using the settings stored in models

Upgrading from version 1.0

Tests which have been created with version 1.0 can be converted to version 1.1 with the provided conversion script. As most changes are related to the Utilities package, conventional test models are not affected and the conversion script will not change anything.

Only if functions from the Utilities package are used the code will be changed when upgrading.
3.8.15 Thermal Systems Library

A new version 1.2.0 has been released. Some new features are:

- Numerical robustness of almost all valves (gas, liquid, VLEFluid) improved.
- Set-point weighting available in PI-controller, which decouples set-point and feedback acting.
- New blocks in Utilitis->Numerics: SmoothSwitch and DiscreteSignalSmother can be used to get smooth transitions between signals.

3.8.16 VeSyMA (Vehicle Systems Modeling and Analysis) Library

A new version 2018.1 of the library has been released.

New features and Major Changes

Road and Driver Models

Drivers now have the ability to stop at pre-defined distances when using the table based roads which can now include stopping points, either defined in the .mat file or in the road model. This is implemented through an extra function added to the base road interface. This function requires the position and stop number and outputs the position of the next stopping point sStop and stop time tStop, i.e. the time to be stationary for. The road generation functions have been extended to include generation of the stopping table and activation flags. The roads also include either repeating stop or table input that can override the stopping points read in from the .mat file. In road models that don't use a .mat file the table based and repeating stops are still available.

The driver model has required modification to implement the stopping points. The driver planning has been modified that can modify the speed target if a stopping event is ahead. This modifies all the speed target outputs of the planning block for the longitudinal target to attain.

Road models, and the wheel contact or sensor models that use them, can now include the concept of a wheel identifier. This is an Integer value that allows the model to uniquely identify each wheel during simulation in order to optimize performance. Functions that determine the identifier must extend from registerWheel, and can use the name of the component and/or the initial location of the component to identify the wheel. The registerWheel function is called in the GroundCoordinates block for sensors, and the CentralPlane contact model for wheel contacts. The default output from registerWheel is -1, defined by the constant NoWheelIdentifier - road models that use wheel identifiers may also need to take account of situations where no identifier has been determined.

The structure of the road models has also been revised to modularize the code. Several classes are now extended to create the different road models with standard templates for the friction, cones and stop definitions.

A new road block has been added that can determine the lap number for closed circuit roads. This block has been included into the "senses" block within the driver models.
Trailers and Towing

Support for towing tests has been added to the library. There are now templates and examples for trailers and an optional hitch frame and toHitch translation in the BaseCar template. The trailer models have their own templates and interfaces, allowing replaceability of trailer models using the same interface. The hitch connection type is contained within the trailer model, requiring no modification to the vehicle to change the hitch type. The hitch options available include ball and socket, fifth wheel and a rigid connection.
Other Changes

- Corrections to the Claytex Mounting1D changes the reaction torque sign of many models including the brakes, engine and transmission. This corrects the reaction torque direction in the case of the engine and transmission, but required changes to the brake torque Reaction axis to change from \{-1,0,0\} to \{1,0,0\}. This can effect the roll and pitching of the chassis due to the reaction torque of the powertrain.

- Changes to the road radialFromTable function to make output, e_w_0, normalised. Causes slight changes to lateral road position during simulations.

- Correction to the dynamic battery due to change to the SOCCalculator which has changed the input to the cellVoltage table from % discharge of the battery to State of change of the battery. This is to make variables more consistent and easier to query.

- Change to the Driveline base class has added a sourceSign variable that is dependent on the orientation of the source of rotation that can control the sign of the final drive ratio, maintaining the correct sign out. This removes the requirement for the Rear (RI) and Front Input (FI) nomenclature as the sign can be controlled from the top level of the model. Modifications have been made to the vehicle templates to incorporate the automatic modification of the sourceOrientation to make the sign control more automatic.
• Updated the brake experiment templates InertiaTest and TableTorqueTest to include a World, replaceable atmosphere and total mass block to remove translation warnings. Examples extended from these templates now use the Vehicle Interfaces constant atmosphere.

• The brake disc model now extends from a new brake disc interface; the brake disc thermal model also now extends from a new thermal model interface. The new brake disc interface is now used in the FourWheelBrakes template. The brake examples have been tidied to reflect this change.

