Dymola

Dynamic Modeling Laboratory

Dymola Release Notes

Dymola 2026x Refresh 1

The information in this document is subject to change without notice.

Document version: 1

Dassault Systèmes AB
Ideon Gateway
Scheelevägen 27 – Floor 9
SE-223 63 Lund
Sweden

Support: https://www.3ds.com/support
URL: https://www.dymola.com/
Phone: +46 46 270 67 00

Contents

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 2026x Refresh 1:

Microsoft Visual Studio C++

This is the recommended compiler for professional users. Both free and full compiler versions are supported. Refer to section "Compilers" on page 91 for more information. **Notes**:

- From Dymola 2024 Refresh 1, Visual Studio C++ compilers older than version 2015 are no longer supported:
    - From Dymola 2024x Refresh 1, Visual Studio 2012 is not supported anymore.
    - From Dymola 2022x, Visual Studio 2013 is not supported anymore. (Visual Studio 2012 was however still supported until Dymola 2024x, due to the logistics of changing the oldest supported version)

Intel

Important. The support for Intel compilers is discontinued from the previous Dymola 2022 release.

MinGW GCC

Dymola 2026x Refresh 1 has limited support for the 64-bit MinGW GCC compiler. For more information about MinGW GCC, see section "Compilers" on page 91, the section about MinGW GCC compiler. (Note that the support for MinGW GCC 32-bit compiler is removed from this Dymola version, Dymola 2026x Refresh 1.)

WSL GCC (Linux cross-compiler)

Dymola 2026x Refresh 1 has support for the WSL (Windows Subsystem for Linux) GCC compiler, 64-bit. For more information about WLS GCC, see section "Compilers" on page 91, the section about WSL GCC compiler.

Clang compiler

If you first select to use Visual Studio 2019, Visual Studio 2022, Visual Studio 2026, or WSL GCC as compiler, you can then select to use Clang as code generator instead of native Visual Studio/WSL GCC.

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 95 for more information. Note that you can use Clang as code generator.

2 About this booklet

This booklet covers Dymola 2026x Refresh 1. The disposition is similar to the one in Dymola User Manuals; the same main headings are being used (except for, e.g., Libraries and Documentation).

3 Dymola 2026x Refresh 1

Note! *From this version, Dymola 2026x Refresh 1, Microsoft Windows 10 operating system is not officially supported.*

3.1 Introduction

3.1.1 Additions and improvements in Dymola

A number of improvements and additions have been implemented in Dymola 2026x Refresh 1. In particular, Dymola 2026x Refresh 1 provides:

- Simulation performance reports (page 21)
- Improved GUI for model calibration (page 62)
- Dymola Modelica Compiler (DMC) tool (page 29)
- Replacing analytic Jacobian by directional derivatives (page 24)
- Virtual Companion Integration – LEO on the **3D**EXPERIENCE Platform (*beta feature*) (page 45)
- Multiple sets of display units (*beta feature*) (page 56)
- Harmonization and name changes of flags, and how to update the names (page 10)
- Improved navigation from Used Classes layer in the Modelica Text editor
- Plot: Smart simplification of plot legend (page 25)
- Support for the Microsoft Visual Studio 2026 compiler (page 41)
- Discontinued support for Microsoft Windows 10 operating system (page 43)
- Discontinued support for the 32-bit MinGW compiler (page 42)
- Upgrade of FLEXnet Publisher version (page 43)
- Model Experimentation: Running parameter sweep for a group of parameters (page 58)
- Changed GUI for versioning commands – a new tab (page 67)
- Support on Windows for code signing of simulation binaries (page 69)
- Improved support for SSP
    - Renaming of SSP flags (page 76)
- Improved support of FMI:
    - Renaming of FMI flags (page 79)
    - FMI export:
        - Exposing nonlinear equations and iteration variables in Model Exchange FMUs (*beta feature*) (page 54)

        - § Specifying the FMI model version when exporting an FMU (page 79)
        - § Specifying the preferred communicationStepSize for exported Co-simulation FMUs (page 81)
    - o FMI import:
        - § FMI 2: Option to preserve the hierarchical structure of IO (page 82)
        - § FMI 2 and FMI 3: Option to use a standardized package of FMI functions when importing FMUs (page 83)
- Improved support of eFMI, main improvements: (page 88)
    - o Integrated production code generation
    - o Improved event handling of GALEC code generator
    - o Export of production code as FMI 3.0 source code co-simulation FMU
- Model structure: Improved model editing API (page 89)
- Improvement in the 3DEXPERIENCE app “Design with Dymola”: Working with simulation data in Design with Dymola and 3DEXPERIENCE (page 71)

3.1.2 New and updated libraries

New libraries

There are no new libraries in this Dymola version.

Updated libraries

The following libraries have been updated:

Note. The below list build on what is displayed when using **File > Libraries**. The name of the package in the package browser may be different, and it may be that more than one package is opened, due to help packages, underlying library combinations etc.

- Aviation Systems Library, version 1.7.1
- Battery Library, version 2.9.0
- Brushless DC Drives Library, version 1.4.5
- ClaRa DCS Library, version 1.9.0
- ClaRa Grid Library, version 1.9.0
- ClaRa Plus Library, version 1.9.0
- Claytex Library, version 2026.1
- Claytex Fluid Library, version 2026.1
- Cooling Library, version 1.5.6
- Dassault Systemes Library, version 1.15.1
- Design Library, version 1.2.5
- Dymola Commands Library, version 1.21

- Dymola Embedded Library, version 1.0.8
- Dymola Models Library, version 1.12.0
- eFMI_TestCases, version 1.0.8
- Electric Power Systems Library, version 1.7.2
- Electrified Powertrains Library (ETPL), version 1.13.0
- Fluid Dynamics Library, version 2.21.0
- Fluid Power Library, version 2026.1
- FTire Interface Library, version 1.3.4
- Human Comfort Library, version 2.21.0
- HVAC (Heating, Ventilation, and Air Conditioning) Library, version 3.6.0
- Hydrogen Library, version 1.4.4
- Model Management Library, version 1.4.3
- Modelica_DeviceDrivers, version 2.2.0
- Pneumatic Systems Library, version 1.7.4
- Process Modeling Library, version 1.4.1. **Note**. This library is currently not supported on Linux.
- Sustainable Supply Systems Library, version 1.1.0
- Testing Library, version 2.0.0
- Thermodynamics Connector Library (previously named Multiflash Media Library), version 1.3.2. **Note**. This library is currently not supported on Linux.
- TIL Suite 2026.1:
    - TIL 2026.1
    - TIL Adsorption 2026.1
    - TIL Automotive 2026.1
    - TIL Heat Storage 2026.1
    - TIL Hydrogen Energy Systems 2026.1
    - TIL PCM Storages 2026.1
    - TIL NTU 2026.1
- VeSyMA (Vehicle Systems Modeling and Analysis) Library, version 2026.1
- VeSyMA - Engines Library, version 2026.1
- VeSyMA - Powertrain Library, version 2026.1
- VeSyMA - Suspensions Library, version 2026.1
- VeSyMA2ETPL Library, version 2026.1
- Visa2Base, version 1.20
- Visa2Paper, version 1.20
- Visa2Steam, version 1.20

For more information about the updated libraries, please see the Release Notes section in the documentation for each library, respectively.

3.2 Developing a model

3.2.1 Harmonizing, changing, and updating flag names

Harmonizing flag names

For many Dymola versions, most `Advanced` flags had a flat naming structure, that is, the flags were named `Advanced.xxx`. When the **Flags and Variables** GUI for handling flags was introduced in Dymola 2019 FD01, the flags were inserted in a tree structure. It was natural to introduce parallel hierarchical flag names according to this structure, like `Advanced.Editor.xxx` for the flags. When this naming was introduced, it was only used in the **Flags and Variables** GUI, but later you could use both names everywhere.

As time went by, the **Flags and Variables** GUI was slightly changed to prefer the hierarchical flag names, the tooltip did not show the old names anymore, and searching for the old names, the new names were displayed instead.

In Dymola 2026x Refresh 1, the hierarchical names are considered the only fully valid names. However, you can still use the old flag names, but they will give warnings when used, telling that these names are obsolete. An example of such a warning:

```
Warning: Script using obsolete flag Advanced.AbsSignEvent. Flag
is now called Advanced.Modelica.AbsSignEvent.
```

Changing flag names

In addition to the above harmonization, a major renaming has been implemented for SSP and FMI flags. See section “Renaming of SSP flags” starting on page 76, and section “Renaming of FMI flags” starting on page 79, respectively.

Important! Note the difference in the name changing of SSP flags and FMI flags. For FMI flags, the old names can still be used, but generate warnings. For SSP flags, the old flag names are removed, you **must** change the names to the new names; if not, you will get errors.

Updating flag names

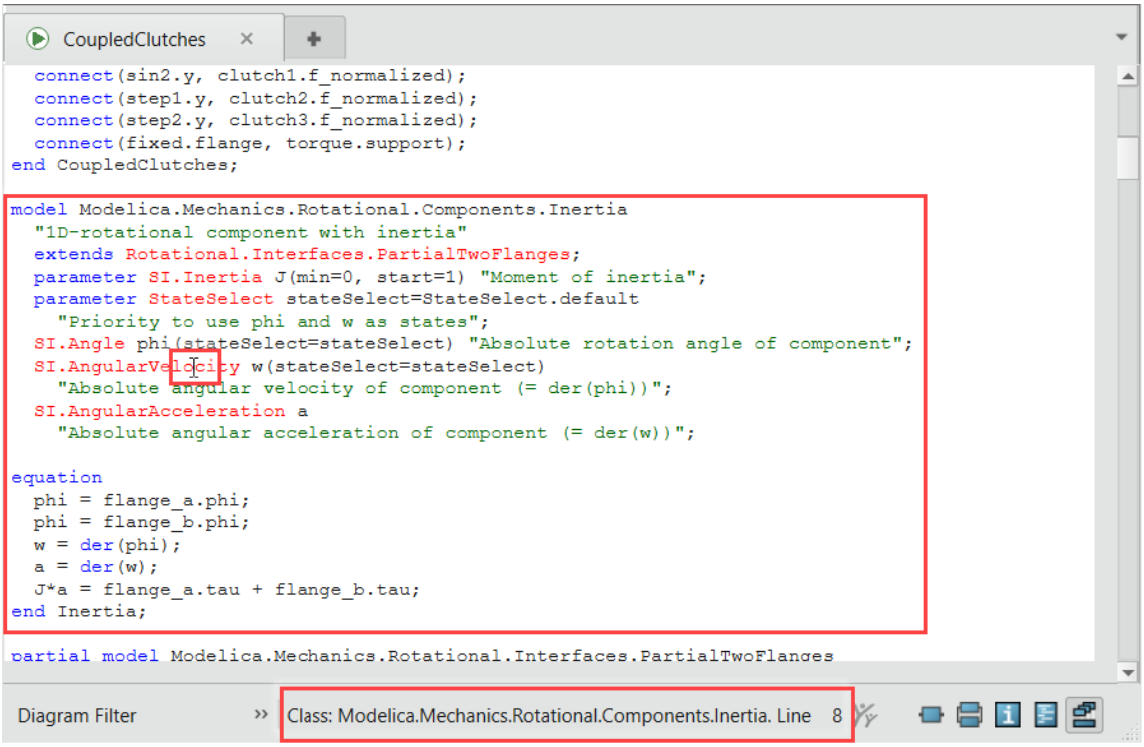
To update flags to the current names in models and scripts, you can use the built-in function `updateModelicaFlags`, see section “New built-in function to update Advanced flag names in models and scripts, and simulation setup flag names saved in the model” on page 32.

3.2.2 Minor improvements

Improved navigation from Used Classes displayed in the Modelica text editor

Used class name and line number in used class displayed

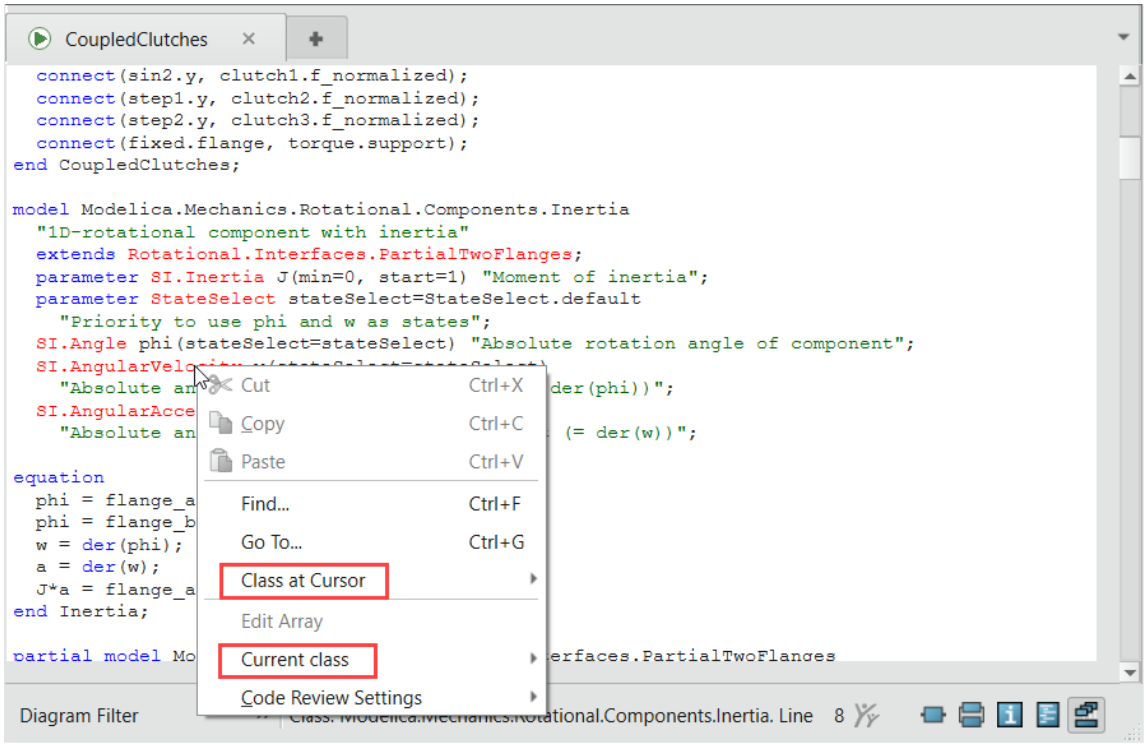
When clicking in the Used Classes layer in the Modelica text editor, the name of the used class you click in appears in the bottom bar of the main window, as well as the code line number you clicked on, for *that* used class. An example from the demo Coupled Clutches:



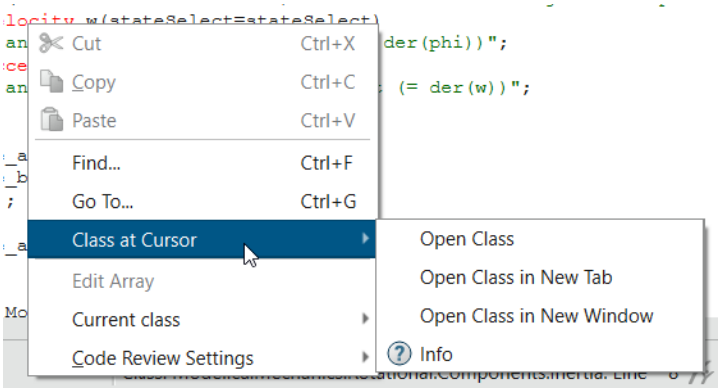
(Note that the display of the code line number is also implemented for the “usual” Modelica text presentation.)

Context commands for opening classes

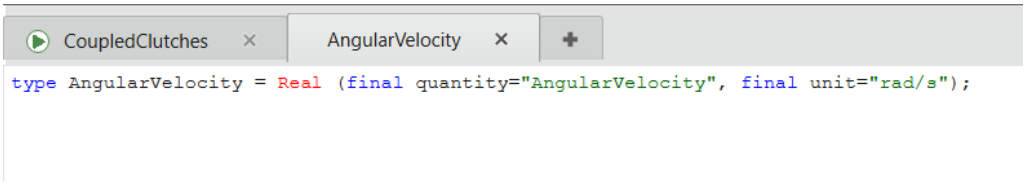
Right-clicking when the above clicking has been made, you get the context menu:



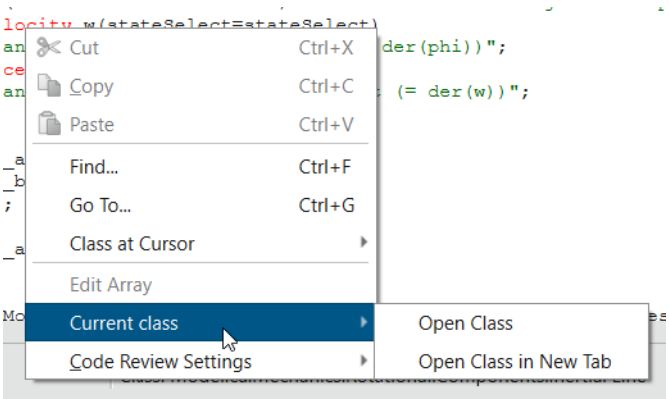
The first framed commands above was available also in previous versions of Dymola, but was then named **Selected Class**. To indicate better the context of the command, it has now been renamed to **Class at Cursor**. As previously, the command has the subcommands:



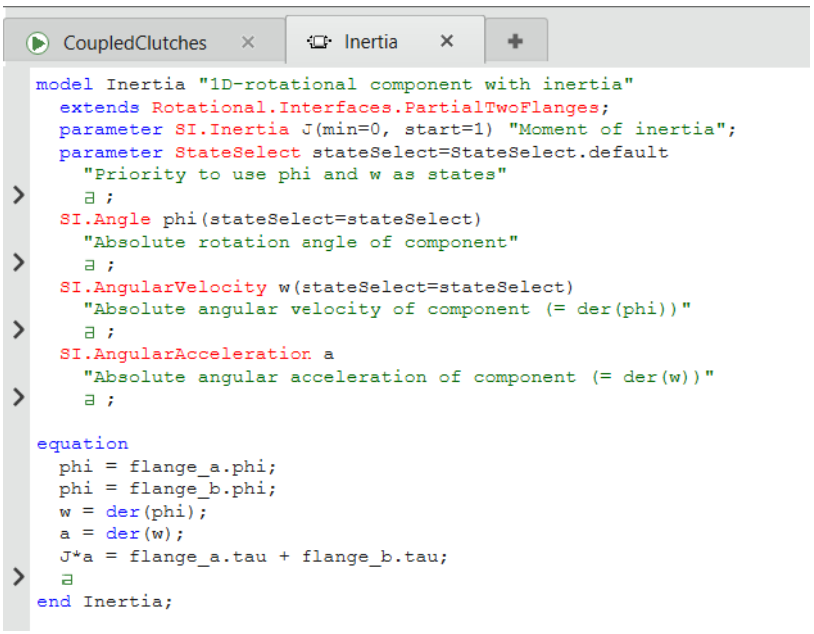
Now, right-clicking, and using the command **Class at Cursor > Open Class in New Tab**, from the cursor location above, you open, as in previous Dymola versions, the *class at the cursor position* in a new tab, in Modelica text mode:



The second framed context command, **Current Class**, is a new command in Dymola 2026x Refresh 1. It has the subcommands:



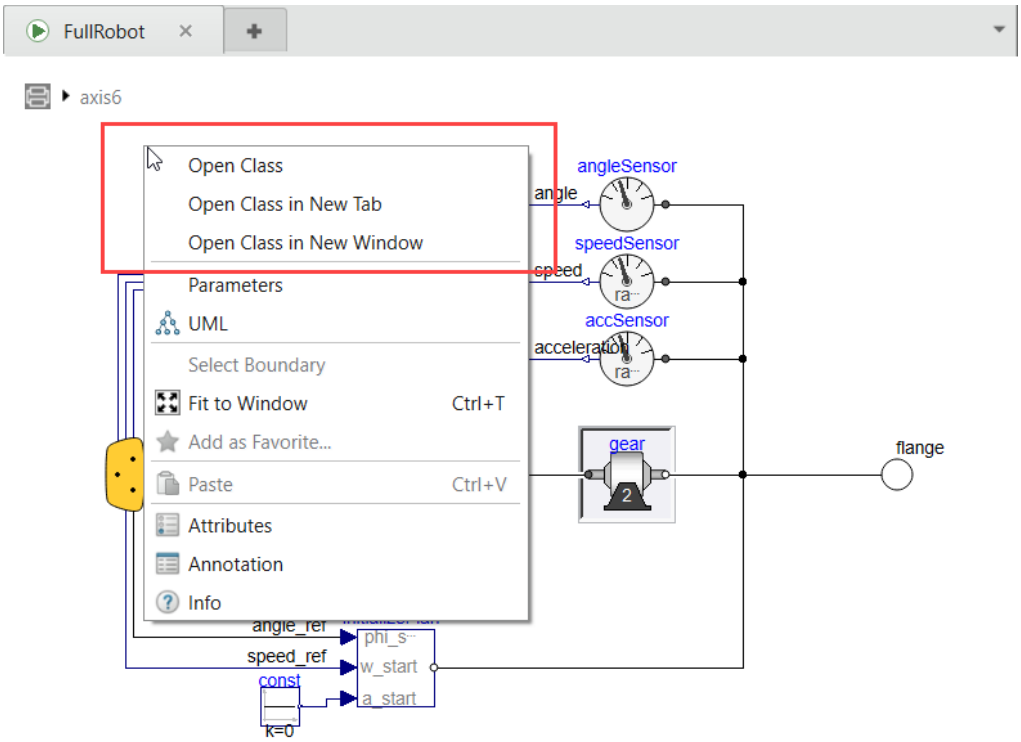
Now, right-clicking and using the command **Current Class > Open Class in New Tab**, from the cursor location above, you instead open *the used class* at the cursor position in a new tab, in Modelica text mode:



Note that this used class is also the used class that is displayed with name in the bottom bar of the main window in the Used Classes display (compare with the first figure above).

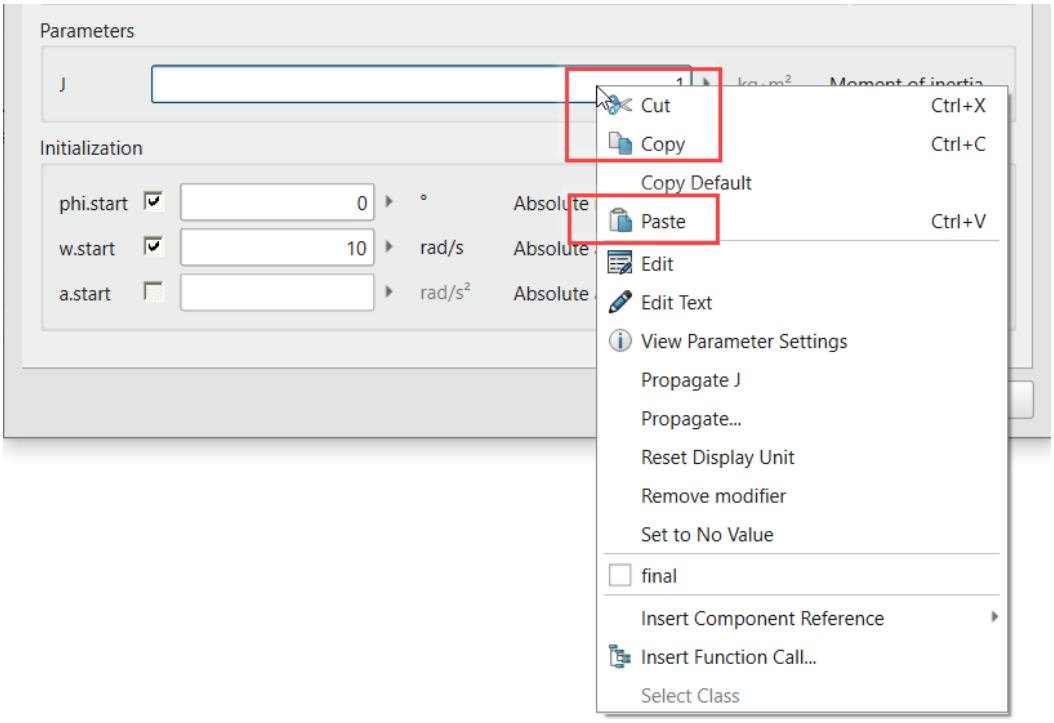
Easier to open the editable class after having used the command Show Component

Previously, when you wanted of open the class shown when using the command **Show Component** to edit it, you had to use the component browser. Now the commands for opening the class are present in the context menu of the diagram of the shown class. An example from the Robot demo:



Cut, Copy, and Paste added for parameter dialog input fields

The commands **Cut**, **Copy**, and **Paste** are now available in the input fields of the parameter dialog:



Default line thickness

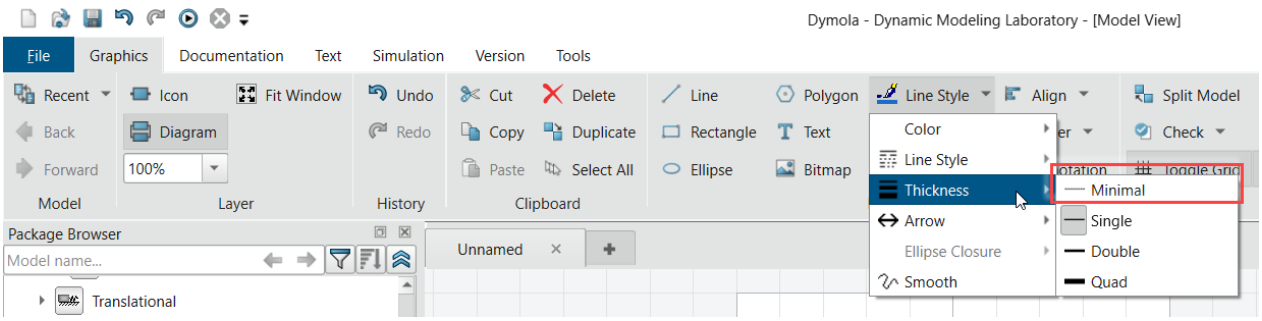
The default line thickness used is now 0.25 mm, which is in accordance with the Modelica Language Specification. The default line thickness is specified by the flag `Advanced.UI.DefaultThickness`. The default value of the flag is `true`. This is the line thickness **Single**; see below image. (If you set `Advanced.UI.DefaultThickness=false`, the default line thickness value will be 0.)

(The earlier name of the flag was `Advanced.Beta.UI.DefaultThickness`. In addition to name change, not being beta flag anymore, the default value is changed to `true`. *This feature was a beta feature in the previous Dymola version.*)

To update old models, you can use `updateModelicaAnnotations`, this built-in function is now enhanced to handle this. See section “The built-in function updateModelicaAnnotations enhanced” on page 32.

New minimal line thickness

A new line thickness **Minimal** has been added, corresponding to 1 pixel (even when zoomed in). This is implemented in GUI for graphical objects, as well as for scripting. For GUI:



Controlling what ModelicaServices library to demand-load

You might have access to different ModelicaServices libraries, since the ModelicaServices can be seen as a “template” available from Modelica Association, to be adapted by various vendors.

In Dymola 2026x Refresh 1, when working with Dymola, you by default demand-load the ModelicaServices that matches the Dymola application, if there are several ModelicaServices available. To go back to the behavior from previous Dymola versions, that is, to not prefer what ModelicaServices library to demand-load, you can set the flag:

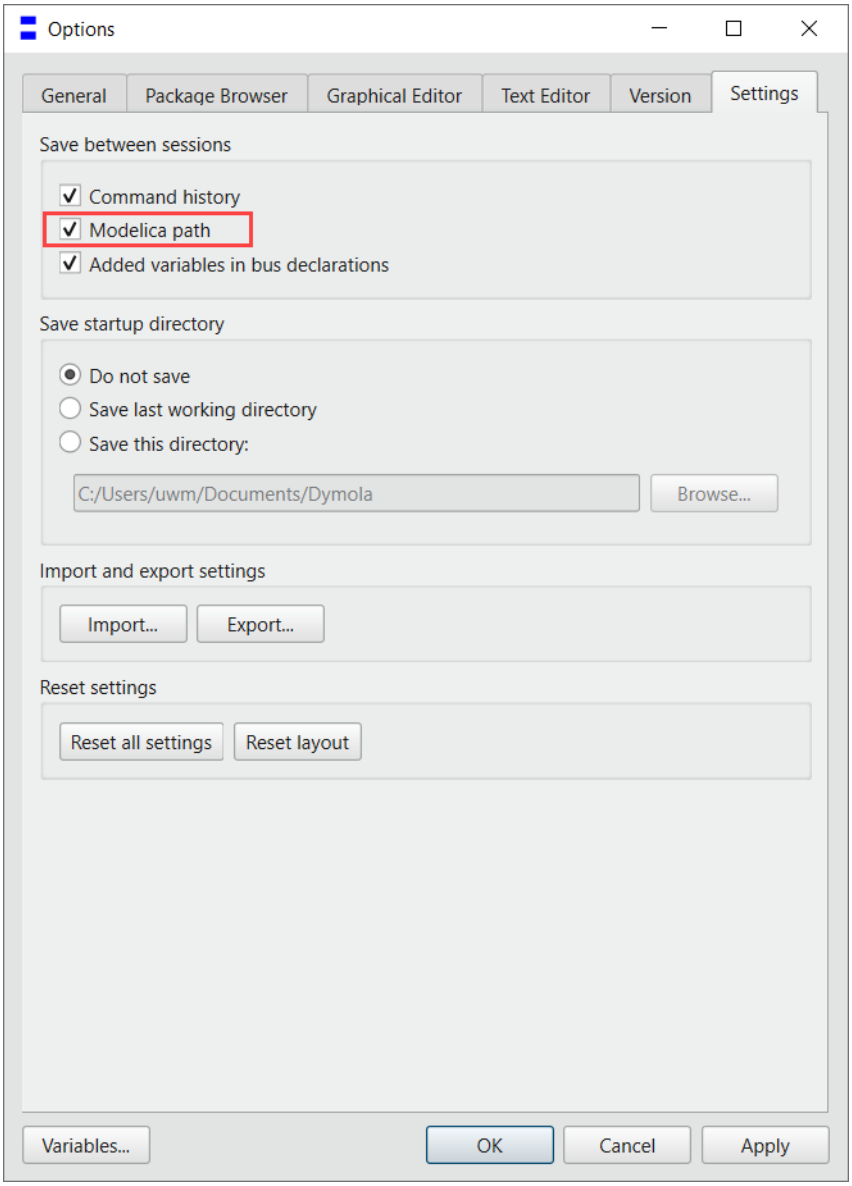
```
Advanced.UI.MatchingModelicaServices = 0
```

(The flag value is by default 1.) The value of the flag is saved between sessions.

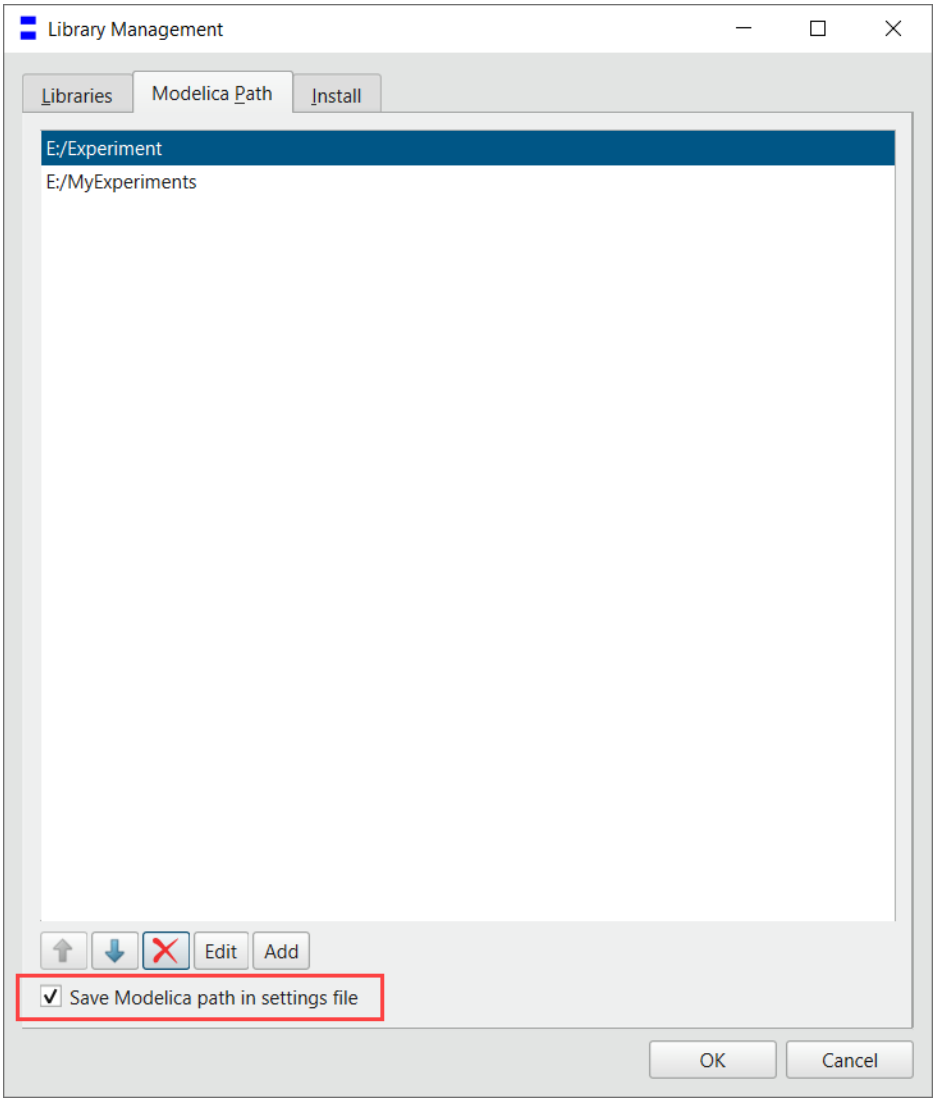
By default, Modelica Path is saved in the settings file

By default, the Modelica path is now saved in the settings file. This means that by default, the flag `Advanced.UI.Store.ModelicaPath` is `true`. It also means that the below settings are by default activated:

In **Tools > Options**, the **Settings** tab:



In **Tools > Library Management**, the **Modelica Path** tab:



Better handling of enabling the editing of a parameter in the parameter dialog by another parameter

Previously, some cases of enabling the edition of a parameter in the parameter dialog by another class parameter did not work. The flag `Advanced.Editor.EnableUsingModifierLogic` can be used in those cases. The value of the flag can be any of the following:

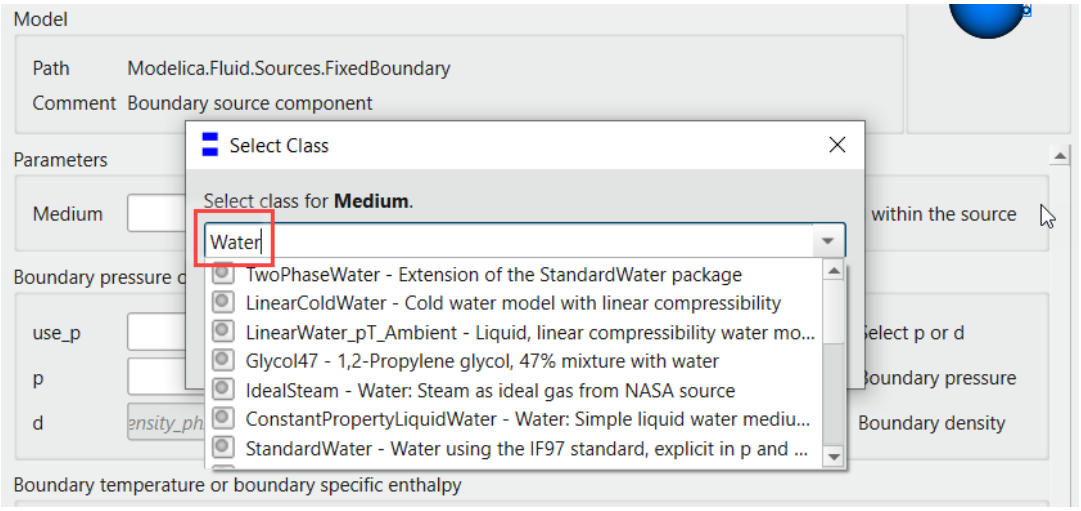
- 0 – No change of behavior compared to older Dymola versions.
- 1 – Smart handling, enable new logic when parameters depend on parameters. This is the default value of the flag.

- 2 – Always use the new handling of parameters.

(*This feature was a beta feature in the previous Dymola release, with the default value of 1 for the corresponding Beta flag.*)

Improved filtering for the Select Class dialog in parameter dialogs

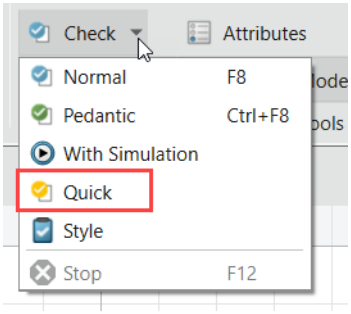
In parameter dialog, in the context menu of a parameter field, you have the entry **Select Class** for re-declaration of the class. Selecting this entry, you get a dialog with all matching classes to select from. In Dymola 2026x Refresh 1, you can enter text in the first line to filter the matching classes on the entered characters. An example:



Smarter top-level structural check

A new feature is that you can perform a top-level structural check only checking the top-level model, ignoring sub-components.

You can activate the check from the GUI, as a new alternative of the Check command, one example:



You can also activate the check by setting the flag:

```
Advanced.Check.TopLevelStructuralQuick=true
```

It is intended to do a quick check of the model to detect common errors in manually and/or AI-generated models (like missing connections, incorrect class names, etc.) This check is currently faster than regular check, and still detects relevant issues at the top.

(The flag by default `false`. It is not saved between sessions.)

The check may fail with an inconclusive result if the model relies on variables from sub-components.

(*This feature was a beta feature in the previous Dymola release. It has been improved in this version by not requiring any additional flag to be set, and is now faster.*)

Comparing this new flag with other already present flags for checking, you have:

- `Advanced.Check.TopLevelStructural`, first checks the top-level for common errors and then does a normal check.
- `Advanced.Check.TopLevelStructuralTemplate`, automatically performs a top-level check in case the model could be simulated except for the partial components (i. e. it is a template). Note that this flag is `true` by default.
- `Advanced.Check.TopLevelStructuralQuick`, only checks the top-level.

Checking for comparison between real entities

The check for comparison between real entities not being parameters previously did not issue diagnostics for some incorrect models; now such diagnostics is given. **Note** that is legal to compare reals in functions, that is consistent with functions (as default) not generating events for less than etc.

When checking, checking for comparison between real entities *not* being parameters is *always* included.

By default, the check now also includes checking for comparison between *parameters* of type real. To deactivate this check, you can set the flag `Advanced.Modelica.RealComparison.Parameter = false`. (The flag is by default `true`.) The flag is not saved between sessions.

By default, all findings from the check are issued as warnings. To make them into errors, you can set the flag `Advanced.Modelica.RealComparison.Strict = true`. (The flag is by default `false`.) The flag is not saved between sessions.

Generating zipped help files

Dymola can generate zipped help files if you set the flag:

```
Advanced.HTML.ZipGeneratedHelp = true
```

This will reduce the number of files that need to be installed, and reduces storage requirements. It also might help somewhat when it comes to problems with long file names that may occur for help files.

The feature is only applicable if the following is fulfilled:

- The packages are stored in folders, not as single files.

- The type of HTML documentation selected when generating the help files is **Model documentation (including Modelica text)**.
- The help files are generated in help subdirectories, that is, the flag `Advanced.HTML.ResourcesHelpDymola` is `false`. (This is the default value.)

The zipped help file is automatically unzipped in a temporary directory the first time you access the help contents in Dymola.

(The default value of the flag for zip generation is `false`. The flag is not saved between sessions.)

Improved automatic re-routing of connections

Already in earlier versions of Dymola, when moving a component, the connection lines of that component were automatically re-routed, if the setting **Automatic routing of connections** was activated. (This setting is reached by the command **Tools > Options**, the **Graphical Editor** tab.)

In Dymola 2026x Refresh 1, this automatic re-routing is also applied if the component is rotated or flipped, given the above setting is activated.

A minor, related, improvement is that the optional text of a connection is now better orientated than previously.

Option to select to store classes in the enclosing package.mo

The advanced menu for storing packages hierarchically has been improved. This menu is reached by right-clicking a package in the package browser and selecting **Attributes** (the **General** tab), and then clicking the button **Advanced settings for Hierarchical storage**.

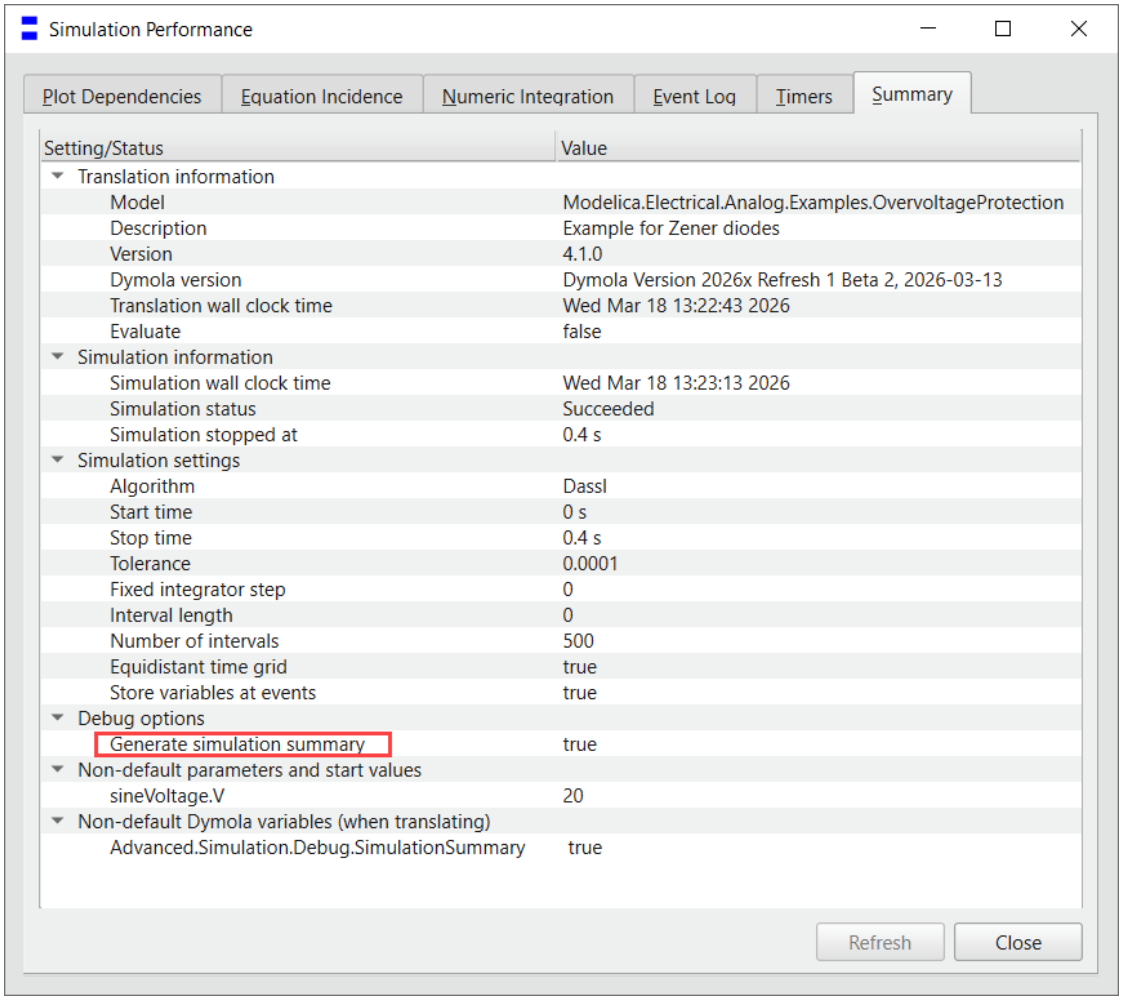
It is now possible to select to store classes in the enclosing `package.mo`, either by selecting it for each specific class, or select to store an entire package as one file in a directory.

This is used in rare cases for individual classes like the World class in the MultiBody library, and for the entire ModelicaReference package.

3.3 Simulating a model

3.3.1 Simulation performance report

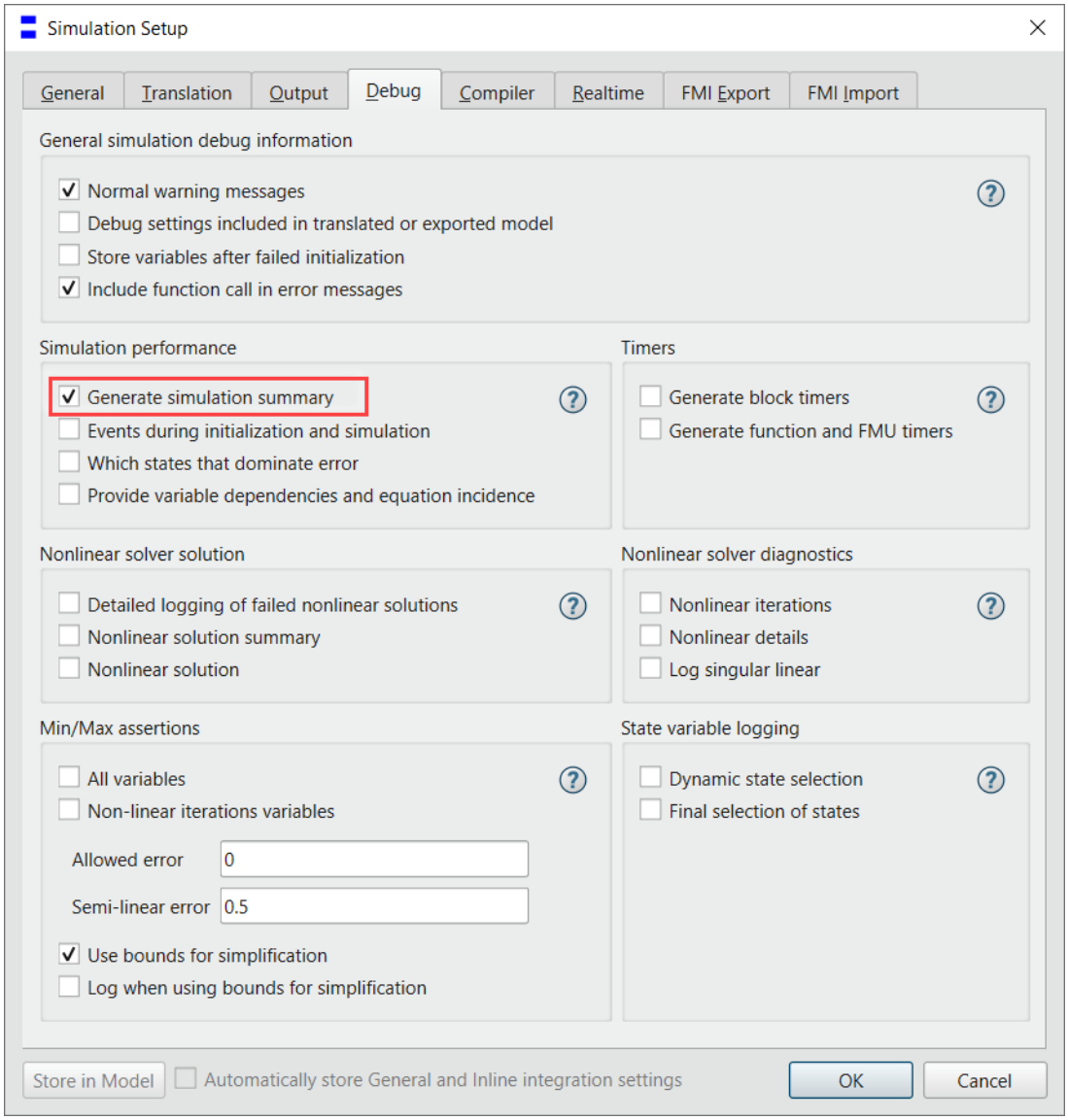
In Dymola 2026x Refresh 1, the **Simulation Performance** window contains a new tab **Summary**. This is a simulation performance report, currently consisting of the most relevant information from the translation and simulation of a model. An example of such a report:



Most information is self-explanatory, but it may be mentioned that “Non-default parameters and start values” is a list of all parameter values and start values that were changed after the last translation, but before the last simulation. Note that also that the debug options and non-default flags are listed; the option to generated performance reports is framed in the figure above.