• The driveshaft compliance in the driveline template is now a class parameter instead of a type.

Conversion of user models

Conversion scripts are used to automatically update user's models built using previous versions of the library to be compatible with this version. Any manual updates required will be described above. Users should check their models following an update to the libraries used.

3.8.17 VeSyMA – Engines Library

A new version 2018.1 has been released, with a number of new features and improvements:

New Features

Turbochargers

• An excel-based tool has been created to calculate the coefficients for the compressor and turbine in the equation-based turbocharger model. Available at the location where the Claytex libraries are installed in: Engines/Maps/Tools.

• New turbocharger model that can read directly manufacturer data. Its inputs are in the form of pressure ratio, corrected/ non dimensional mass flow rate and efficiency for several corrected/ non dimensional speeds.

Summary

Added a summary record

After treatment

• DOC model: Diesel Oxidation Catalyst model used to reduce the amount of HC and CO going down the exhaust. A test is available for this model.

• ASC model: Ammonia Slip Catalyst. This is the last after treatment device, it gets rid of the remaining ammonia injected upstream and that did not react. A test for it is available.

• DPF and DOF controller models: Diesel Particulate Filter and its controller. Used to reduce the amount of soot. An experiment is available to test it.

• An engine experiment has been created that integrates the entire diesel after treatment devices. All these models need to use the medium EngineAir to work correctly.
IMEP and BMEP calculation

- IMEP and BMEP (with regards to pumping losses) are calculated within the cylinder models.

Knock detection

- An empirically based knock model is included for both Wiebe and predictive combustion models. It correlates the Arrhenius function to measured data on induction or autoignition times, for given fuel-air mixtures, over the relevant mixture pressure and temperature ranges. The specific correlation used is that proposed by Douaud and Eyzt. A precursor, describing the evolution of auto-ignition delay is used together with a fixed criteria for critical autoignition boundary to determine whether autoignition occur before the normally propagating flame consumes the end gas. There are four categories to quantify knock intensity, which are no knock, trace knock, medium knock and strong knock.

Dyno

- A new rig controller has been created that controls the accelerator pedal position in order to follow a speed profile.
- The speed demand in the dyno rig controller can now be filtered to avoid the PID generating torque spikes at changes of speed derivative.

Modifications

Engine top level

- New torque and power sensor added. Both quantities can be filtered as required, particularly for CAREM applications. Replaces the old experiment top level torqueFilter flag-activated sensor.
- Friction model using 2D table has been set as obsolete and will be removed in future version. Tabular friction models can make use of Engines.EngineFriction.Components.TableBased3d.

Fuel injectors

- Revised injector hierarchy to improve user friendliness and reduce duplicate code
After treatment
- Improvement to the catalytic converter. The structure of the model has been modified to match the one of the other after treatment devices.
- Extra functionalities added to the SCR: more chemical reactions are taken into account, including those to form ammonia from the injected urea.

Combustion
- Mean value combustion models have been rationalized to only keep a few. Some were redundant. Their names have also changed to make it clearer and to match those of the crank angle resolved combustion models.
- Rationalization of the injector models to reduce levels of hierarchy and variants.

Air path
- Idle control valve removed. Idle is now controlled with torque by default
- Throttle body has been enhanced to account for effective area calculation by removing the projected area of the butterfly spindle
- Implementation of valveless duct in place of throttle body for Diesel applications that require no throttling
- Injector models based on Engines.FuellingSystems.Components.FuelInjectors.Pfi_p in Engines 2017.3 where the valve diameter is parameterised within the valve will have to be changed to reference injHoleDia in the new model Engines.FuellingSystems.Components.FuelInjectors.DI_pressure. Fuel rail dynamics investigations are more commonly associated with CAREM DI engines so the injector has now been named DI_pressure.

Controllers
- Improvements in units descriptions required for data in controllers
- Diesel fuelling control improved and it torque demand based and engine speed based.

Pistons
- Piston diameters have been reduced to take into account the piston/liner clearance which has been corrected to a value of 0.05mm
- The blow-by model has been modified to eliminate a nonlinear system and numerical Jacobian.