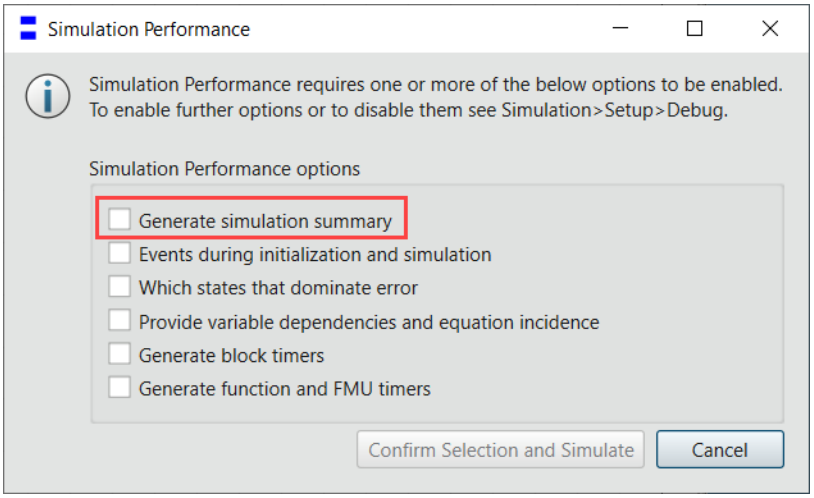
You reach the **Simulation Performance** window by the command **Simulation > Performance**.

By default, the option to create such a report is not activated. You can activate the option **Generate simulation summary** in the simulation setup (reached by the command **Simulation > Setup**), in the **Debug** tab:

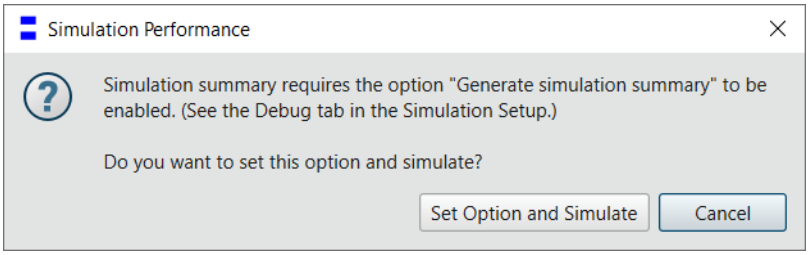


The setting corresponds to the flag `Advanced.Simulation.Debug.SimulationSummary = true`. (The flag is by default `false`.) The flag/setting is saved between sessions.

The report can also be enabled when opening the **Simulation Performance** window by the command **Simulation > Performance**. If no related option is activated for this window, you get:



If you already have some of the other options above activated, the **Simulation Performance** window is opened. If you click in the **Summary** tab in such case, you get the message:



Note that the simulation summary report uses information both from translation and from simulation. Therefore, it is best to activate the option *before* translation; else, you will have to re-translate and re-simulate to get the full report. (The **Refresh** button in the report can be used to update the report when you have performed a new simulation, but cannot activate re-translation.)

The simulation summary report file can be converted from text format to SSP LS Traceability meta-data format (SRMD). This is done by running the Python-script …`Dymola/insert/simreport.py`.

3.3.2 Replacing analytic Jacobian by directional derivatives

The analytic Jacobian can be replaced by directional derivatives.

There are two benefits:

- Less code compared to analytic Jacobian (especially important if hundreds of states and/or inputs); the additional code for directional derivatives is similar in size to the code without Jacobians.

- More natural fit for directional derivatives in FMUs.

It still gives the same accuracy as analytic Jacobian, but it may be slightly less efficient.

The analytic Jacobian now also generates a smaller code size for many models.

The feature is controlled by the flag `Advanced.Translation.JacobianOrDirectional`. The flag can have the following values:

- **0** Jacobian (default)
- **1** Directional derivative
- **2** Both

The flag is saved between sessions. Note that the setting **Generate Analytic Jacobian for the ODE problem** must be activated for this flag to have any meaning. (You reach the setting by the command **Simulation > Setup**, the **Translation** tab.)

3.3.3 Harmonizing, changing, and updating flag names

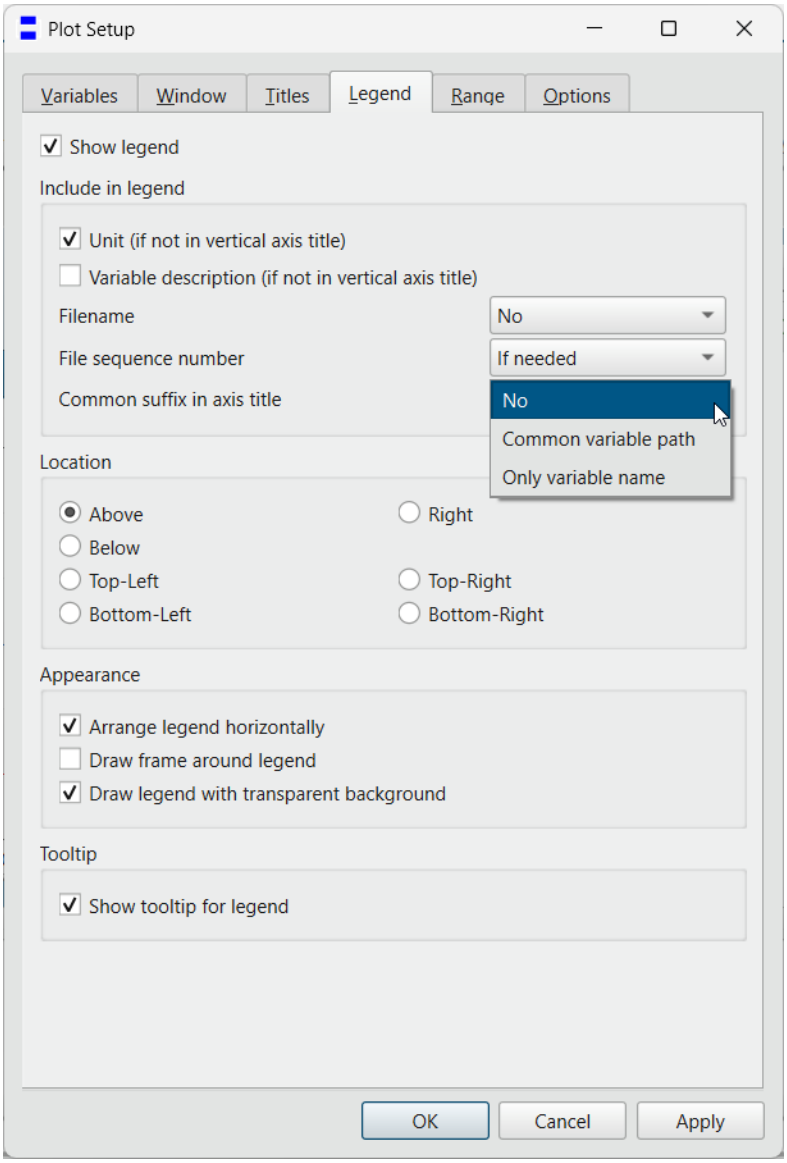
For more information, see the corresponding section for model development.

3.3.4 Plot tab

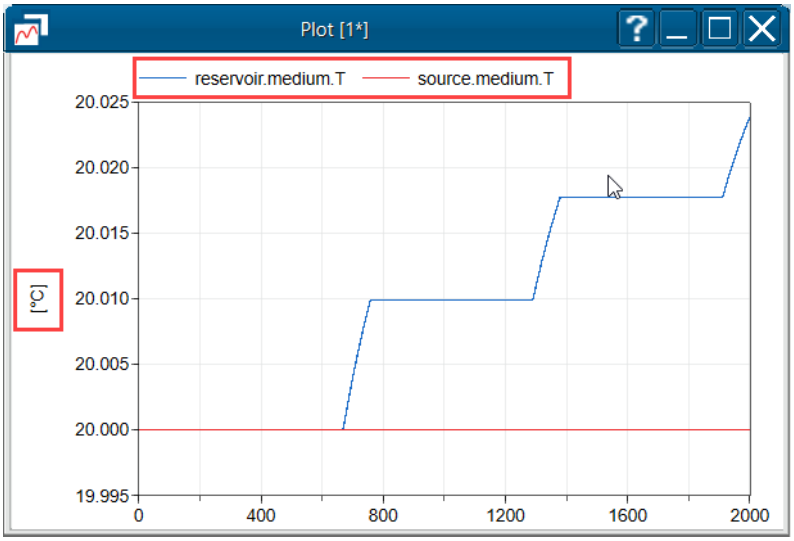
Smart simplification of plot legend

It is quite common that you plot variables with some common parts in the end of the names (suffix) in the same plot. If so, you can now select if you want to move such suffix from the variable legends to the axis title instead. The feature works for both left and right axis.

You do that by a new option **Common suffix in axis title** in the plot setup (reached by right-clicking in the plot and selecting **Setup…** the **Legend** tab):

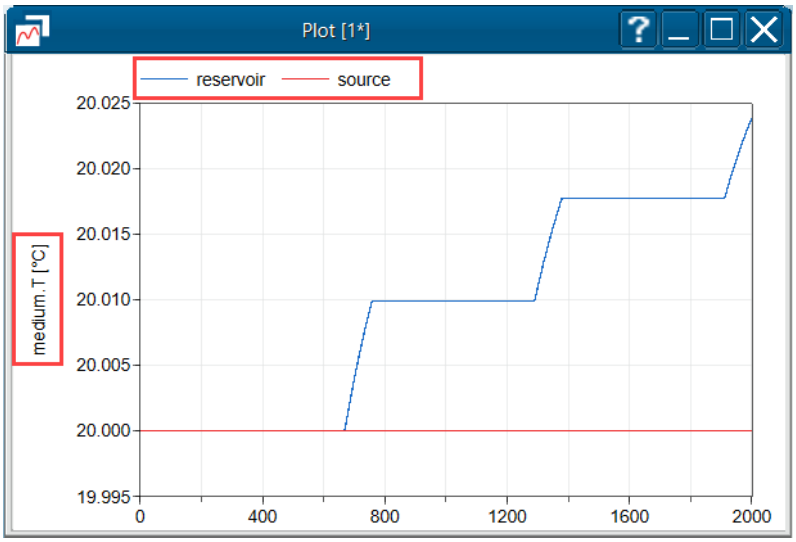


As an example, consider the model `Modelica.Fluid.Examples.PumpingSystem`. If you simulate that model and plot the two variables `source.medium.T` and `reservoir.medium.T` in the same plot, you get, by default. (The legends and axis title are framed in red for clarity in the images below.):



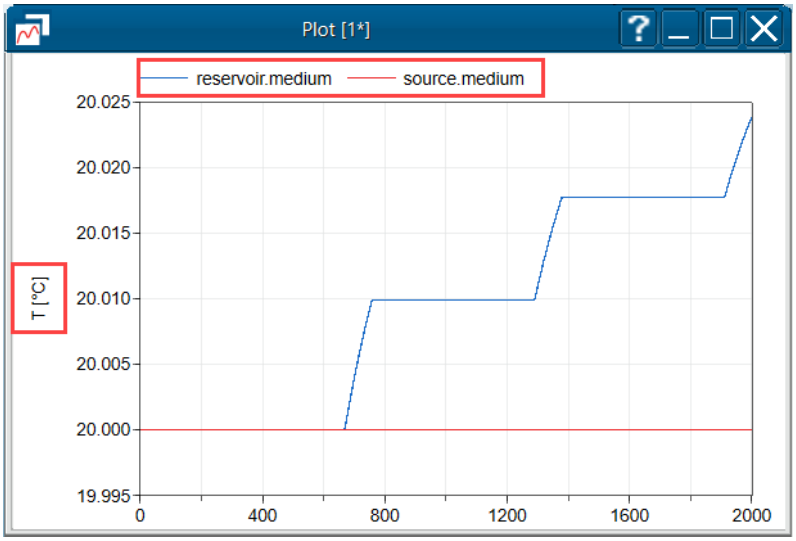
This is the behavior of previous versions of Dymola, nothing is moved; it corresponds to **No** in the menu above. (There is a corresponding flag value `Advanced.Plot.Legend.Simplified = 0`.)

Now, if you set the above option to **Common variable path**, you get the plot below. Note that here the whole common last part of the signal names is moved to the axis title.



(This corresponds to the flag value `Advanced.Plot.Legend.Simplified = 1`.)

If you set the above option to **Only variable name**, you get the plot below. Here only the common signal name is moved to the axis title.



(This corresponds to the flag value `Advanced.Plot.Legend.Simplified = 2`.)

The setting is saved between sessions.

Aligning grids of vertical axes in plots

In Dymola 2026x Refresh 1, if you use right axes in plots, their grid is, by default, always aligned with left axes when using the same unit on the axes.

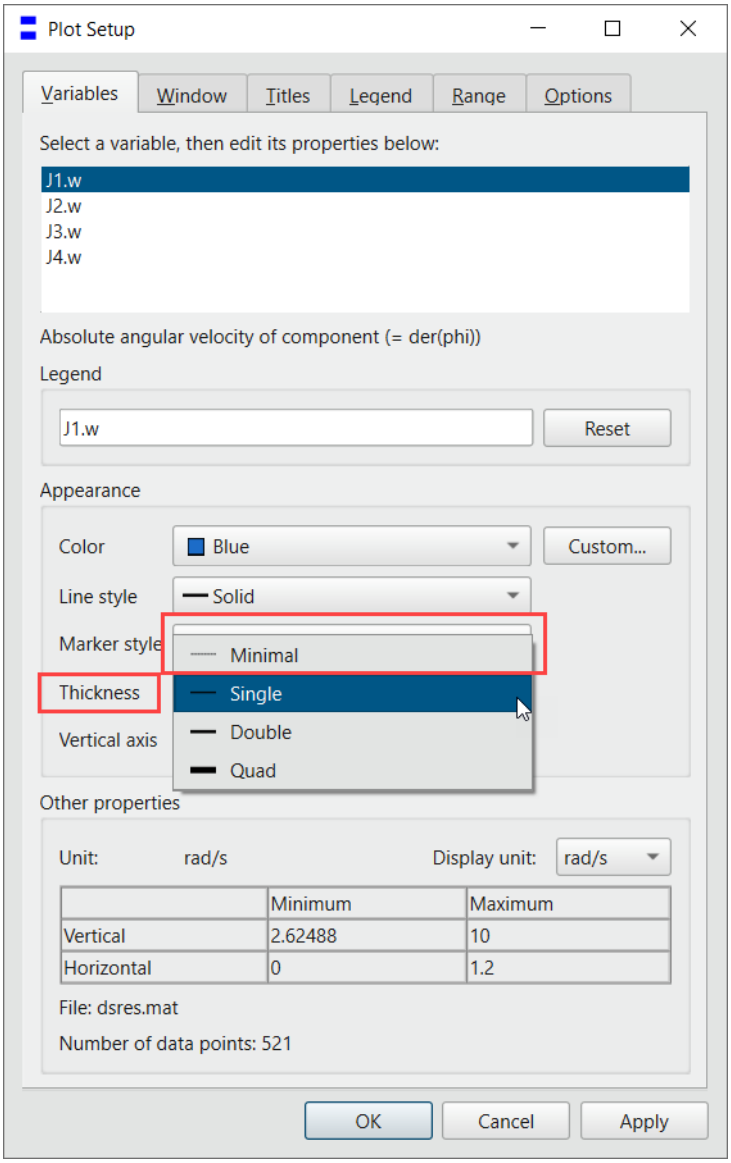
You can disable this feature by setting `Advanced.Plot.AlignVerticalAxes = false`.

(The flag value is by default `true`.) The flag value is saved between sessions.

Note that in most cases, you have different units on the left and right axes; then this is not a problem.

New minimal line thickness for curves in plots

A new line thickness **Minimal** has been added, corresponding to 1 pixel (even when zoomed in). This is implemented in GUI for curves, as well as for scripting. You can change the thickness of a curve by the command **Plot > Setup**, the **Variables** tab:



3.3.5 Scripting

Dymola Modelica Compiler (DMC) tool

Overview

There has been a long-standing wish to better support scripting and remote execution in Dymola. Typical use-cases include Continuous integration toolchains, optimization, deployment on computing clusters, and scripting from command scripts, or e.g. Python. In

each of these cases, the normal Dymola user interface is more of a problem than a benefit, even if it can be hidden with the option `–nowindow`.

Note that also means that DMC is useful where there is no monitor available, such as headless servers or containers and virtual machines without a graphical console.

We provide a minimalistic version of Dymola, the Dymola Modelica Compiler (DMC), which is a command line tool without any graphical user interface at all. In short, DMC is designed to:

- Translate and simulate Modelica models.
- Run `.mos`-scripts.
- Accept commands on a network port, to support e.g. Python integration.

Operation

The Dymola Modelica Compiler is located in the `dymola\bin64` directory, and is started as:

```
dmc.exe
```

It supports the following command line arguments:

Option	**Description**
-h	Prints a summary of available operations.
-o *file-name*	Opens the named Modelica file into DMC.
-x *command*	Runs the given command line. For example: Dos: `dmc –x "simuateModel(""Foo"", stopTime=2);"` Linux: `dmc –x 'simulateModel("Foo", stopTime=2);'` (For Linux, the shell-quoting may differ between Linux versions.)
-r *file-name*	Runs a script file (.mos)
-p *port*	Starts the HTTP server on the designated port. This option is needed to support e.g. Python integration.

Constraints

Due to the nature of the command line tool, there are several constraints:

- There is no user interface to set up the license server, the C compiler, etc. Instead, DMC will read the Dymola setup file `setup.dymx` to get the initial setup.
- Commands that are graphical in nature are not implemented, for example, anything related to plotting and animation. Likewise, operations that use the graphical representation of the model, such as `exportDiagram()`.

- DMC uses a Dymola license and licenses for optional products, such as libraries. To ensure uninterrupted execution of a sequence of DMC commands, the license will remain checked out for a short period after the last DMC execution.

Using DMC with Python

When instantiating the Python interface, you can now specify if Dymola standalone should be used or the new Dymola Modelica Compiler (DMC).

There is a new argument to the constructor of DymolaInterface called `kernel`. When set to `True`, DMC is used. The Python interface will look for the executable file `dmc.exe` in the installation folder. The default value is `False`, which starts standalone Dymola.

```
dymola = DymolaInterface(kernel=True)
```

Note 1, `kernel=True` needs to be set even when providing the path to the DMC executable.

```
dymola = DymolaInterface("C:/Program Files/Dymola 2026x Refresh
1/bin64/dmc.exe", kernel=True)
```

Note 2. On Linux, you need to call the start script for DMC to get the correct environment variable setting. There are alternatives:

- For the latest installed Dymola:

  ```
  dymola = DymolaInterface('/usr/local/bin/dmc', kernel=True)
  ```

- For Dymola 2026x Refresh 1:

  ```
  dymola = DymolaInterface(
  'opt/dymola-2026xRefresh1-x86_64/bin64/dmc.sh', kernel=True)
  ```

(This feature was a beta feature in the previous Dymola version.)

Stricter diagnostics when attempting to set an evaluated parameter

In some cases, the built-in function `simulateExtendedModel` tries to set an evaluated parameter. This can also happen in general scripting. Previously such issues gave warnings that could easily be ignored. In Dymola 2026x Refresh 1, to have more options here, a new flag `Advanced.Simulation.StrictNonParameters` is introduced, with the following possible values:

- **0** – Warnings as before.
- **1** – Error if `simulateExtendedModel` tries to set an evaluated parameter, but warning if script tries the same (default value of flag)
- **2** – Error also if scripts tries to set an evaluated parameter.

The value of the flag is not saved between sessions.

New built-in function to load a result file

A new built-in function that loads the result file has been added:

```
loadResult(fileName, loadAnotherCopy=false)
```

The string input argument `fileName` specifies the result file to load. To load a file unconditionally, you can set `loadAnotherCopy=true`, by default the loading of an already loaded file is skipped.

The built-in function corresponds to the command **Simulation > Load Result**.

New built-in function to update Advanced flag names in models and scripts, and simulation setup flag names saved in the model

A new built-in function has been added, to update Advanced flag names in models and scripts, and simulation setup Advanced flags saved in the model:

```
updateModelicaFlags(className, experimentFlags=true,
scripts=true)
```

The function has the following input arguments:

- `className` – usually you specify the top-level package name (`String` argument), but you can in some cases also specify a certain class/model inside a package.
- `experimentFlags` – when set to `true` (default value), you update the simulation setup flags stored in the package/model. (Such simulation setup flags are saved under the annotation `__Dymola_experimentFlags` in a model.)
- `scripts` – when set to `true` (default value), you update `.mos`-scripts in the directories named **Resources** and **Scripts**. **Note**! For this to work, you must specify the corresponding top-level package by the `className` argument.

Important! This built-in function cannot handle Advanced flag names in two cases; you must manually update the flags in these cases:

- Flag names inside user-defined functions.
- Flag names of removed flags; see section “Removed Advanced flags” on page 37.

The built-in function translateModelFMU enhanced

The built-in function `translateModelFMU` now has a new input string argument `FMUModelVersion`, which allows you to specify a model version for the exported FMU. The default is an empty string.

The version you specify sets the `fmiModelDescription.version` attribute in the file `modelDescription.xml`. If you leave `FMUModelVersion` as an empty string, the attribute will be the version extracted from the corresponding Modelica package (if such a version exists).

There is a corresponding setting the GUI, see section “Specifying the FMI model version” on page 79. Note the last section about the information added in the documentation view, this is the case also for scripting.

The built-in function updateModelicaAnnotations enhanced

The built-function `updateModelicaAnnotations` now has three new arguments:

- A Real input argument `setDefaultThickness`, by default -1. If you set this value to 0 or higher, all unspecified line thickness are set to this value. Modelica Language

Standard specifies 0.25 and previous versions of Dymola used 0.0. If you consider the model working fine in previous Dymola versions, you can set `setDefaultThickness = 0.0` to update the old model while preserving the visible line thickness.