Conversion of user libraries
Automatic conversion of user libraries from previous versions is supported by included conversion script. Note that IgnitionFuelingModuleCarem from the last release 2017.3 is changed to SparkIgnitionFuelingModuleCarem in this release. Models using IgnitionFuelingModuleCarem in compression Ignition engine should switch to CompressionIgnitionFuelingModuleCarem.

Users should check their models following an update to the libraries used.
3.8.18 VeSyMA – Powertrain Library

A new version 2018.1 has been released.

New features

- A seat model has been added to the library. It can be used to study, for example, the head rest acceleration. An example of how to use it in a vehicle is available.
- A temperature dependent mesh loss model has been added (and the existing power dependant model has been improved).
- A function has been added to the library that, when run, creates a reduced version (first image below) of a multibody gearset (second image below). See the documentation of this function for more information on how to use it.
Here are the steps that the function goes through:

1. Run the multibody gear set over all its operating points (using ramps for the speed and load inputs) in all the gears
2. Collect the results in terms of inertia and losses (speed dependent and torque dependent, for both driving and driven conditions)
3. Create a set of data records to store this information
4. Extend and parameterize a reduced (1D, table based) version of the gearset with the data we collected
5. Save all the models and data records created by the function in a new package and open it in Dymola

Here is what the package created by the function looks like:
Other changes

To improve the usability a number of changes have been made to the driveline and shaft models to make it easier to parameterize these models. The Characteristic parameter has been removed and instead the shafts and other relevant components have a replaceable complianceModel. The complianceModel is selected through the pull-down menu on the components parameter dialog button and then you click the Edit button next to the drop-down to access the parameters for the selected model. To make sure the complianceModel and its parameters are visible when using show component, you also need to click on the little black arrow and select the replaceable option.

Conversion of user models

Conversion scripts are used to automatically update user's models built using previous versions of the library to be compatible with this version. Any manual updates required will be described above. Users should check their models following an update to the libraries used.

3.8.19 VeSyMA – Suspensions Library

A new version 2018.1 has been released.

New Features and Major Changes

Build Road Models from Logged GPS Data

A new road building function has been added, RoadFromLoggedData. This function enables the conversion of recorded/logged GPS data to be used to generate a road model for use in Dymola. It does this by converting the latitude, longitude and altitude values as recorded by the GPS sensor into 3D data points, using the first value in the data file as the origin. These
3D data points are then supplied to the `Suspensions.Roads.RoadBuildingFunctions.RoadFromCentreLine` function to generate a road file. The road banking angle/camber angle experienced by the vehicle logging the data is calculated from the supplied body accelerations in x, y and z. Optional output of the 3D data points and banking angle in a CSV format file is also supported. If creating a closed circuit using the new `RoadFromLoggedData` function, then some pre-processing of the input data is required to produce a good match at the start/end of the data, this is described in the documentation of the function.

### Wheel Rig

There is now a Boolean called `zActuationPosition` which controls whether the rig is driven in z by force or position. To facilitate this, the base controller has an added comparable Boolean which controls the declaration of an output to be used to supply a z position to the rig. A displacement sweep example test has been included demonstrating how such an experiment can be undertaken. The controller created for this test is open loop, but more sophisticated controllers can be built for displacement sweeping due to the modification of the base controller.
Vehicle Test Rigs

Multiple changes have been made to the vehicle test rigs:

- An eventless noise generation block has been created, ExpSineNoise, allowing users to define a non stepped road noise from a randomly phased number of sinusoidal frequencies and related amplitudes. The relationship between the amplitudes and frequencies is defined by an exponential relationship allowing users to define the exponent and coefficient to match road types.

- The Swept sine input has been modified to be more consistent throughout the wave; the amplitude/frequency relationship is now governed by the peak velocity relationship, which can either be calculated from initial amplitude or can be user input.

- Hub and Contact point forces have been added to both MultiVerticalAcuator and MultiActuator rigs, allowing users to apply forces and torques at both the hubFlange or at the contact point, without use of tyres with inputs. Activated through the cPForce and hubForces Booleans in each rig, they are controlled through the rig controller but no modifications need to be made to current controllers.