- A Boolean input argument `noBorderGradient`, by default `false`. If you set this argument to `true`, all shapes with gradients set (except ellipses and rectangles with `FillGradient.Gradient`) get the line pattern set to `LinePattern.None`.
- A Boolean input argument `removeEmptyAnnotations`, by default `true`. This means that by default, when the built-in function is executed, it now also removes any empty annotation. (These empty annotations have no meaning semantically.)

The built-in function convertElement enhanced to handle conversion between replaceable class and replaceable component (parameter), and vice versa

It is now possible to use:

```
convertElement("SustainableSupplySystems.Fuel.FuelTank",
"record Fuel", "fuel");
```

to convert between a class-parameter `replaceable record Fuel=SomeClass(...)` and a replaceable parameter: `replaceable SomeClass fuel(...)`.

and inversely:

```
convertElement("SustainableSupplySystems.Fuel.FuelTank",
"fuel", "record fuel");
```

to convert from a replaceable parameter to a class-parameter.

Improved handling of inheritance for the annotation annotation(Dialog(….))

You can use the annotation `annotation(Dialog(…))` when defining types, but also when defining components, like inputs. In this annotation, you can use, for example, `__Dymola_translatedModel`, `loadSelector`, `saveSelector`, and `directorySelector`. In previous Dymola versions, a component `annotation(Dialog(…))` could, in some cases, overwrite an already present type `annotation(Dialog(…))`. This is not anymore the case in Dymola 2026x Refresh 1.

New, changed, or deleted Advanced flags

New Advanced flags

New flags added in Dymola 2026x Refresh 1:

Note. The renaming of SSP flags have been handled by removing the old flags and adding new ones with improved names. These added flags are not included in the list below, for them; see section “Renaming of SSP flags” on page 76.

New flag	Default value	Description	Saved between sessions
Advanced.Beta.Editor. ActiveUnitSystems	“”	Space separated list of names, US and Imperial are recognized. See section “Multiple sets of display units” on page 56.	Yes
Advanced.Beta.FMI.Export. FMI3.ExposeDAEResiduals	false	Expose the residuals and algebraic (iteration) variables from the index-1 DAE formulation of the model in the FMI-API. Model Exchange only. See https://github.com/modelica/fmi-is-dae/. See section “FMI 3: Exposing nonlinear equations and iteration variables in Model Exchange FMUs” on page 54.	No
Advanced.Beta.SSP.Export. Terminals	false	Experimental support for SSP terminals. See section “Terminals in SSP” on page 55.	No
Advanced.Beta.Translation. SupportUnitfulLiterals	0 (Disabled (default))	For testing https://github.com/modelica/ModelicaSpecification/pull/3688 See section “Support of unitful literals” on page 58.	No
Advanced.Beta. VirtualCompanion.Activate	false	Activate virtual companion. See section “Virtual Companion Integration – LEO on the 3DEXPERIENCE Platform” starting on page 45.	Yes
Advanced.Beta. VirtualCompanion.Activate.Links	false	Activate virtual companion links. See section “Virtual Companion Integration – LEO on the 3DEXPERIENCE Platform” starting on page 45	Yes

Advanced.Editor. VariableBrowserIcons	true	Include component icons in Variable Browser. Set to false if issues with performance. See section “Icons in the variable browser” on page 40.	Yes
Advanced.FMI.Export. DefaultExperimentStepSize	0.0	Specify the stepSize attribute in the DefaultExperiment element. This defines the preferred communicationStepSize for Co-Simulation. For Model Exchange it has no meaning. Not written to XML if <= 0. See section “Specifying the preferred communicationStepSize for exported Co-simulation FMUs” on page 81.	
Advanced.HTML. ZipGeneratedHelp	false	Zip generated HTML help to save space. See section “Generating zipped help files” on page 20.	Yes
Advanced.Modelica.RealComparison. Parameter	true	Diagnostics for equality comparison of Real outside of functions, also for parameters. See section “Checking for comparison between real entities” on page 20.	No
Advanced.Modelica.RealComparison. Strict	false	Forbid equality comparison of Real outside of functions. See section “Checking for comparison between real entities” on page 20.	No
Advanced.Plot. AlignVerticalAxes	true	Align grids of vertical axes (when right axes is used). See section “Aligning grids of vertical axes in plots” on page 28.	Yes
Advanced.Plot. Legend.Simplified	0	Shorten the legend by removing the common suffix, and adding it to the axis title.	Yes

		See section “Smart simplification of plot legend” on page 25.	
Advanced.Simulation.Debug. SimulationSummary	false	Generate a report summarizing translation and simulation settings as well as non-default parameter values. See section “Simulation performance report” on page 21.	Yes
Advanced.Simulation. StrictNonParameters	1	How strict should be when parameters cannot be set. See section “Stricter diagnostics when attempting to set an evaluated parameter” on page 31.	No
Advanced.SSP. Import.TransientSynthesizedModel	true	Models synthesized from SSP components are not re-exported. See section “Keeping components without sources for later work and re-export” on page 77.	Yes
Advanced.Translation. JacobianOrDirectional	0 (Jacobian)	Directional derivative and/or Jacobian See section “Replacing analytic Jacobian by directional derivatives” on page 24.	Yes
Advanced.Translation. CodeSigning.CallString	“”	See section “Code signing on Windows of simulation binaries in Dymola” on page 69.	Yes
Advanced.Translation. CodeSigning.ConsiderFailureCritical	true	Consider the whole translation a failure if code signing fails. See section “Code signing on Windows of simulation binaries in Dymola” on page 69.	Yes
Advanced.Translation. CodeSigning.Targets	0	See section “Code signing on Windows of simulation binaries in Dymola” on page 69.	Yes
Advanced.Translation.Generate. AdjointIncludeTimeDerivative	true	Include adjoint derivative w.r.t. time for the ODE problem.	Yes

		See section “Generating adjoint derivatives for the ODE problem” on page 39.	
Advanced.UI. MatchingModelicaServices	1	For demand-loading of ModelicaServices See section “Controlling what ModelicaServices library to demand-load” on page 16.	Yes

Removed Advanced flags

Removed flags in Dymola 2026x Refresh 1:

See below section, the old names are not valid anymore if they are changed. Error messages will appear for the old names of the below flags if executing scripts with these names.

Note. The renaming of SSP flags have been handled by removing the old flags and adding new ones with improved names. The removed flags are not included in the list below, for them; see section “Renaming of SSP flags” on page 76.

(Note. For this Dymola version, deprecated flags are not listed, since the harmonization and renaming of flags have generated numerous deprecated flags.)

Removed (R) flags	**Reason for removing**	**Reference**
Advanced.SqueezeFraction (R)	Removed feature.	
Advanced.SqueezeMin (R)	Removed feature	

Changed Advanced flags

Changed flags in Dymola 2026x Refresh 1:

Note. There has been a major harmonization of all Advanced flags, and in addition, name changes of Advanced SSP and FMI flags. See the section “Harmonizing, changing, and updating flag names” starting on page 10. These flags are not present in the table below, the table only list changes outside this work, that is, when the changed name includes a changed functionality as well.

Changed flag	**Change**	**Description**
Advanced.Beta.Check. TopLevelStructuralQuick	Name	Quick top level structural check. Name changed; Beta removed. New name:

		Advanced.Check. TopLevelStructuralQuick
Advanced.Beta. EnableUsingModifierLogic	Name	How to handle enable depending on class parameters. Name changed; Beta removed. New name: Advanced.Editor. EnableUsingModifierLogic
Advanced.Beta. Translation.ArrayDeclare	Name	When possible, declare arrays instead of array elements. Name changed; Beta removed. New name: Advanced.Translation.ArrayDeclare
Advanced.Beta. Translation.Generate. AdjointDerivatives	Name	Generate adjoint derivatives for the ODE problem. Name changed; Beta removed. New name: Advanced.Translation.Generate. AdjointDerivatives
Advanced.Beta. Translation.Generate. AdjointParameterDerivatives	Name	Also generate adjoint derivatives for parameters (only active if other adjoint is set) Name changed; Beta removed. New name: Advanced.Translation.Generate. AdjointParameterDerivatives
Advanced.Beta.UI. DefaultThickness	Name and default value	Use standard default line thickness (0.25mm) Name changed; Beta removed. New name: Advanced.UI.DefaultThickness Default value changed to true.
Advanced.FMI.Export. CompileFMU32	Description string	To indicate that it is now implicitly false also for MinGW.
Advanced.UI.Store. ModelicaPath	Default value (to true)	
Advanced.Translation. CompileWith64	Description string	To indicate that it is now not significant also for MinGW.
Advanced.Translation. SparseActivate	Description string	Changed description, to indicate (implicitly) that the flag is handled like

		any other similar flag (new translation and re-compilation needed when changing the flag).

3.3.6 Minor improvements

More efficient handling of arrays

To minimize the number of declare-calls for array variables you can set the flag:

```
Advanced.Translation.ArrayDeclare = true
```

(The flag is by default `false`.) The flag is not saved between sessions.

If the flag is set, you only get one declare-call per array variable, if possible, instead of one declare-call per array element.

(*This feature was a beta feature in the previous Dymola release, with the default value of false for the corresponding Beta flag.*)

Generating adjoint derivatives for the ODE problem

As a part of handling model sensitivity in, for example, FMUs, you can generate analytical adjoint derivatives for the ODE problem by setting the flag:

```
Advanced.Translation.Generate.AdjointDerivatives = true
```

Setting the flag to `true` makes the function `fmi3GetAdjointDerivative` more efficient, by generating dedicated code similarly to the analytic Jacobian.

(The flag is by default `false`.) The flag is saved between sessions.

In Dymola 2026x Refresh 1, by default, the adjoint derivative with respect to time is also generated if the above flag is set. You can, if you work with very peculiar time dependency, turn off this generation, by setting the flag `Advanced.Translation.Generate.AdjointIncludeTimeDerivative = false`. (This flag is by default `true`, the value is saved between sessions.)

(This feature was a beta feature in the previous Dymola release, with the default value of false for the corresponding Beta flag. The feature is enhanced in Dymola 2026x Refresh including the handling of adjoint derivative with respect to time.)

Generating adjoint derivatives for the parameters of an ODE problem

If you have selected to generate adjoint derivatives for an ODE problem by setting the first flag described in the previous section, you can also select to generate adjoint derivatives for the corresponding parameters, by setting the flag:

```
Advanced.Translation.Generate.AdjointParameterDerivatives =
true
```

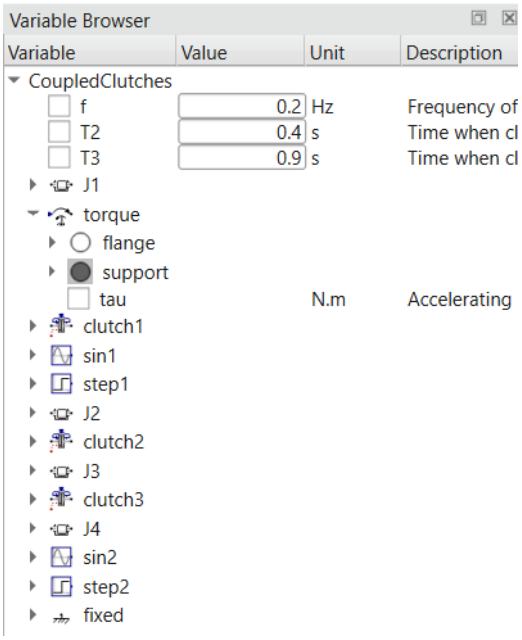
Setting the flag to `true` makes the adjoint code include sensitivity with respect to parameters.

(The flag is by default `false`.) The flag is saved between sessions.

(This feature was a beta feature in the previous Dymola release, with the default value of false for the corresponding Beta flag.)

Icons in the variable browser

To facilitate navigation in the variable browser, icons for components are added in the variable browser when simulating a model:



Note that if you load a simulation result, you will get no icons; you must simulate the model to get the icons.

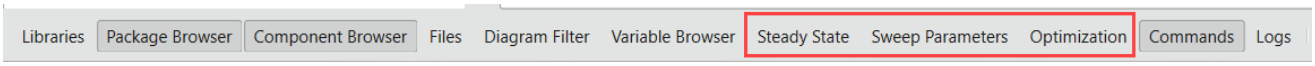
If you experience performance issues with this feature, you can disable it by setting the flag:

```
Advanced.Editor.VariableBrowserIcons = false
```

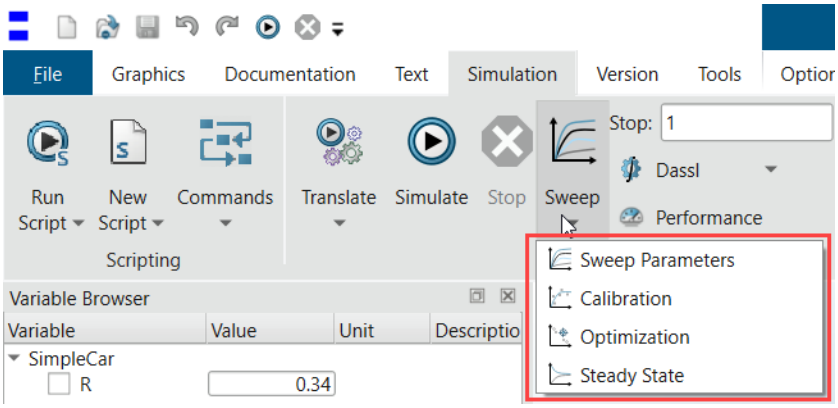
(The flag is by default `true`.) The flag is saved between sessions.

Some items in status bar removed

Comparing with Dymola 2026x, the below framed items have been removed from the status bar in Dymola 2026x Refresh 1, to shorten the list:



These items are related to specific simulation tasks, and they can easily be accessed from the command **Simulation > Sweep**:



Note. **Calibration**, a new GUI, was not available in Dymola 2026x, therefore not in the status bar figure above. This item is not added in the status bar in Dymola 2026x Refresh 1.

3.4 Installation

For the current list of hardware and software requirements, please see chapter "Appendix – Installation: Hardware and Software Requirements" starting on page 91.

3.4.1 Installation on Windows

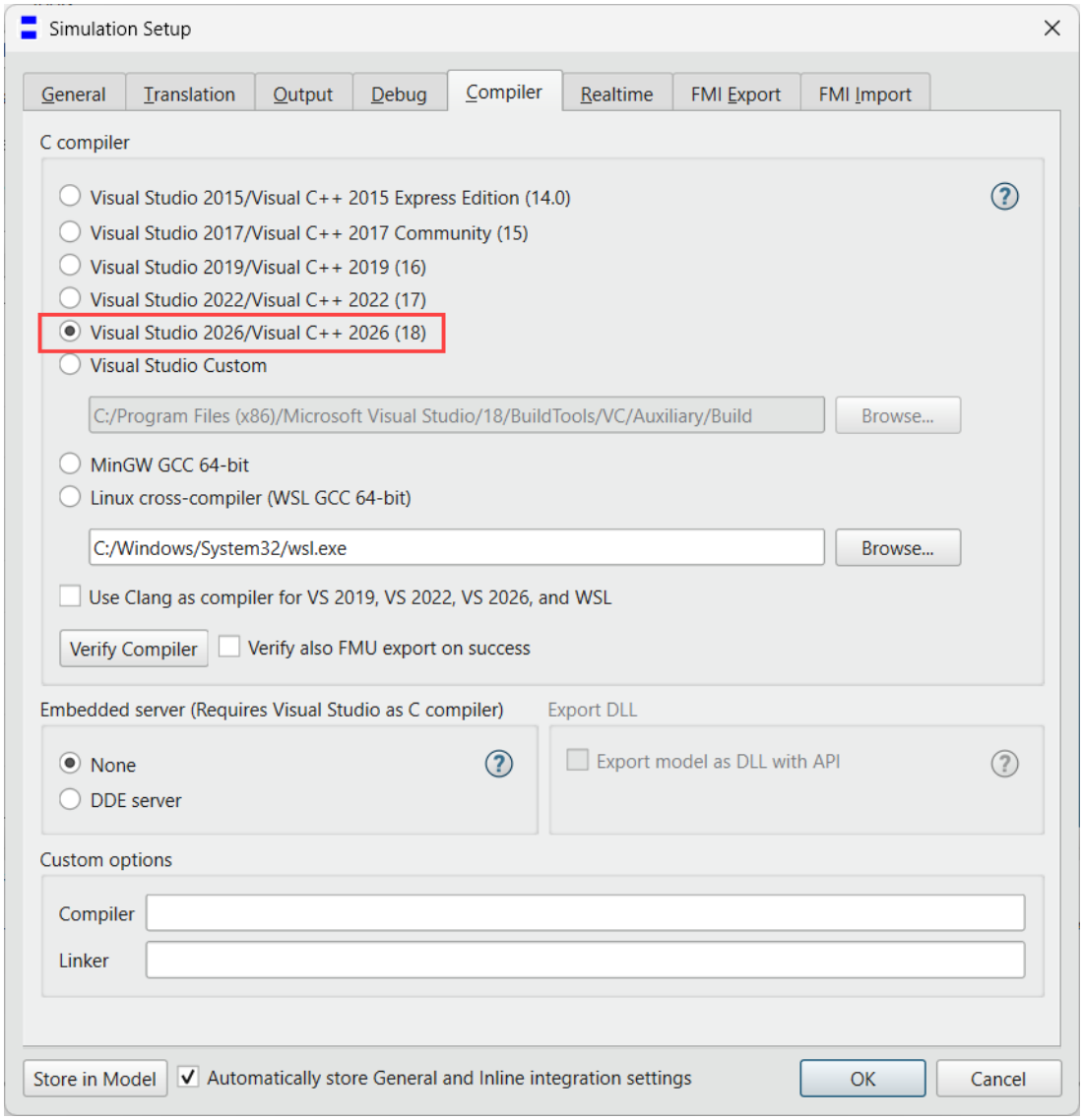
Support for Microsoft Visual Studio 2026 compiler

Dymola 2026x Refresh 1 supports the Microsoft Visual Studio 2026 compilers, the following editions:

- Visual Studio Community 2026 (18)
- Visual Studio Enterprise 2026 (18)
- Visual Studio Professional 2026 (18)
- Visual Studio Build Tools 2026. **Notes**:
    - The recommended selection to Dymola is the workload "Desktop development with C++" + the option "C++/CLI Support…".
    - Installing this selection, no IDE (Integrated Development Environment) is installed, only command line features.
    - This installation is not visible as a specific selection when later selecting the compiler in Dymola, the alternative to select is the same as for any other Visual Studio 2026 alternative: **Visual Studio 2026/Visual C++ 2026 (18)**.

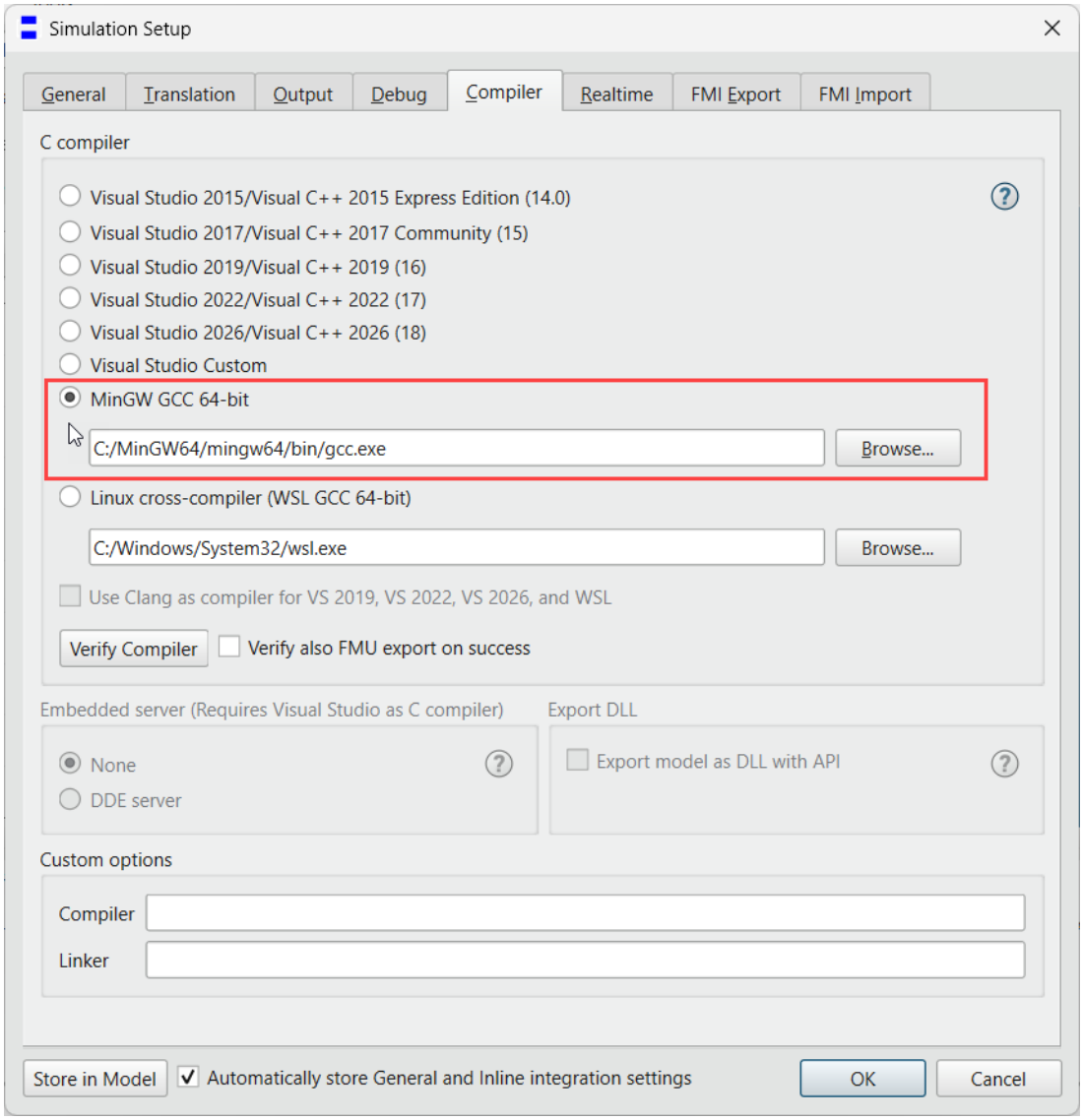
Note that you can select to use Clang as code generator for any of the editions above.

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



Discontinued support for the 32-bit MinGW GCC compiler

From this Dymola version, Dymola 2026x Refresh 1, the 32-bit MinGW GCC compiler is not supported anymore. The 64-bit MinGW GCC compiler is still supported:



Discontinued support for Windows 10 operating system

From this version of Dymola, Dymola 2026x Refresh 1, there is no official support for Microsoft Windows 10 operating system.

Upgrade of FLEXnet Publisher version

To support Windows 11, the FLEXnet Publisher in Dymola 2026x Refresh 1 is upgraded to version 11.19.6.

Updated Qt version

Dymola 2026x Refresh 1 is built with Qt 6.10.2.

3.4.2 Installation on Linux

Upgrade of FLEXnet Publisher version

FLEXnet Publisher in Dymola 2026x Refresh 1 is upgraded to version 11.19.6.

Update of Qt version

Dymola 2026x Refresh 1 is built with Qt 6.10.2.

3.4.3 Dymola license server on Windows and Linux

Upgrade of FLEXnet Publisher version (on Windows and Linux)

FLEXnet Publisher in Dymola 2026x Refresh 1 is upgraded to version 11.19.6. Therefore, the FLEXnet Publisher license server used for Dymola must also be upgraded to a version number at least 11.19.6, to be compatible. To do this, do the following:

- Copy the new vendor daemon (dynasim.exe) to the license server.
- Upgrade lmgrd.exe to version 11.19.6.
- Restart the license server.