Roads

- The number of road models has been reduced from 5 to 3 adding functionality to the road functions, allowing table based functions to use a non table based alternative which use flat surface, straight direction or constant values. These changes allow the IndependentDrivingLineFlat to be made obsolete as the same functionality can be obtained using the IndependentDrivingLine and activating flatRoad=true. The FTire roads have also been reorganized, with a single FTire road extending from IndependentDrivingLine, therefore retaining the same flat or straight options and reducing duplication. FTireStraight has been made obsolete as the straightRoad option allows the driving line to be straight, rather than read from the .mat file.

- The NPost road model, which matches the VeSyMA Road interface but defines dynamic positions of a number of wheel pads in order to model a test rig, has improved performance. At the start of simulation, each wheel or sensor calls the function registerClosestWheel. This determines an identifier for the pad of the rig that is closest in the X-Y plane, or -1 if there is no local pad. During simulation this identifier is used in the nPostGroundCoordinates function to provide the coordinates of the closest pad without calculating it at each timestep.

- The roughness in the road model has been reviewed and developed, still using the ISO 8608 definition, allowing a higher amount of control over the range of frequencies used. This allows users to define the range of frequencies that are of interest, which allows the ability to focus on either high or lower frequencies independently. These changes have been made primarily in the roughness Functions and Interfaces. Added functionality also includes an altered method of frequency selection, which can reduce lower frequencies creating dominant oscillations; this is done by using an exponentially controlled stepping method, which can choose more frequencies from the lower end of the range. The experiments within the library that use the rough road models, RoughRoadDoubleLaneChangeTest, RoughRoadWithCurbsTest and SlowRoughRoadGridContact, have been modified due to the changes to the rough road to now demonstrate higher frequency roughness.
Multiple Point Contact Models

Following the addition of the higher frequency road roughness added to the road models in 2017.3 the addition of multiple point contact models have been added to the library to reduce the tyre contact to a single point required for the Pacejka tyre models. Both the position and road normal are generated by the contact patch.

Two types of the contact patch have been added to the Suspensions library with 2 different patterns:

- The Grid based contact patch has n number of longitudinal rows and n number of lateral columns of points projected down onto the road surface to create a "Grid" of the rough road surface. The position of the contact patch is generated by a weighted average of each point depending on the penetration depth, if the penetration of a point is zero then it is not used. The normals are generated by using the cross product of vectors between adjacent points; the road normal is a weighted average of all normals in using same methodology of ground position.

- The 5 point contact model use much of the same theory with the position of ground and normal but uses a constant weighting that does not depend on the penetration of each point. The layout has a single central point with 4 points; 2 along the central axis and 2 perpendicular.

OpenCRG roads

The OpenCRG implementation has been updated to use v1.1.1 which was released by OpenCRG in July 2017. A number of extensions to their code have been incorporated to gain access to the partial derivatives of the surface positions. Support has also been added for a variable driving line and speed profile. Both of these are functions of the distance along the centre line of the road and can be generated using the road building function OpenCRGRoad or entered directly onto the road model but this is only recommended for small tables.

Trailers

The vehicle templates in VeSyMA now include provision for towing a trailer by enabling the hitch frame. By default this is disabled in all the vehicle examples. An example single axle trailer with trailing arm suspension has been added to this library. This example is used in the experiment SlalomTestTrailer.

Active Rear Steering

An example vehicle model equipped with an example of an active rear steering system and control algorithm has been added to the library. The CurvedAndSlopedRoadRearSteer experiment demonstrates the function of active rear steering.

Other Changes

- Steering angle and torque channels in the half car summary records are now multiplied by -1 so the directionality is ISO compliant.

- Corrections to the Claytex Mounting1D changes the reaction torque sign of many models including the brakes, engine and transmission. This corrects the reaction
torque direction in the case of the engine and transmission, but required changes to the brake torqueReaction axis to change from \{-1,0,0\} to \{1,0,0\}. This can effect the roll and pitching of the chassis due to the reaction torque of the powertrain.

- By default the Boolean leftWheel has been set to false in the sensor packs utilised in the Suspensions.QuarterCar.Experiments.Examples.FrontMacPhersonKinematics and Suspensions.QuarterCar.Experiments.Examples.FrontDoubleWishboneKinematics example experiments so the calculated camber and toe values have the correct sign.

- CamberRelativeVehicle sensor made obsolete in favor for the Camber sensor (renamed from camberRelativeGround) as camberRelativeVehicle produces camber value measured in the wrong plane.

- The calculation of the protected parameter nAxis in DamperStrutIdeal has been updated to use the absolute value of the strut axis normal for the equality comparison.

- The parameter ratio was added to the steering linkage interface and used in the steering linkage examples. This parameter was then propagated from the linkage in the steering template, so the steering ratio is now set at the top level of the steering model rather than the steering linkage level.

- Corrections to PitmanArmIdeal steering linkage model as the connections and parameterization of the Drag Link and fixed translations to the steering frames were wrong.

- Corrected the default centre of mass position of the Drag Link component in the PitmanArm steering linkage.

- The ESP brake model PseudoHydraulicSingleCircuit has been updated to extend from the VeSyMA brakes template FourWheelBrakes rather than the brake system interface to reduce duplication. New tests of this brake model were added to demonstrate the ESP behaviour.

- Mass origin parameter propagated in Link, Trackrod, LinkUSAgregated, LinkSUAggregated, ControlArm and ControlArm2.

- A number of corrections have been made to the power steering models:
  - The values in the first column of the ForceApp lookup table were changed in the power steering examples translational TransferQuasiStatic, translational QuasiStaticSpringDamper, rotational TransferQuasiStatic and rotational TransferQuasiStaticSpringDamper. The values were reduced in magnitude to match the output from the quasi static filter.
  - The values in the ForceApp lookup table were changed in the power steering example translational TransferSpringDamper. The values are now suitable for a relative angle input. The same table values are also used in the linkages example IdealRackAndPinionWithFrictionTranslationsPowerSteering which uses this power steering example.
  - The values in the ForceApp lookup table were changed in the power steering example translational PseudoHydraulic. The values are now suitable for use with the pseudo hydraulic block.
Updated the powerSteeringLookup table in the linkage example IdealRackAndPinionWithFrictionColumnPowerSteering to be the same as that in the power steering model used. The propagated parameters in the power steering have been finalized to protect the link.

In the steering linkage PitmanArmPowerSteering propagated finalized the power steering gain and forceApp lookup table for consistency with other linkage models.

Conversion of user models

Conversion scripts are used to automatically update user's models built using previous versions of the library to be compatible with this version. Any manual updates required will be described above. Users should check their models following an update to the libraries used.

3.9 Documentation

General

In the software distribution of Dymola 2019 Dymola User Manuals of version “March 2018” will be present; these manuals include all relevant features/improvements of Dymola 2019 presented in the Release Notes.

New document: Dymola Referential

A new document is available, summarizing benefits and technical overview information for Dymola, Modelica, and FMI: Dymola Referential. Running Dymola, the document can be found, like the other Dymola documents, by the command Help > Documentation, or using the shortcut F1.
3.10  Appendix – Installation: Hardware and Software Requirements

Below the current hardware and software requirements for Dymola 2019 are listed.

3.10.1  Hardware requirements/recommendations

**Hardware requirements**
- At least 1 GB RAM
- At least 400 MB disc space

**Hardware recommendations**
At present, it is recommended to have a system with an Intel Core 2 Duo processor or better, with at least 2 MB of L2 cache. Memory speed and cache size are key parameters to achieve maximum simulation performance.

A dual processor will be enough if not using multi-core support; the simulation itself, by default, uses only one execution thread so there is no need for a “quad” processor. If using multi-core support, you might want to use more processors/cores.

Memory size may be significant for translating big models and plotting large result files, but the simulation itself does not require so much memory. Recommended memory size is 6 GB of RAM.

3.10.2  Software requirements

**Microsoft Windows**

**Dymola versions on Windows and Windows operating systems versions**
Dymola 2019 is supported, as 64-bit application, on Microsoft Windows 7, Windows 8.1, and Windows 10. Since Dymola does not use any features supported only by specific editions of Windows (“Home”, “Professional”, “Enterprise” etc.), all such editions are supported if the main version is supported.