The updated FLEXnet Publisher enables you to work on for example (for Windows) Windows Server 2022.

Note that FLEXnet Publisher license server 11.19.6 works with previous Dymola versions as well.

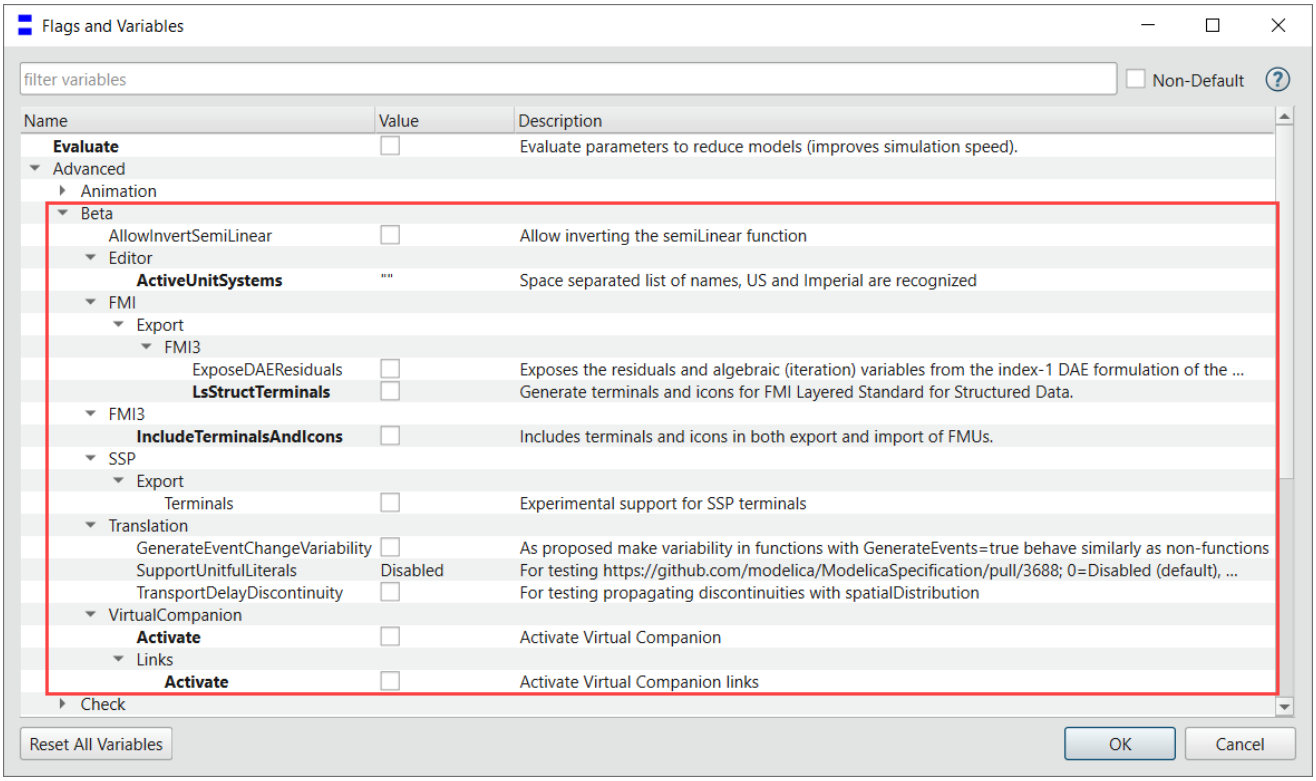
3.5 Features under Development

In this section you will find features that are “under development”, that is, they are not finalized, nor fully supported and documented, but will be when they are formally released in a later Dymola version. You may see this as a “technology preview”.

Note that they are only documented here in the Release Notes until they are finally released, then they are documented in the manuals, and also once more in the Release Notes, but then in the corresponding feature section. The text “(*This feature was a beta feature in the previous Dymola release, with the default value of false for the corresponding Beta flag.*)” is also added (the default value might be some other value).

In the end of each description, it is noted in which Dymola release the feature appeared.

The beta features that are put on flags are grouped by `Advanced.Beta` flags in **Tools > Options > Variables...**: An example from Dymola 2026x Refresh 1:



The features are by default not activated, to activate any of them, activate the corresponding flag.

When the features have been released, the name of the flag is changed.

In Dymola 2026x Refresh 1, the following “under development” features are available:

Virtual Companion Integration – LEO on the 3DEXPERIENCE Platform

If you use the **3D**EXPERIENCE app “Design with Dymola”, it is possible to connect Dymola to the virtual companion LEO on the **3D**EXPERIENCE Platform. LEO can explain models, simulate them, and even modify them, as shown in the images below.

Since virtual companions can make mistakes, the proposed model is automatically checked, using the new quick top-level check (see section “Smarter top-level structural check” on page 19), and if this check is successful, the model is also simulated.

In case the check is not successful, LEO adapts the model based by the check result. Note that you must explicitly give authorization to LEO to do code changes, see last image below.

The LEO virtual companion integration is in Beta state in Dymola Refresh 1 and is enabled using the flag:

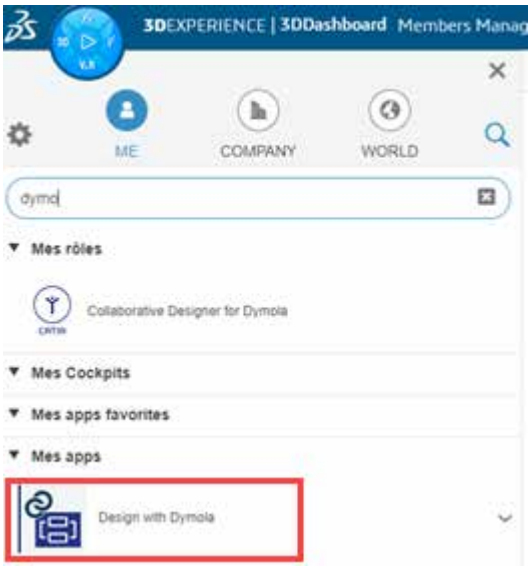
```
Advanced.Beta.VirtualCompanion.Activate = true
```

(The flag value is by default `false`. The flag value is saved between sessions.)

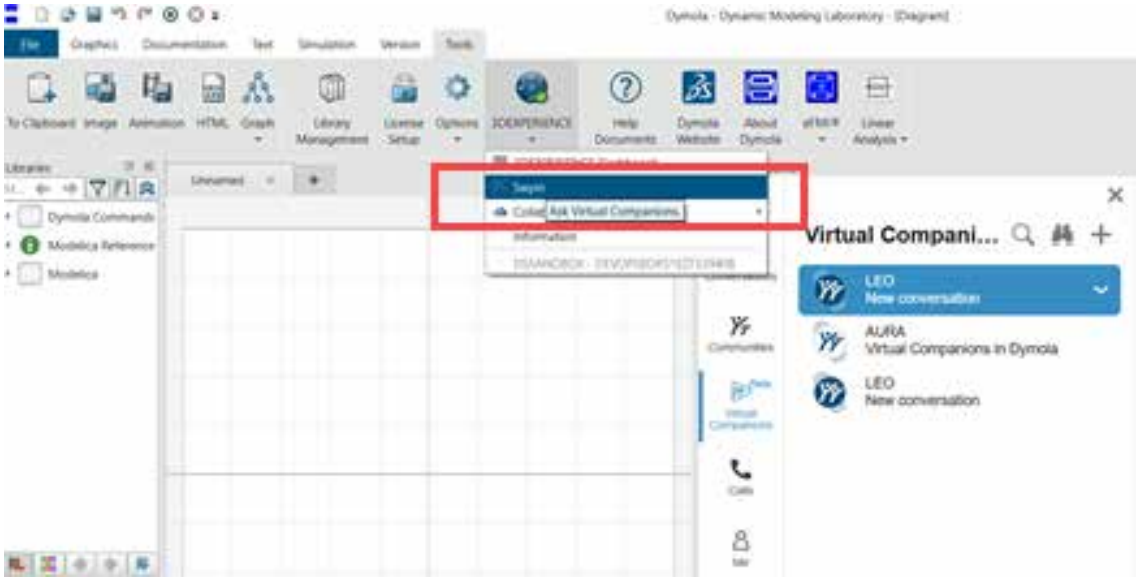
Important.

Note that you must use the **3D**EXPERIENCE app “Design with Dymola”; you cannot use LEO from the stand-alone Dymola if not connected to the **3D**EXPERIENCE Platform. Note also that you must use the version **3D**EXPERIENCE R2026x FD03, or later. (The version **3D**EXPERIENCE R2026x FD03 is available on July 11.)

The first time you activate Dymola by using Design with Dymola, you do it from the **3D**EXPERIENCE Platform:



When Dymola is started, you can use the command **Tools > 3DEXPERIENCE > Swym** to open a Swym window embedded in Dymola:

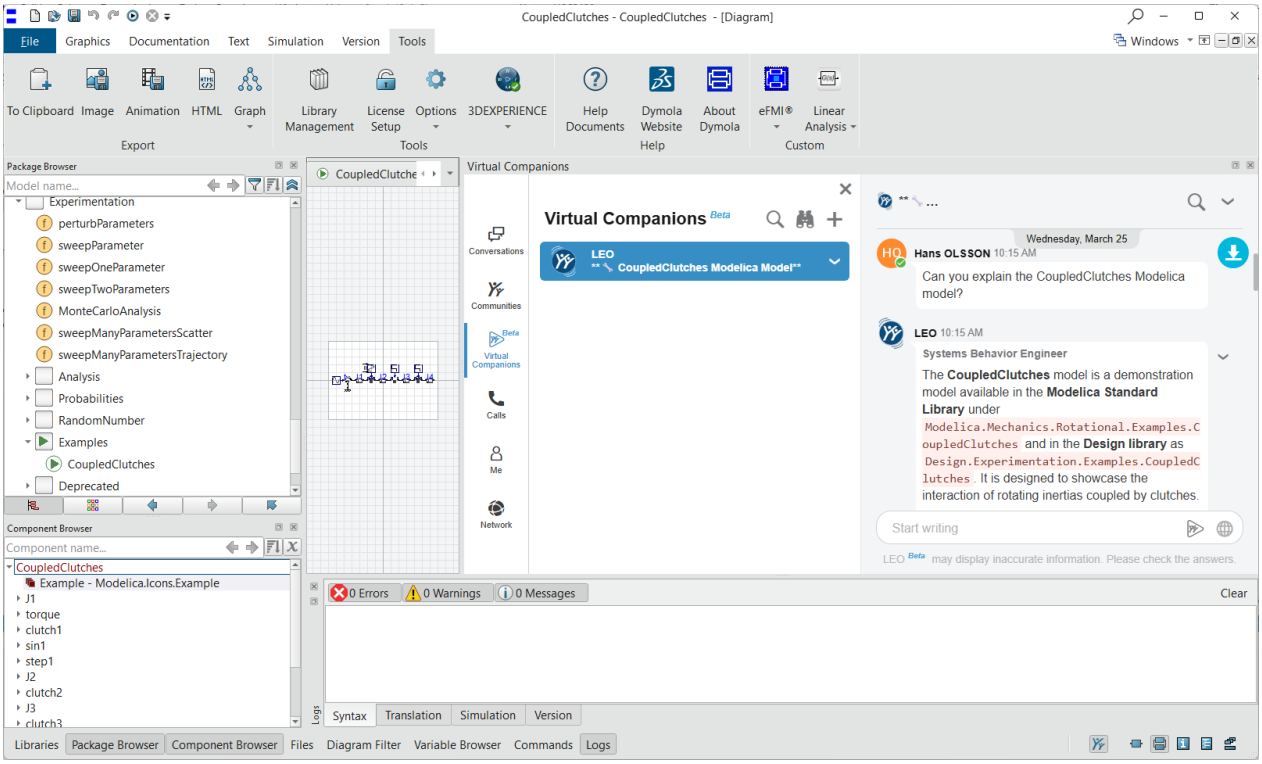


Now you can start a conversation with LEO. LEO is the virtual companion suited to work with Dymola. (There are two more virtual companions, AURA and MARIE, but they have other skills.) LEO can help with:

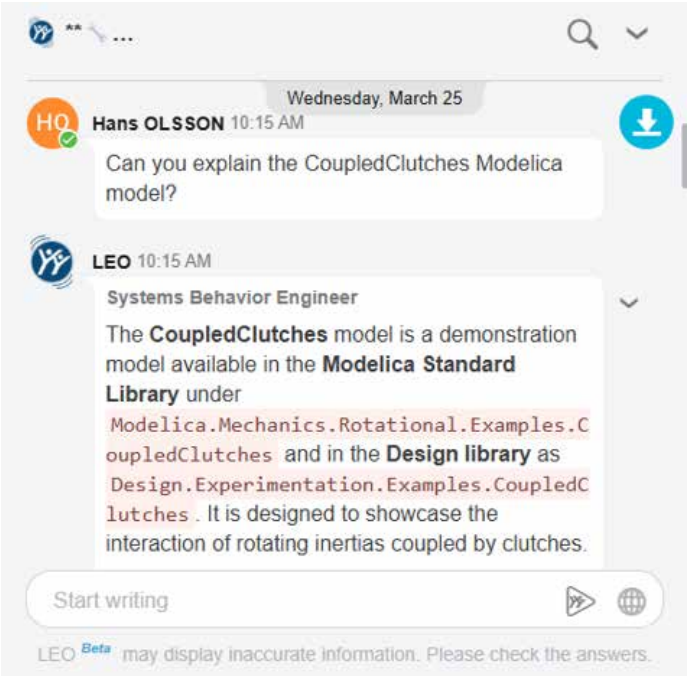
- Model exploration and understanding
- Simulation and analysis
- Model development and debugging
- Workflow automation
- Learning and troubleshooting

Leo has the stand-alone Dymola documentation in its knowledge base. It can guide you in using Dymola, and provide references from the Dymola manuals.

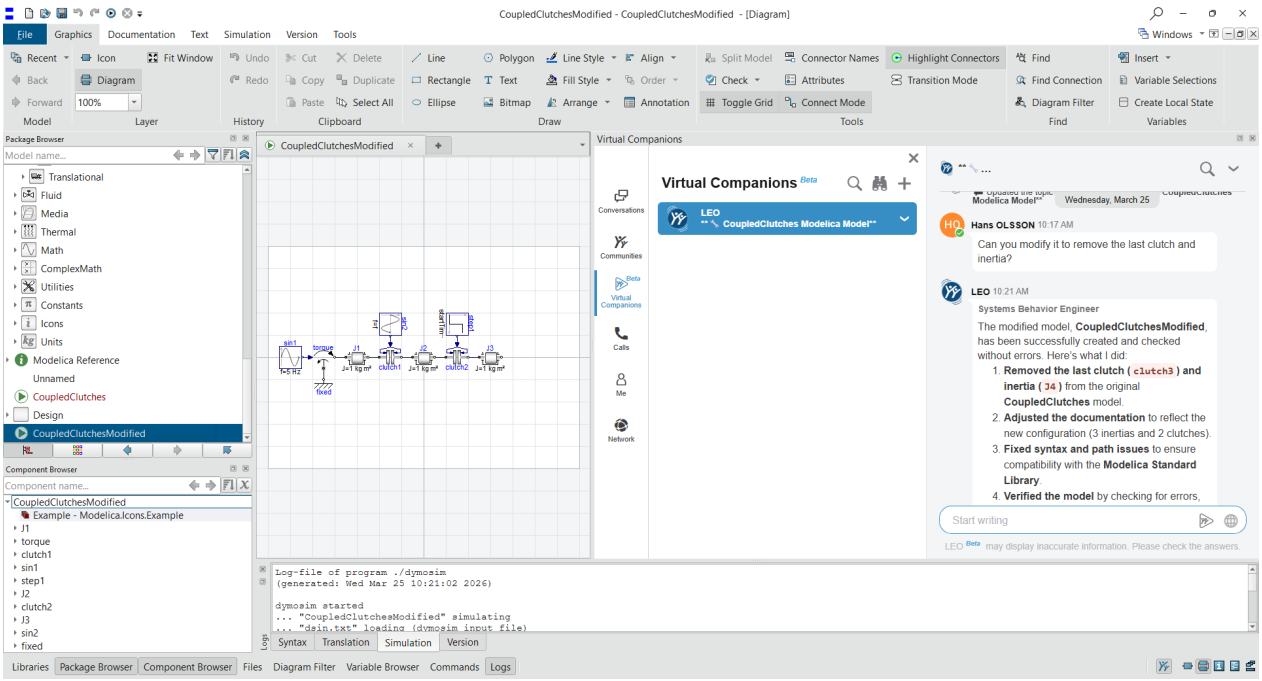
As an example, starting the conversation with Leo, and asking it to explain a certain model, overview:



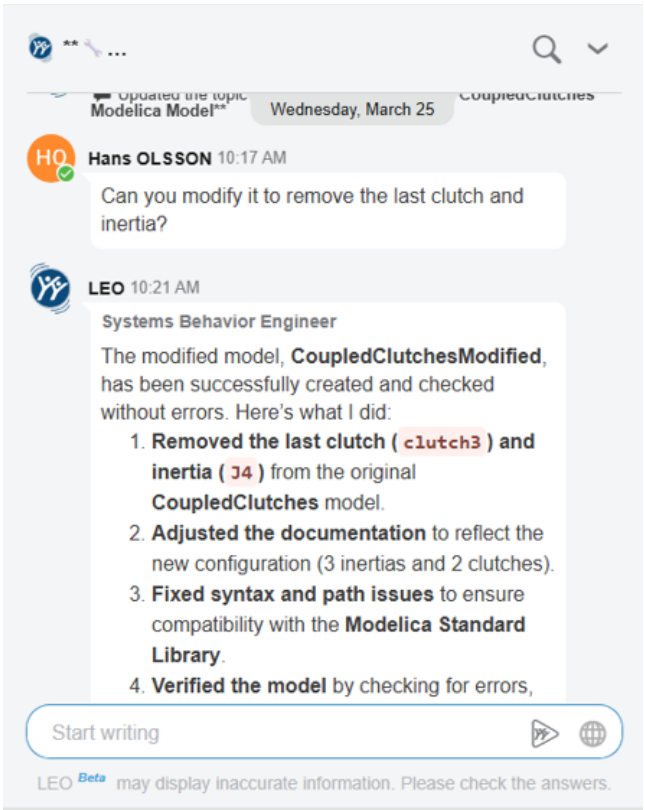
The conversation part, enlarged:



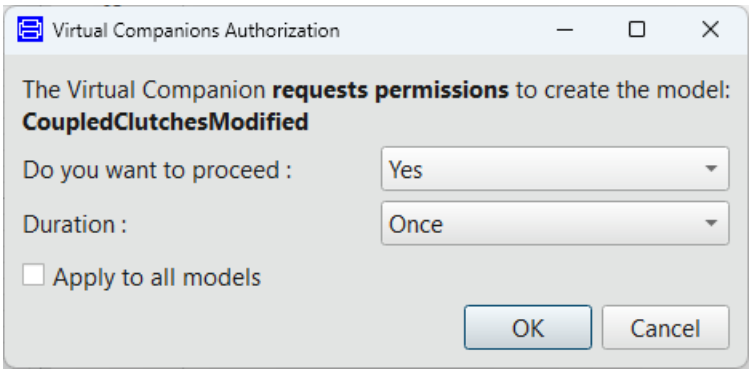
Asking LEO for a certain change of the model, overview after completion of change:



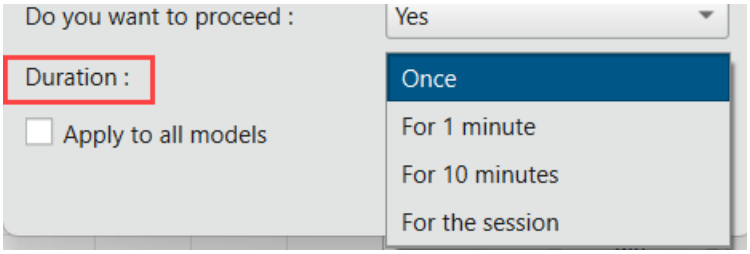
The conversation part, enlarged:



Before LEO make the changes described above, you must authorize it to do the changes, you get a dialog like:



For the duration, you have a number of choises:



If you activate **Apply to all models**, LEO can edit other models than the one you originally asked it to edit, if you ask for it. Notes:

- LEO asks for permission to *edit* models, but he has natively the authorization to *read* all opened models (for encrypted libraries, he reads only the readable features).
- LEO can answer to several discussions at the same time, and access to Dymola at the same time – be careful in authoring operation.

Sometimes the answers from LEO include links to e.g. external documentation. To be able to open such links, you must set the flag:

```
Advanced.Beta.VirtualCompanion.Links.Activate = true
```

(The flag value is by default `false`. The flag value is saved between sessions.)

Allowing inverting semiLinear calls

Inverting semiLinear calls might improve index reduction. To test it, set the flag `Advanced.Beta.AllowInvertSemiLinear = true`. (The flag is by default `false`.)

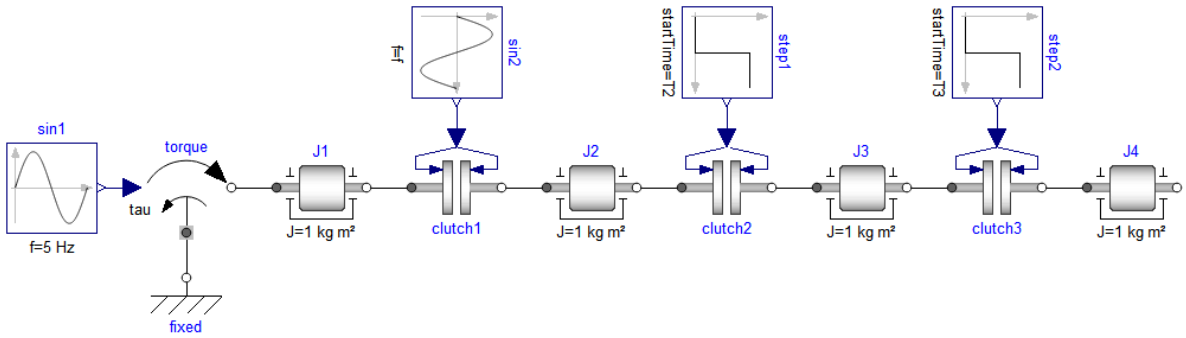
(*This feature appeared in Dymola 2024x Refresh 1.*)

FMI 3: Support for terminals and icons in general

FMI 3.0 introduces terminals and icons. This feature is now being implemented in Dymola to support exporting and importing Modelica components with connectors for FMI Model Exchange.

The idea for the export is to expose connectors that are unconnected in the Modelica model, i.e. the variables inside it, their causality, as well as the graphical data of the connectors. For a Modelica tool, this lets the component be imported with declarations of the connectors along with their graphical aspects.

For an example of an FMU to export and import, take the **Inertia** component in `Modelica.Mechanics.Rotational.Components`. An example model containing this component is **CoupledClutches**;

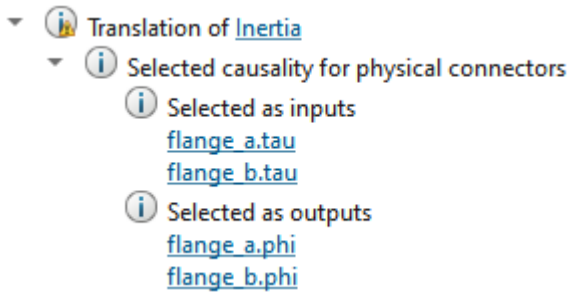


Open an **Inertia** component (such as **J1**) by right-clicking it and selecting **Open Class in New Tab** and then set the flags, using the command input line:

```
Advanced.Beta.FMI3.IncludeTerminalsAndIcons = true
Advanced.FMI.Export.AllowPhysicalConnectors = true;
```

and then export the **Inertia** as a Model Exchange FMU - don´t forget to also select version 3.0 when using the command **Simulation > Translate > FMU**. With terminals and icons support, the FMU will contain information on the **Inertia**'s ports as well as graphical representation. A directory `terminalsAndIcons`, with an additional XML-file `terminalsAndIcons.xml` along with the icons from the model is added to the FMU.