**Compilers**

*Please note* that for the Windows platform, a Microsoft C/C++ compiler, an Intel compiler, or a GCC compiler, must be installed separately. The following compilers are supported for Dymola 2019 on Windows:

**Microsoft C/C++ compilers, free editions:**
- Visual Studio 2008 Express Edition (9.0)
- Visual C++ 2010 Express (10.0)
- Visual Studio 2012 Express Edition (11.0)
• Visual Studio 2013 Express Edition for Windows Desktop (12.0)
• Visual Studio 2015 Express Edition for Windows Desktop (14.0)
• Visual Studio 2017 Desktop Express (15) **Note!** This compiler only supports compiling to Windows 32-bit executables.
• Visual Studio Community 2017 (15)

**Microsoft C/C++ compilers, professional editions:**
• Visual Studio 2005 (8.0)
• Visual Studio 2008 (9.0)
• Visual Studio 2010 (10.0)
• Visual Studio 2012 (11.0)
• Visual Studio 2013 (12.0)
• Visual Studio 2015 (14.0)
• Visual Studio Professional 2017 (15)
• Visual Studio Enterprise 2017 (15)

**Intel compilers**
The Intel compilers Intel Parallel Studio XE 2016 and XE 2017 are supported.

**Note** that you must also select a Visual Studio compiler when selecting any Intel compiler.

Current limitations:
• Embedded server (DDE or OPC) is not supported.
• Export DLL is not supported.

**GCC compilers**
Dymola 2019 has limited support for the MinGW GCC compiler. The following versions have been tested:
• For 32-bit GCC: version 4.8.1 and 5.3.
• For 64-bit GCC: version 4.9.2 and 5.3.

Hence, at least the versions in that range should work fine.

To download any of these free compilers, please visit [http://www.Dymola.com/compiler](http://www.Dymola.com/compiler) where the latest links to downloading the compilers are available. Needed add-ons during installation etc are also specified here. Note that you need administrator rights to install the compiler.

Current limitations with 32-bit and 64-bit GCC:
• Embedded servers (DDE or OPC servers) are not supported.
• Support for external library resources is implemented, but requires that the resources support GCC, which is not always the case.
• No support for runtime concept\(^1\).

• For 32-bit simulation, parallelization (multi-core) is currently not supported for any of
the following algorithms: RadauIIa, Esdirk23a, Esdirk34a, Esdirk45a, and Sdirk34hw.

**Dymola license server**

For a Dymola license server on Windows, all files needed to set up and run a Dymola license server on Windows, except the license file, are available in the Dymola distribution.
(This includes also the license daemon, where Dymola presently supports FLEXnet Publisher version 11.14. This version is part of the Dymola distribution.)

**Linux**

**Supported Linux versions and compilers**

Dymola 2019 runs on openSUSE 42.1, 64-bit, with gcc version 4.8.5, and compatible systems (see [http://doc.qt.io/qt-5/supported-platforms.html](http://doc.qt.io/qt-5/supported-platforms.html)). Any later version of gcc is typically compatible. In addition to gcc, the model C code generated by Dymola can also be compiled by clang.

To change compiler, change the variable `CC` in `/opt/dymola--<version>–x86-64/insert/dsbuild.sh`. As an example, for a 64-bit Dymola 2019 application:

```
/opt/dymola-2019-x86_64/insert/dsbuild.sh
```

Dymola 2019 is supported as a 64-bit application on Linux.

**Notes**

• 32-bit compilation for simulation might require explicit installation of 32-bit libc. E.g. on Ubuntu: `sudo apt-get install g++-multilib libc6-dev-i386`

• Dymola is built with Qt 5.9.0 and thereby inherits the system requirements from Qt. Several xcb helper libraries are bundled with Qt (in detail, QT was built with the flag –qt-xcb) in order to reduce the system dependencies as much as possible. However, some libraries might still require explicit installation:
  o libglu1-mesa
  o libpng12-0

• For rendering of jpg files, libjpeg62 must be installed.

**Note on libraries**

• Please note that you have to use the Optimization library version 2.x or higher to use multi-criteria design optimization on Linux; the older Design.Optimization package does not support multi-criteria design optimization on Linux.

• The library UserInteraction is not supported on Linux.

\(^1\) The Dymola runtime concept means that a Dymola model developed by a user that lack export options still can be executed on a different computer with a Dymola license (without an installed Dymola application) by pointing to that license by an environment variable.
Dymola license server

For a Dymola license server on Linux, all files needed to set up and run a Dymola license server on Linux, except the license file, are available in the Dymola distribution. (This also includes the license daemon, where Dymola presently supports FLEXnet Publisher 11.14.)