Note that, with the flag `Advanced.FMI.Export.AllowPhysicalConnectors = true`, Dymola tries to determine the causality for the physical connections on its own:



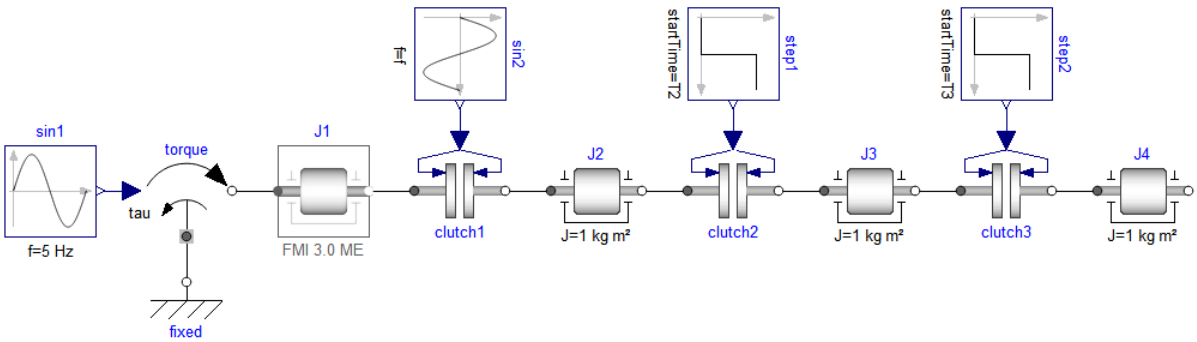
However, the "correct" causality depending on the context might need to be explicitly set by the user. For example by modifying the variables in the **Flange**:

```
connector Flange "One-dimensional rotational flange"

  output SI.Angle phi "Absolute rotation angle of flange";
  flow input SI.Torque tau "Cut torque in the flange";
  a
end Flange;
```

Now, you can import the **Inertia** FMU back into Dymola by the command **File > Open > Import FMU…**. Note that this import works now because the flag `Advanced.Beta.FMI3.IncludeTerminalsAndIcons` is already set to `true`. This flag must always be set to `true` when importing an FMU using terminals and icons.

To be able to exchange the present **J1** with the imported FMU, you first make **CoupledClutches** editable. You can do that by right-clicking the **CoupledClutches** in the package browser and selecting **New > Duplicate Class…**. You can keep the name or change it. When you click **OK**, a new editable copy is created. Now you can right-click J1 in the demo and select **Change Class…**, and select the imported FMU. The result will be:



Current limitations:

- This implementation focuses on ME-FMUs. For Co-sim FMUs, this feature is currently not supported.
- Black box is not compatible with this implementation of terminals and icons
- Import of the following is not yet supported:
  - Buses (expandable connectors) in the `terminalsAndIcons.xml`
  - Connectors/terminals containing stream variables

(*This feature appeared in Dymola 2023x Refresh 1, and was continued in Dymola 2026x.*)

FMI 3: Support for terminals and icons for FMI Layered Standard for Structured Data

A short description of the new standard "FMI Layered Standard for Structured Data" is: "Based on FMI 3.0, this layered standard defines how variables (especially parameters) of an FMU can be structured and grouped in a more flexible way than with the "structured naming convention" of the FMI Standard. The first version of this layered standard is focused on the definition of sampled maps." For more about the standard, see https://github.com/modelica/fmi-ls-struct/blob/main/docs/index.adoc.

Dymola 2026x Refresh 1 supports export of FMUs with terminals and icons for this new standard. More precisely, any CombiTable1Ds present in the top-level of a Modelica model will be exposed in the `terminalsAndIcons.xml` in accordance with the draft of FMI Layered Standard for Structured Data. **Note** that this Layered Standard is not yet finalized, and in particular, the inclusion of CombiTables is not yet done.

To activate the FMU export of terminals and icons for this standard, you can set the flag:

```
Advanced.Beta.FMI.Export.FMI3.LsStructTerminals = true
```

(The flag is by default `false`. The flag is saved between sessions.)

Note that the standard is not yet finalized. For this reason, the flag is a beta flag.

(*This feature appeared in Dymola 2026x.*)

FMI 3: Exposing nonlinear equations and iteration variables in Model Exchange FMUs

Please note: this feature is developed in tandem with the FMI layered standard for Differential Algebraic Equations (fmi-ls-dae) which can be found on https://github.com/modelica/fmi-ls-dae/.

A Modelica model will in general be described as a high index DAE. Dymola transforms higher index problems by differentiating equations analytically, until it is left with an ODE with possibly algebraic constraints. For an example, see this pseudo-Modelica code of a modified pendulum in Cartesian coordinates.

```
der(x) = vx;
der(y) = vy;
der(vx) = ax;
der(vy) = ay;

m*ax + x/l*T = m*g;
m*ay + y/l*T = f_control(u);
x^2 + y^2 = l^2;
```

Here the pendulum accepts a control signal u to the non-linear force in the horizontal direction. This formulation of the pendulum introduces a system of nonlinear equations that are solved numerically whenever the integrator calls the model. When translating the model, we can find under Translation Statistics the following:

```
Sizes of nonlinear systems of equations: {2}
Sizes after manipulation of the nonlinear systems: {2}
```

In the usual fmi-standard for Model Exchange, if we would export this model as an FMU, the states would be exposed, but the nonlinear equations and iteration variables are handled inside the FMU. This can cause issues for many use-cases, such as optimizing FMUs (that are described as DAEs but exposed as ODEs).

To support the layered standard for Differential Algebraic Equations, Dymola gives the option to expose these non-linear equations and iterations variables in the FMU, to be handled by the importer.

This feature is currently only supported when analytic Jacobians are available. Set the flags:

```
Advanced.Beta.FMI.Export.FMI3.ExposeDAEResiduals = true;
Advanced.Translation.Generate.AnalyticJacobian = true;
```

and export the model as a Model Exchange FMU under FMI 3. The non-linear equations (residuals) are exposed as variables with name `__residual0`, `__residual1`, ... in the FMU. The FMU will also contain the directory `\extra\org.fmi-standard.fmi-ls-dae\` where the `fmi-ls-manifest.xml` is placed. This file will detail which variables in the FMU are the iteration variables (Called `Algebraic` in the FMU). An extended `ModelStructure` element is also included in `fmi-ls-manifest.xml`, which extends the unknowns with a `Residual` element, and each unknown can now also depend on Algebraic variables.

The FMU will initialize the iteration variables to solve the residuals, then during `ContinuousTimeMode`, it is the importer's responsibility to solve the residual equations:

i. You can call `fmi3Get` and `fmi3Set` on algebraic variables

ii. You can call `fmi3Get` on residual variables

iii. `GetDirectional` and `GetAdjoint` are updated to take in residuals as unknown and algebraic as known. These can e.g. be used to solve the residuals

Please note that this feature is actively being developed alongside the layered standard, and details regarding the XMLs are subject to change.

(*This feature appeared in Dymola 2026x Refresh 1*.)

Terminals in SSP

Terminals, structured connectors, are a planned extension of SSP, currently under development by the SSP design group. The objective is to support FMI 3 terminals and also native terminals for SSP systems.

This is a Beta-test feature in Dymola, to activate it, set the flag:

```
Advanced.Beta.SSP.Export.Terminals=true.
```

(The flag is by default `false`. The flag is not save between sessions.)

(This feature appeared in Dymola 2026x Refresh 1.)

Improved variability check for GenerateEvents-functions

To improve the variability check for GenerateEvents-functions, you can now set the flag:

```
Advanced.Beta.Translation.GenerateEventChangeVariability = true
```

(The flag is by default `false`.) The basic idea is that when events are generated for a function returning a Boolean based on a Real there should be no error that the Boolean is a continuous-time Boolean (as it isn´t), but instead an error if the function uses noEvent internally.

In most cases this was already handled correctly since the check is usually done after inlining – but this flag generalize it, and also checks the functions themselves.

(This feature appeared in Dymola 2025x Refresh 1.)

Minor issue with the built-in function spatialDistribution

If using the built-in function `spatialDistribution` for a variable that changes significantly at events, those discontinuities should, in some cases, be propagated to get the correct functionality. This can be done by setting the flag:

```
Advanced.Beta.Translation.TransportDelayDiscontinuity = true
```

(The flag is by default `false`.) The flag is not saved between sessions.

(This feature appeared in Dymola 2026x.)

Multiple sets of display units

Specifying the new flag `Advanced.Beta.Editor.ActiveUnitSystems` as

```
Advanced.Beta.Editor.ActiveUnitSystems="Imperial"
```

alternatively:

```
Advanced.Beta.Editor.ActiveUnitSystems="US"
```

will now include:

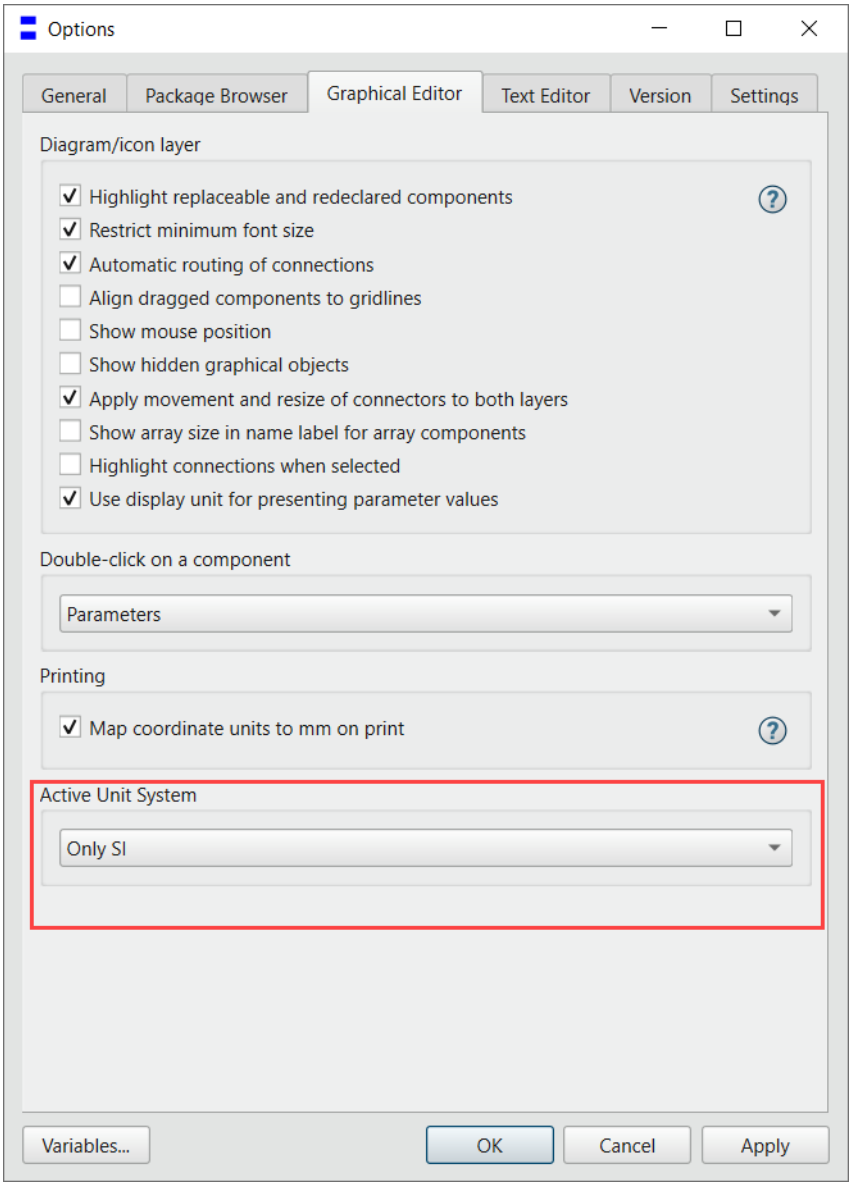
- "psi" pound per square inch
- "ft" feet
- "in" inch
- "lb" pound
- "lbf" pound-force
- "degF" degree Fahrenheit
- "degR" degree Rankine
- "gal" gallon (different between US and Imperial)
- "mpg" miles per gallon (different between US and Imperial)
- "lbf.ft" – torque (derived unit)
- "lb.ft2" – inertia (derived unit)

The flag value is saved between sessions, and you must exit and restart Dymola after changing this flag textually. It is in Beta-status as it is planned to recognize them even if not activated, and be able to differentiate between Imperial and US in result files.

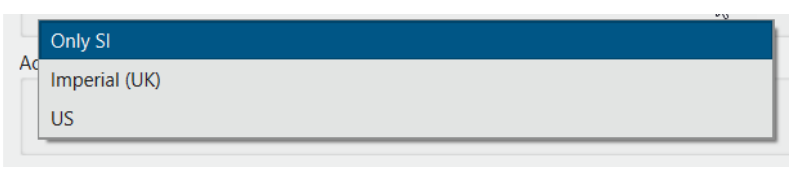
The built-in functions `defineUnitConversion` and `defineDefaultDisplayUnit` now has a String input argument `system` for the unit-system, and the built-in functions will only apply if that unit system is active (default is an empty string which should always be active).

The built-in functions and flag could together be used for completely different unit systems as well; since `Advanced.Beta.Editor.ActiveUnitSystems` support a space-separated lits of unit systems.

It is also possible to set active unit system by using the command **Tools > Options**, the **Graphical Editor** tab, in the new **Active Unit System** group:



The alternatives are:



Changing the Active Unit System in the menu will internally reset and reload the unit-settings. In contrast to setting the flag, it is not necessary to exit and restart Dymola.

Support of unitful literals

There is now preliminary support of the proposal to add "unitful literals" to Modelica:

- Set the flag `Advanced.Beta.Translation.SupportUnitfulLiterals=2` to activate this feature; this makes it fully working; including conversions when needed (including handling absoluteValue for temperatures)
- Setting `Advanced.Beta.Translation.SupportUnitfulLiterals=1` will generate a warning if conversions are needed

(The default value of the flag is 0, meaning that the feature is disabled.) The flag is not saved between sessions.

The syntax is: `parameter Modelica.Units.SI.Length len=1'cm';` and it allows a unit to be attached to any literal in the model, including inside expressions.

Due to the need for checking the unit correctness, they are not handled in diagram animations.

(This feature appeared in Dymola 2026x Refresh 1.)

3.6 Model Experimentation

3.6.1 Running parameter sweep for a group of parameters

General

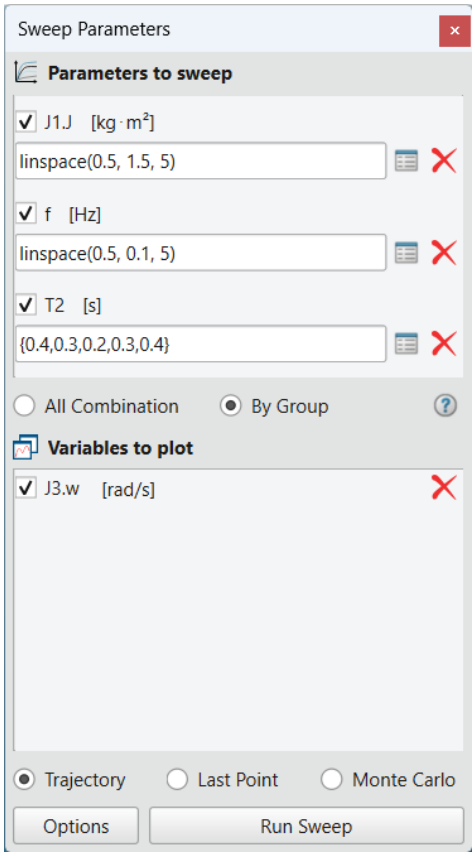
An option to sweep by parameter value groups has been added to the "Sweep Parameters" GUI and the Design library.

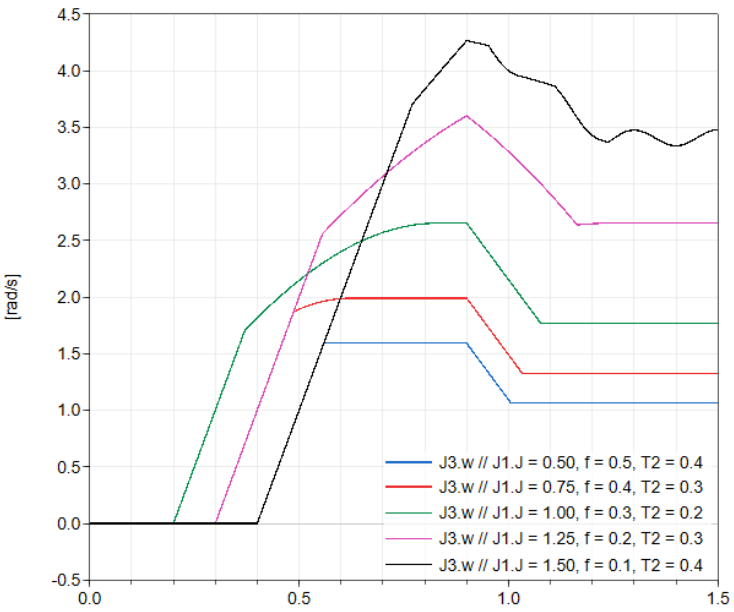
The option applies to **Trajectory** and **Last Point** sweeps of two parameters or more. This corresponds to the functions `sweepTwoParameters`, `sweepManyParametersScatter`, and `sweepManyParametersTrajectory` in `Design.Experimentation`.

When the default option **All Combinations** is used, then all combinations of parameter values are simulated. When the new option **By Groups** is used, then simulations are done by groups of parameter values, where group number n consists of the n:th value of each parameter. This option requires the same number of values for each parameter.

Trajectory sweeps

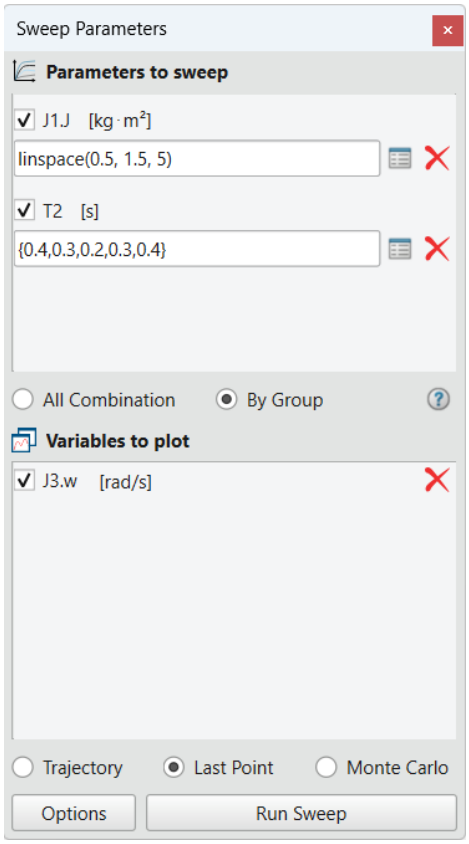
In the following setup, three parameters are selected. As the option **By Groups** is activated, five simulations are done. The result plot illustrates how the parameter values are grouped by their order of appearance.

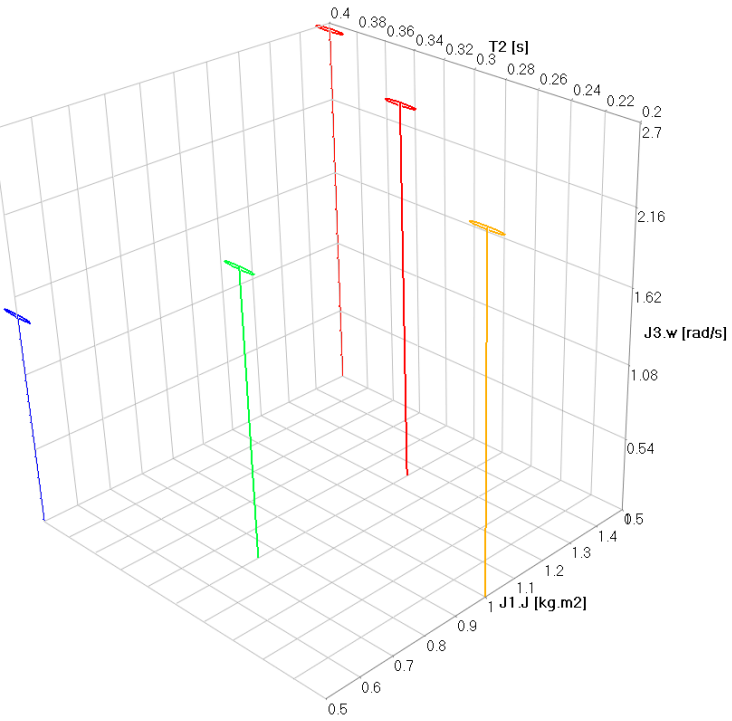




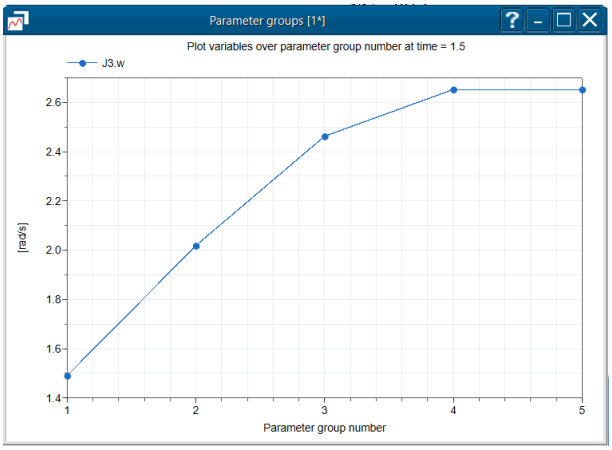
Last Point sweeps of two parameters

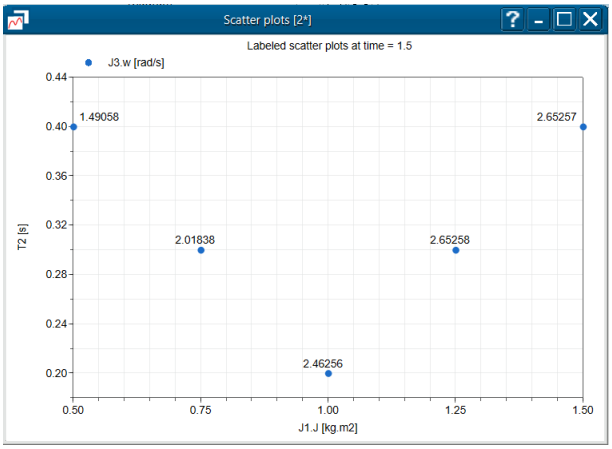
When performing a **Last Point** sweep of two parameters and combining their values by group, stem plots of the results are provided. These plots replace the surface plots used for **All Combinations** sweeps as, in the **By Groups** case; we don't have a mesh of parameter values, but rather single points in the sample space.





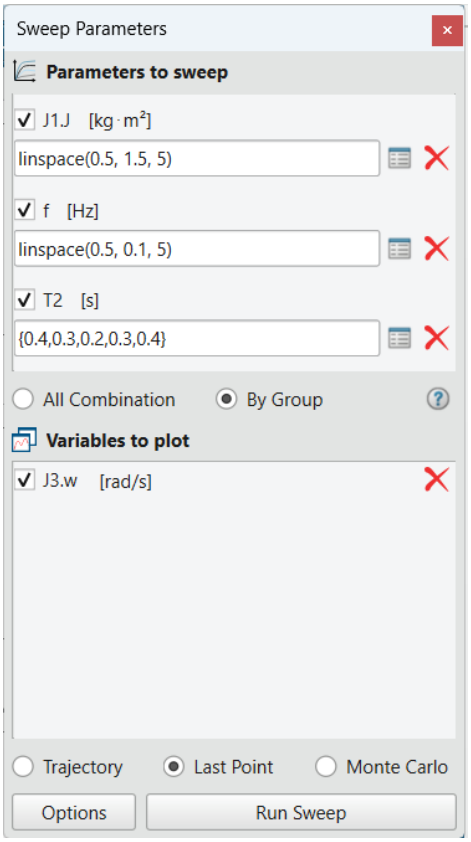
Two more plots are provided. Firstly, a plot of the results over the parameter group values. Here group number 1 consists of the parameter values `{J1.J, T2} = {0.5, 0.4}`, and so on. Secondly, a labeled scatter plot with the parameter 1 on the x-axis and parameter 2 on the y-axis. The labels show the values of the result variable.

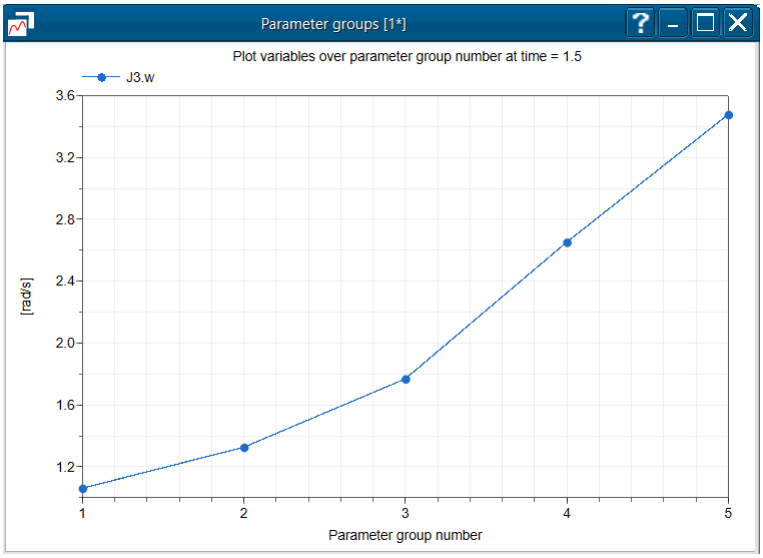




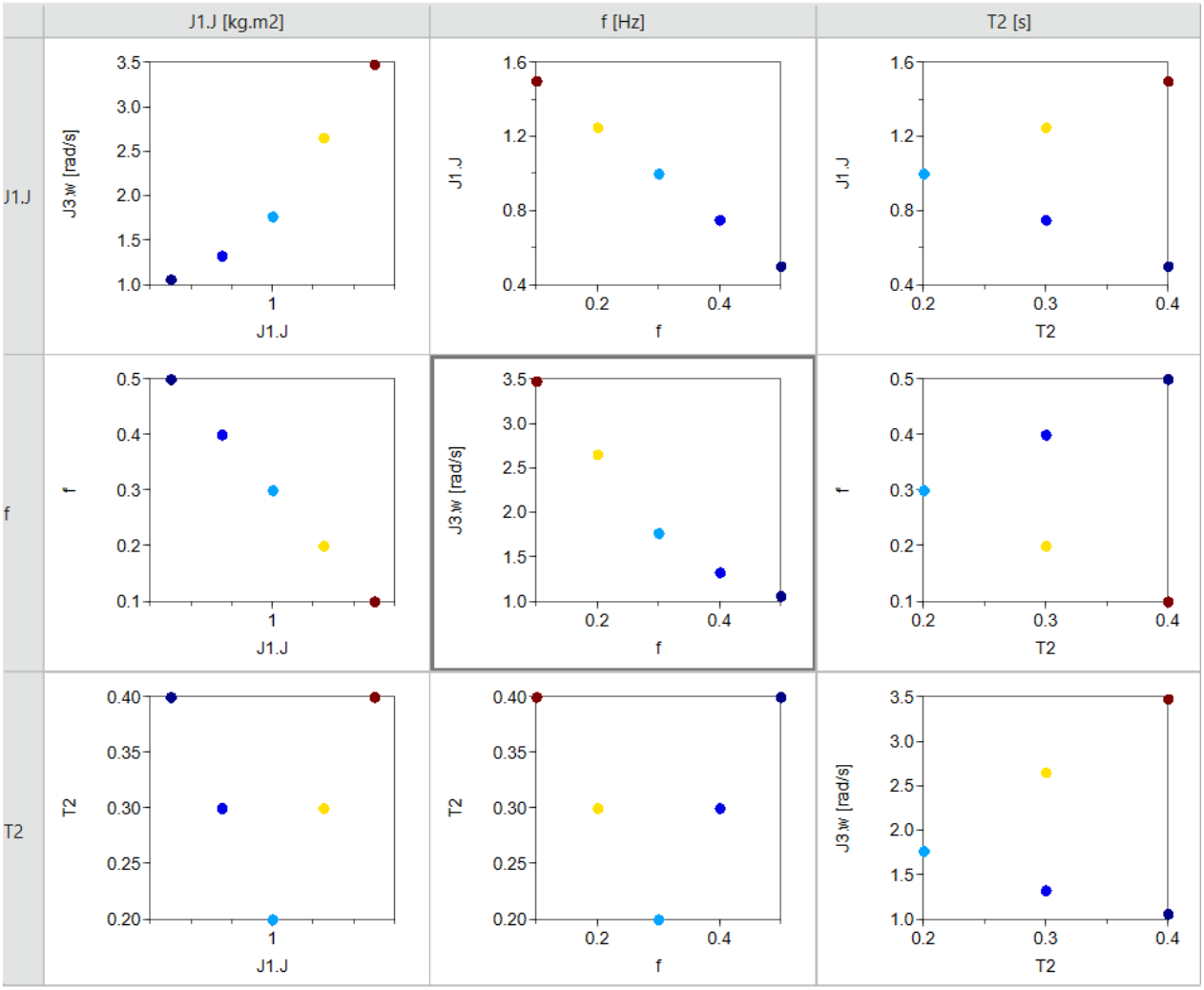
Last Point sweeps of more than two parameters

Also for **Last Point** sweeps **By Groups** of more than two parameters the result over group number plot is provided. Here group 1 consists of the values `{J1.J, f, T2} = {0.5, 0.5, 0.4}`.





For each result variable, a scatter plot is also created. These are the same scatter plots as for **All Combinations** sweeps. The difference here is that random perturbations are not used to perturb the parameter values in the scatter plots. The reason is that the plotted dots typically do not overlap for **By Groups** sweeps.



3.7 Model Calibration

3.7.1 Improved GUI for model calibration

Introduction

Similar to the Sweep Parameters and Optimization integrated GUIs, there is now an integrated GUI for model calibration.

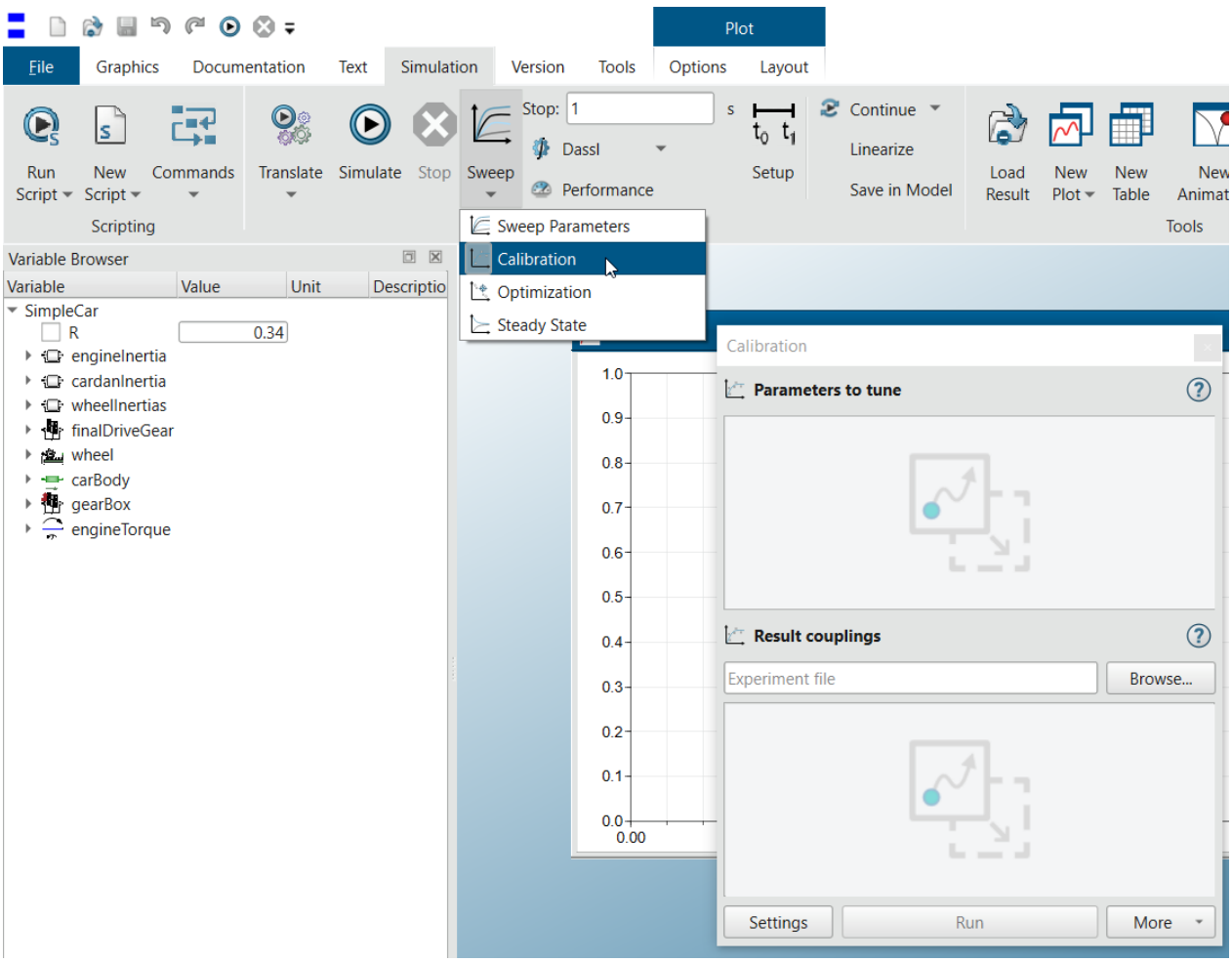
Model calibration (parameter estimation) is the process where measured data from a real device is used to tune parameters such that the simulation results are in good agreement with

the measured data. The parameters that we tune are often referred to as tuners. Dymola varies the tuners and simulates the model when it searches for satisfactory solutions.

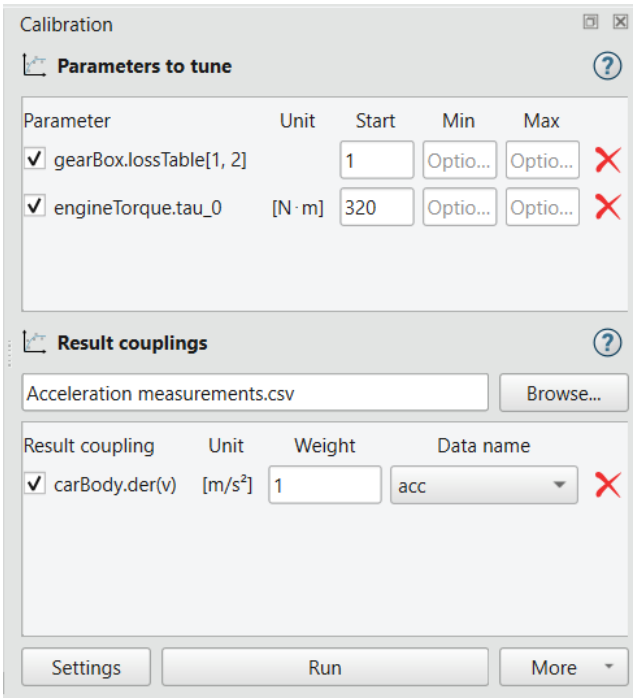
The new integrated GUI is built on the `Calibration` package of the Design library. The purpose of the integrated GUI is to simplify the process of setting up a calibration task and therefore only a subset of the `Calibration` package is supported. For more advanced setups (including e.g. several measurement files), please use the functions of `Design.Calibration` directly.

The Calibration GUI is similar to the Sweep Parameters and Optimization GUIs in that there are two panels, one for parameters and one for results. Variables can be dragged and dropped into these panels.

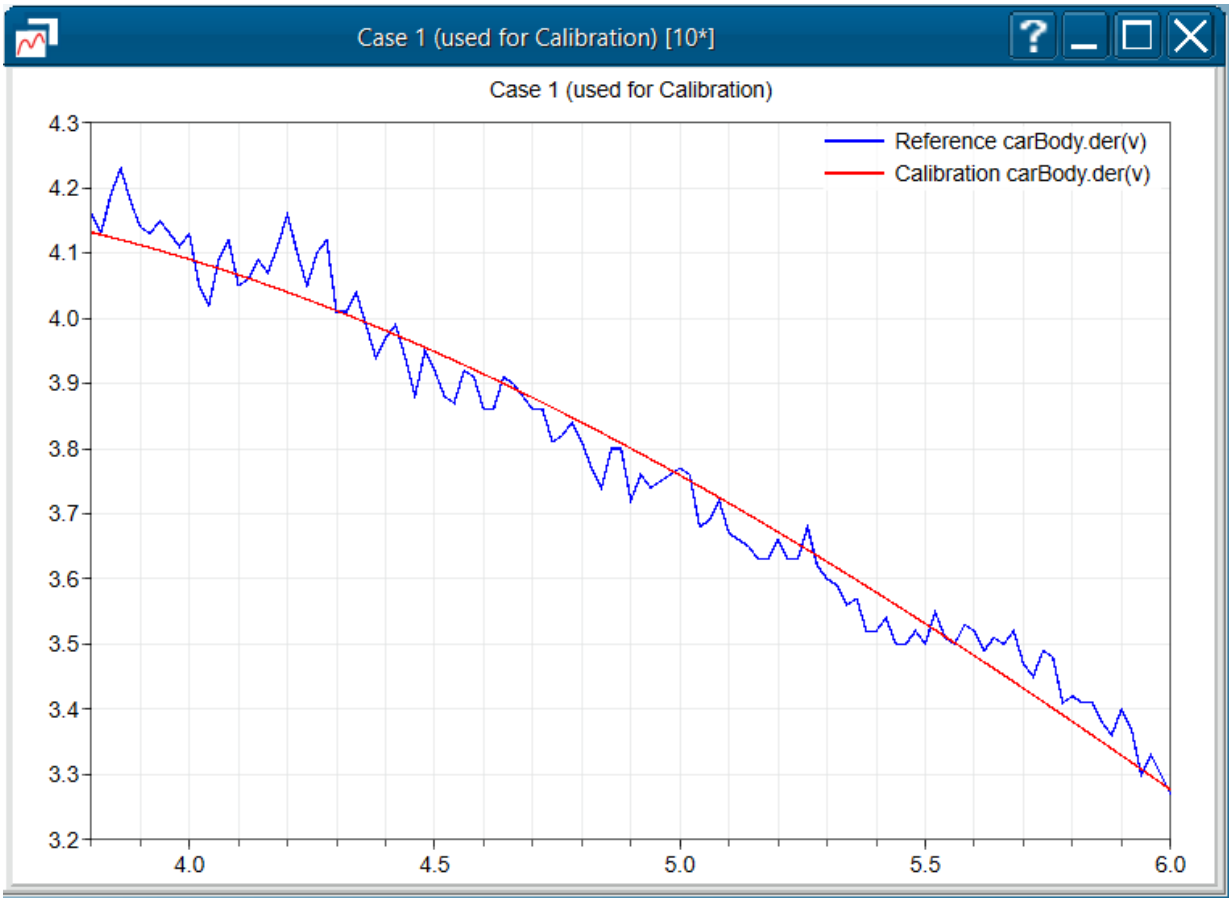
To display the **Calibration** GUI, use the command **Simulation > Sweep > Calibration**:



An example of a populated Calibration GUI:



When tuners and result couplings are defined, the calibration is started by clicking the **Run** button. The result of the calibration is automatically plotted:



Parameters to tune

The panel **Parameters to tune** accepts non-evaluated parameters and start values. These parameters are often referred to as tuners. Dymola varies the tuners and simulates the model when it searches for a satisfactory parametrization. For each such tuner a start guess must be provided. A default value is taken from the Variable Browser. The algorithm begins with the given start values and performs calibration while respecting the optional minimum and maximum constraints on the tuners.

If a tuner is unchecked then that parameter is set with the given start value but the parameter won't be varied during the calibration.

Note that if you change a unit, the start value, minimum value, and maximum value will be automatically rescaled. However, this is for the option of looking at the values in other units. When you start the calibration, the unit is reset, and the values are rescaled again.

Result couplings

Mathematically, the tuning algorithm is an optimization procedure to minimize the error between the simulation results and the measurements along the result trajectory. The method used is a least-squares fit with regularization to ensure that it does not get stuck due to redundant tuners.

Result couplings are defined by selecting result variables and coupling them to measurement data. The measurements should be provided in an experiment file (`.mat`, `.csv`). Use the **Experiment file** input field or the **Browse** button to select file.

The **Result coupling** panel accepts any variable from the Variable Browser. For each result coupling variable, a time series must be selected in the corresponding **Data name** drop down menu. These time series are read from the previously selected experiment file.

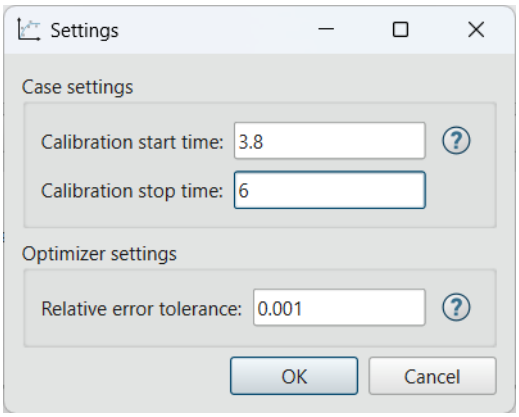
For each result coupling, a weight must be provided. The weight is multiplied with the error contribution of the variable.

If a result coupling is unchecked, then it won't be included in the calibration run, but its value will be plotted.

Note that if you change a unit, the weight will be automatically rescaled. However, this is for the option of looking at the value in other units. When you start the calibration, the unit is reset, and the value is rescaled again.

Settings

By clicking **Settings**, additional options can be set.

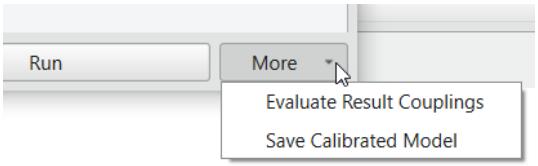


By using the input fields **Calibration start time** and **Calibration stop time** you can define a time interval in which the calibration shall be performed. Outside this interval, measurement data is not read and the error is not computed. Note that this time interval can be different from the simulation interval. For example, if the simulation start time is 0 s, then in the example shown in the figure, simulation is first done without calibration until 3.8 s, then measurements are read and the error is evaluated for each result coupling until the time 6 s.

The **Relative error tolerance** field defines what error tolerance should be used for the termination criteria.

Additional actions

Pressing the **More** button will display two additional actions.



Choose **Evaluate Result Couplings** to evaluate the error between result variables and their coupled measurements at the tuner start values.

After a calibration is done, you can select **Save Calibrated Model** to save the resulting tuner values in the current model or a new model extending from the current model. This option is based on Dymola's **Save in Model** feature.

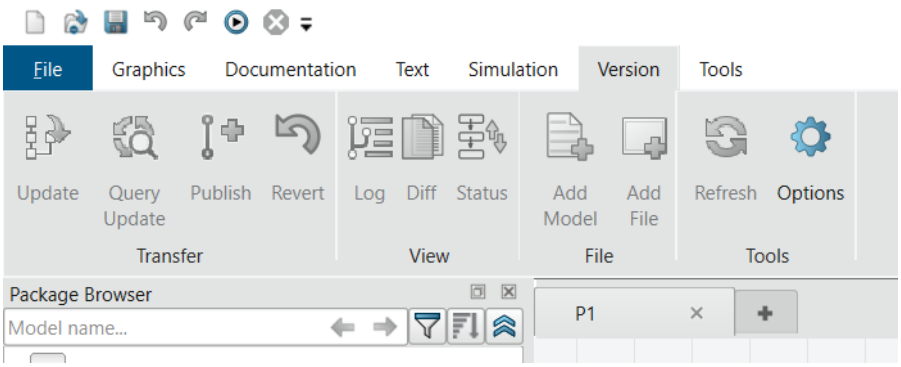
3.8 Model Management

3.8.1 Changed GUI for versioning – a new tab

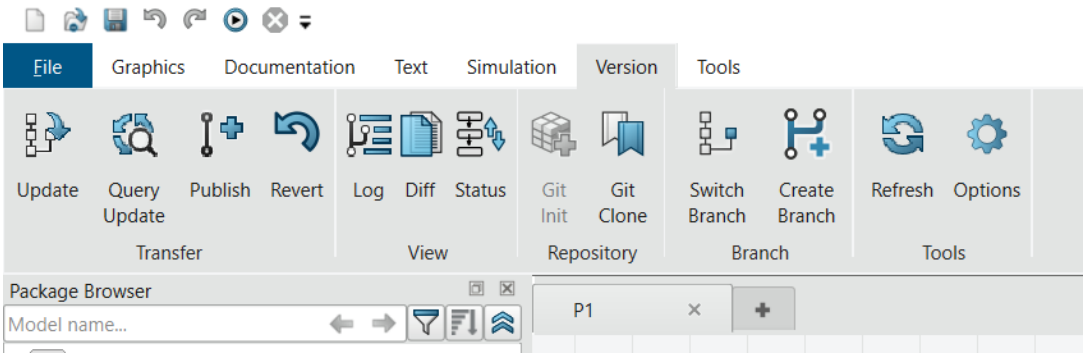
The versioning commands that were previously located, as a **Version** group under the **Tools** tab, have now been moved to a new separate tab **Versioning**.

The commands in this tab depends on the versioning system you have selected. Some examples:

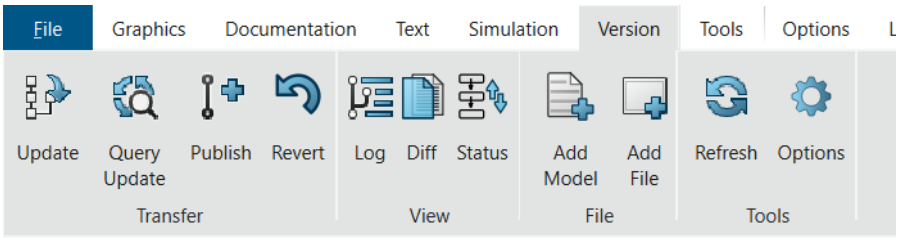
For the default of no versioning system selected, that tab looks like:



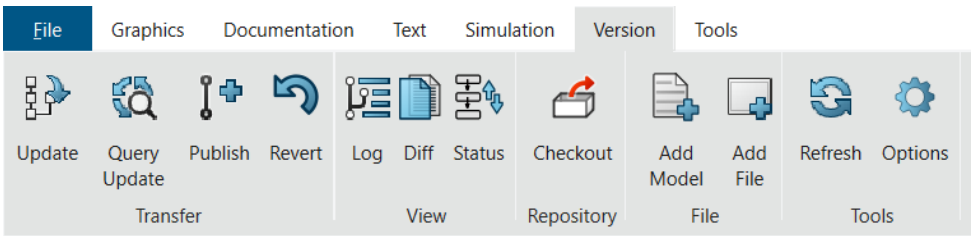
If Git is selected as versioning system, and a model using that versioning is opened, the tab looks like:



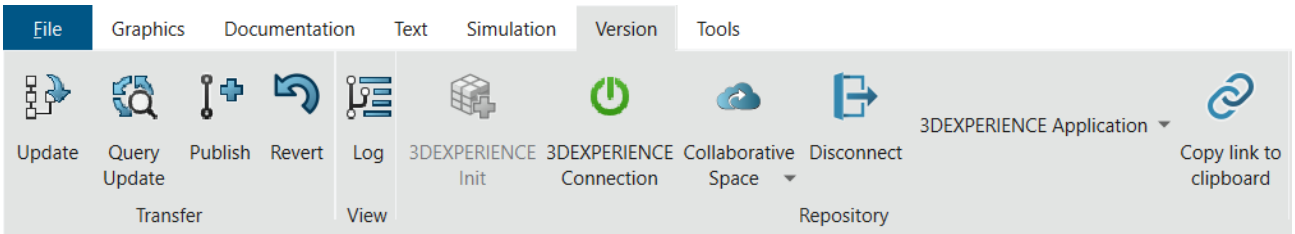
For CVS the tab looks like:

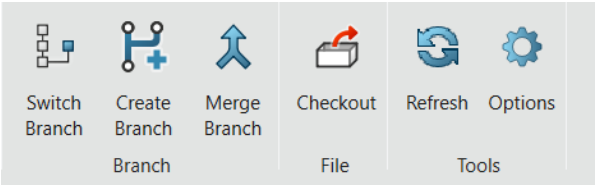


For SVN the tab looks like:



For 3DEXPERIENCE the tab looks like (it is split into two images here since being long):





Note that there are no new commands, and no renaming of any command. However, a number of commands now have icons.

3.8.2 Code signing on Windows of simulation binaries in Dymola

Dymola 2026x Refresh 1 supports code signing on Windows of simulation binaries, for example FMUs or SSP files. Note. Regarding FMUs, only DLLs built during the Dymola export will be signed, not any existing ones from external resources.

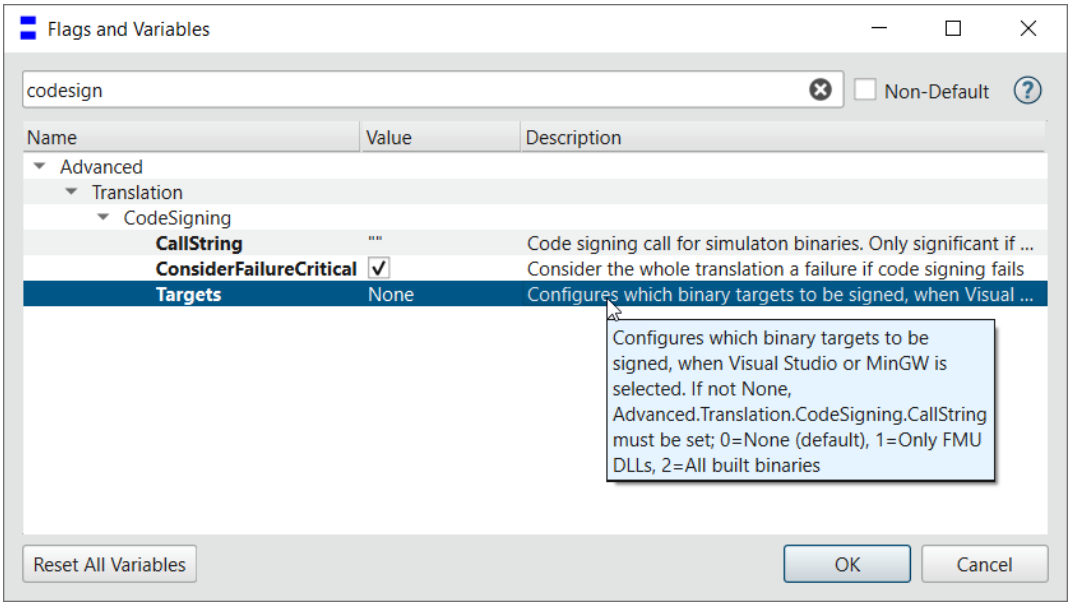
Apart from the security advantages of using signed binaries, some companies block files with unsigned certificates, and the virus protection software may also indicate problems with such files.

It is assumed that you have an external code-signing tool that can be run standalone, what is provided in Dymola is:

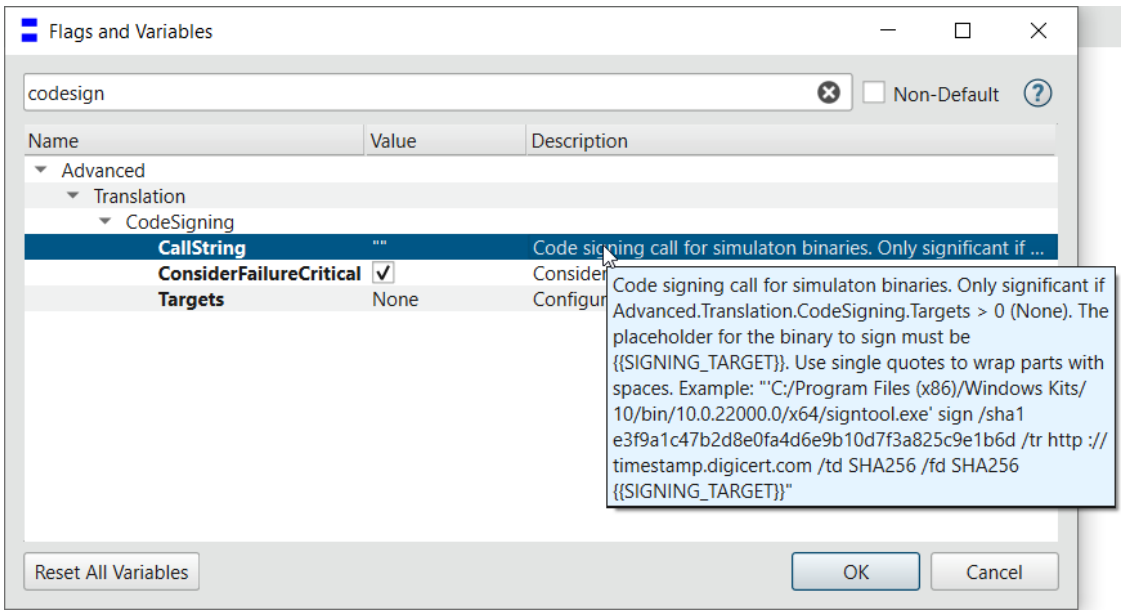
- A user-defined command string to run the tool, with a wildcard to plug in the file name.
- Code signing for generated code if the feature is activated.

Examples of standalone tools of the type that is suited for the task are SignTool (Windows SDK) and JSign (Java).

Three new flags are used in Dymola. The flag `Advanced.Translation.CodeSigning.Targets` specify what types of binaries that should be signed, as described in the flag description in the tooltip in the image below:



The flag `Advanced.Translation.CodeSigning.CallString` is the signing call for the simulation binaries, as described in the flag description in the tooltip in the image below:



Finally, by default, if the code signing fails, the complete translation is seen as failed, even if the translation of the model as such is working. To investigate if the failure is due to unsuccessful signing, or some other failure in model translation, you can set the flag:

```
Advanced.Translation.CodeSigning.ConsiderFailureCritical =
false
```

After setting the flag, translate again. If only the signing fails, you get a successful translation, but a warning in the translation log that the signing failed.

(The flag is by default `true`. The flag value is saved between sessions.)

3.8.3 Improvements in the 3DEXPERIENCE app "Design with Dymola"

Working with simulation data in Design with Dymola and 3DEXPERIENCE

General

In the **3D**EXPERIENCE Platform in general, you can work with a *physics simulation*, consisting of

- The model
- The scenario, that is, the simulation specification
- The simulation result

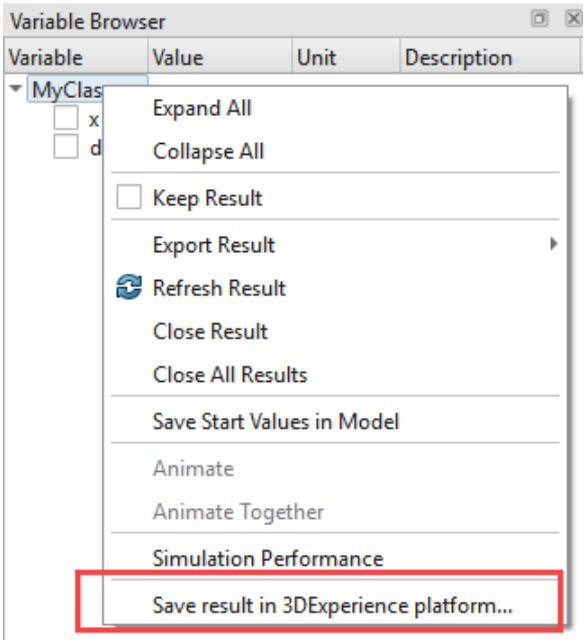
A physics simulation uses the **3D**EXPERIENCE Platform MSR (Model, Scenario, Result) format.

From the **3D**EXPERIENCE version R2026x FD03 (available in July 2026) you can work with the model and the simulation result parts of the MSR concept using Design with Dymola (with a Dymola version 2026x Refresh 1 or later).

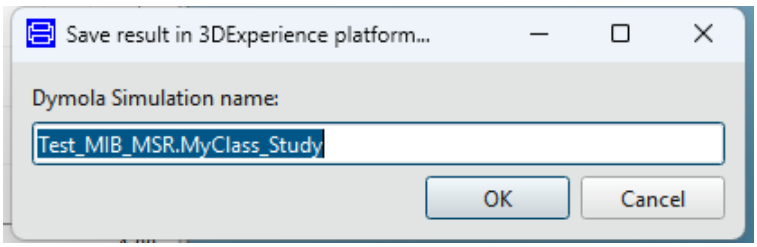
In practice, this means that you can save and retrieve a simulation result in the **3D**EXPERIENCE Platform from Design with Dymola. The simulation result is associated to the corresponding model when saving it, and the simulation result can later be searched for and opened, together with the associated model.

Saving a Design with Dymola simulation result in the 3DEXPERIENCE Platform

If you simulate a model in Design with Dymola, and this model is already stored in the 3DEXPERIENCE Platform, you can, after the simulation, right-click the top-level of the simulation result in the variable browser. You can then select **Save result in 3DEXPERIENCE Platform...**:



A new dialog is opened, proposing a name for the Design with Dymola simulation that will be saved in the 3DEXPERIENCE Platform:



By default, the name is the full class path of the simulation model associated to the result.

Note that the simulation object will always point to the latest iteration of the current branch. If a new iteration is published, opening the simulation model in Design with Dymola will open the last revision of the branch for that model.

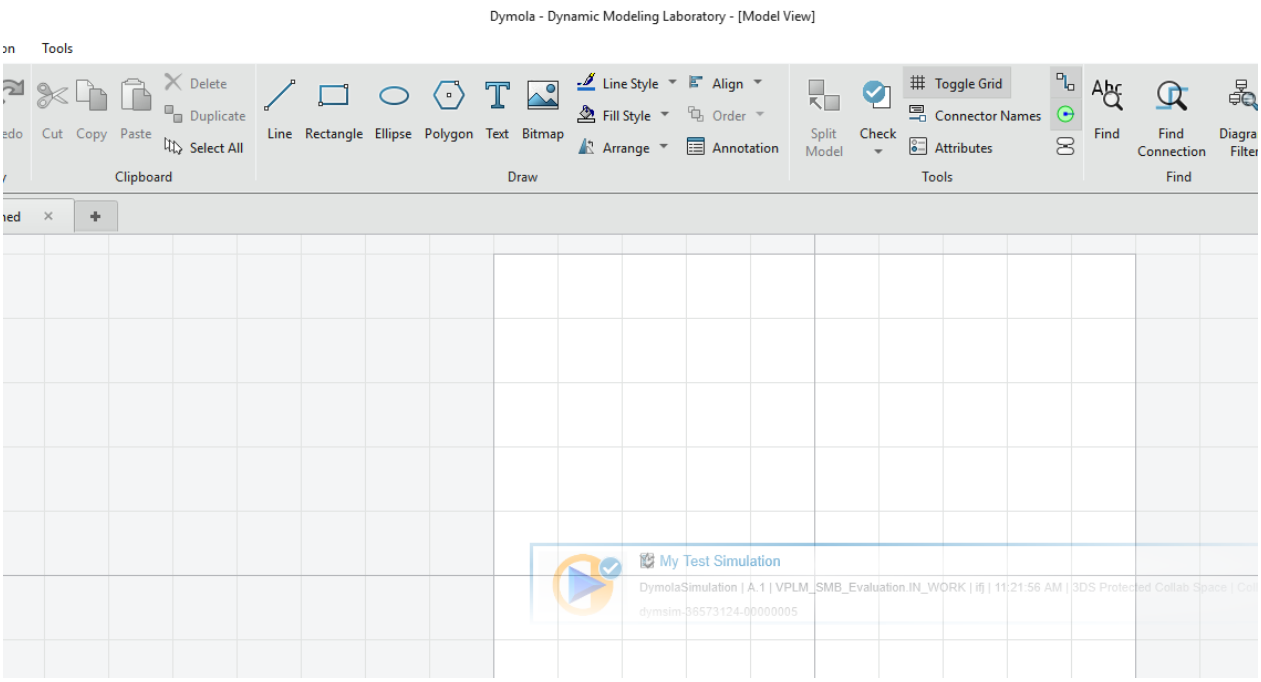
Note also that no versioning is currently available for MSR simulations, if you save the result again, a new MSR simulation object will be created. If the name already exists, a new object with the same name will be created.

Opening an MSR Design with Dymola simulation

Opening a Design with Dymola MSR model must be done in a Design with Dymola connected to the same 3DEXPERIENCE Platform where the model you want to open is stored.

To find a Design with Dymola model, start a search in the 3DEXPERIENCE Platform. If needed, use the associated shortcut prefix `dymsim:`. The result of the search will display Design with Dymola simulations, of object type `DymolaSimulation`. A search example:

To open this model, drag and drop the search result into the Design with Dymola main window:



This will open the corresponding Design with Dymola model pointed to by the MSR simulation object, and will also open the associated result in the Simulation tab of Design with Dymola.

If the model pointed to by the simulation object cannot be found, the simulation result will still be opened in Design with Dymola. However, it will not be associated to any model.

Note that you must search the simulation object from the 3DEXPERIENCE Platform, using the 3DSearch widget; you cannot search such object from within Design with Dymola.

Note also that you can only open the search result using drag and drop as described above.

3.9 Other Simulation Environments

3.9.1 Dymola – Matlab interface

Compatibility

The Dymola – Simulink interface now supports Matlab releases from R2021a (ver. 9.10) up to R2025b (ver. 25.2). On Windows, only Visual Studio C++ compilers are supported to generate the DymolaBlock S-function. On Linux, the gcc compiler is supported. The LCC compiler is not supported, neither on Windows nor on Linux.

Removal of options for models with many parameters

The dialog to select how to treat models with big parameter sets has been removed. The previously recommended option to store variables in a data file is now always used.

3.9.2 Real-time simulation

Compatibility – dSPACE

Dymola 2026x Refresh 1 officially supports the DS1006, MicroLabBox, and SCALEXIO systems for HIL applications. For these systems, Dymola 2026x Refresh 1 generated code has been verified for compatibility with the following combinations of dSPACE and Matlab releases:

- dSPACE Release 2021-A with Matlab R2021a
- dSPACE Release 2021-B with Matlab R2021b
- dSPACE Release 2022-A with Matlab R2022a
- dSPACE Release 2022-B with Matlab R2022b
- dSPACE Release 2023-A with Matlab R2023a
- dSPACE Release 2023-B with Matlab R2023b
- dSPACE Release 2024-A with Matlab R2024a
- dSPACE Release 2024-B with Matlab R2024b
- dSPACE Release 2025-A with Matlab R2024b and R2025a
- dSPACE Release 2025-B with Matlab R2024b, R2025a, and R2025b

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.

New utility functions – dym_rti_build2 and dym_rtimp_build2

Dymola 2021 introduced a new function, `dym_rti_build2`, which replaces `dym_rti_build` for building dSPACE applications from models containing DymolaBlocks.

The new function uses the new dSPACE RTI function `rti_build2` instead of the old function `rti_build`.

A corresponding new multi-processor build function, `dym_rtimp_build2`, is also introduced.

These functions are supported with dSPACE Release 2019-B and later.

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. Therefore, 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:

```
>> dym_rti_build('myModel', 'CM')
>> rti_build('myModel', 'Command', 'CM')
```

Note that this note applies the new functions `dym_rti_build2` and `rti_build2` as well.

Compatibility – Simulink Real-Time

Compatibility with Simulink Real-Time has been verified for all Matlab releases that are supported by the Dymola – Simulink interface, which means R2021a (Simulink Real-Time ver. 7.1) to R2025b (Simulink Real-Time ver. 25.2). Only Microsoft Visual C compilers have been tested.

3.9.3 Java, Python, and JavaScript Interface for Dymola

Java Interface: Name change of the Dymola Java interface file

The name of the Dymola Java interface file now also includes a version number. The name of this file in Dymola 2026x Refresh 1 is `dymola-interface-2026.1.jar`.

(This change was introduced already in Dymola 2026x, the name in this version is `dymola-interface-2026.0.jar`. In previous versions, the name was `dymola_interface.jar` (with underscore).)

Python Interface: Using Dymola Modelica Compiler (DMC) with Python

When instantiating the Python interface, you can now specify if Dymola standalone should be used or the new Dymola Modelica Compiler (DMC), a Dymola application without any user interface at all.

For more about DMC and how to use it with Python, see section "Dymola Modelica Compiler (DMC) tool" starting on page 29.

3.9.4 SSP Support in Dymola

Important; additional features!

Note that there are some FMI Beta features as well. Please see the section "Features under Development" starting on page 44.

Renaming of SSP flags

An increased number of flags related to SSP requires some structuring; the flags have been grouped and renamed accordingly.

The groups are:

- Export
- Import
- Other

The Export and Import groups contain flags that might be of interest to change in usual cases, the Others group contains flags that rarely need changing.

The complete list of changes below. New **bold** flag names indicates that there has been an additional change of the flag name, for example, change of words or splitting words, on top of adding a group name:

Important! Any flag usage from earlier versions or Dymola is not preserved; the flags with the old names have been removed, and new flags with improved names added. You must manually update the flag names if you, for example, has changed the default value of any such flag.

Old name	New name
Advanced.Beta.SSP. ExportExternalBinding	**Advanced.SSP. Export.ExternalBinding**
Advanced.Beta.SSP. MinimumVersion2	**Advanced.SSP. Export.MinimumVersion2**
Advanced.SSP. ApplyUnitConversion	Advanced.SSP. Other.ApplyUnitConversion
Advanced.SSP. CreateReplaceableComponents	**Advanced.SSP. Import.ReplaceableComponent**
Advanced.SSP. ExportCopyResources	Advanced.SSP. Export.CopyResources
Advanced.SSP. ExportProtected	Advanced.SSP. Other.ExportProtected
Advanced.SSP. ProvidePlacement	Advanced.SSP. Other.ProvidePlacement
Advanced.SSP. PurgeResources	Advanced.SSP. Import.PurgeResources
Advanced.SSP. RotationAttribute	Advanced.SSP. Other.ExportRotationAttribute
Advanced.SSP. StoreStartAsValue	**Advanced.SSP. Export.StartAsValue**

Advanced.SSP. SubpackageFMU	Advanced.SSP. Import.SubpackageFMU
Advanced.SSP. SupportValueParameter	Advanced.SSP. Other.SupportValueParameter
Advanced.SSP. TempUnpack	**Advanced.SSP. Import.TemporaryUnpack**
Advanced.SSP. TopLevelPackageSuffix	Advanced.SSP. Import.TopLevelPackageSuffix

SSP import

Including all variables of imported FMUs when importing an SSP file

Before you open an SSP file using the command **File > Open > Import SPP…**, you can select if all variables of imported FMUs should be included or not when importing the SSP file.

You do that by using the flag `Advanced.FMI.Import.IncludeAllVariables`. By default, this flag is `true`, which means you include all variables of contained FMUs when importing an SSP file. If you don´t want to do that, you can set the flag to `false`.

The value of the flag is saved between sessions.

Note that this flag (and corresponding setting) is used also when importing FMUs in general.

Keeping components without sources for later work and re-export

In SSP, it is possible to define a component without any source, for example, to define interfaces without implementation. Modelica requires that every component must have a class. To map SSP to Modelica, Dymola will automatically synthesize a transient model corresponding to such a component in the SSP. (This model is given a name that ends with `_model`.)

When you re-export an SSP, in most cases you do not want to save such transient models. However, if you plan to change the transient model you want to export that model too. In this case, *before importing the model in the first place*, set the flag:

```
Advanced.SSP.Import.TransientSynthesizedModel = false
```

This will specify an annotation that tells Dymola to keep the transient model when re-exporting it later.

(The flag is by default `true`.) The flag is saved between sessions.

Auto-rotation of connectors to better indicate if input or output connectors

When importing an SSP and displaying it, input and output connectors are auto-rotated if needed, in a “clever” way, based on where they are positioned.

The purpose is to make input connectors to face inwards even if located near the right edge, and the output connectors to face outwards even if placed near the left edge.

Prompting to save SSP package only if FMUs are imported

Dymola will only prompt you to save the package representing the SSP if FMUs are imported. This is because the FMU-import itself requires a file system location.

Handling component attributes that do not have any SSP representation

Component attributes that do not have any SSP representation are stored as vendor annotations. The attributes are `final`, `inner`, `outer`, and `replaceable`. Note that `inner` is needed to support SSP models using the **Testing** library.

SSP export

Specifying the SSP version to 2.0 when exporting

To always store the system description as version 2.0 when exporting an SSP, you can set the flag:

```
Advanced.SSP.Export.MinimumVersion2 = true
```

(The flag is by default false. The flag is saved between sessions.)

(This feature was a beta feature in the previous Dymola version. The flag has changed name in Dymola 2026x Refresh 1.)

Option to export SSP parameter bindings as external files

To export the SSP parameter bindings as external files (SSV files), if possible, you can use the flag `Advanced.SSP.Export.ExternalBinding`. The possible values of the flag are:

- **0 Inline par. binding** This is the default value.
- **1 System par. binding external**
- **2 All external**

(The flag value is saved between sessions.)

The file names are derived from the component or system name.

If the external files cannot be written for some reason, Dymola will instead store the parameter bindings inline.

(This feature was a beta feature in the previous Dymola version. The flag has changed name in in Dymola 2026x Refresh 1.)

Handling component attributes that do not have any SSP representation

Component attributes that do not have any SSP representation are stored as vendor annotations. The attributes are `final`, `inner`, `outer`, and `replaceable`. Note that `inner` is needed to support SSP models using the **Testing** library.

3.9.5 FMI Support in Dymola

Unless otherwise stated, features are available for FMI version 1.0, 2.0, and 3.0.

Important; additional features!

Note that there are some FMI Beta features as well. Please see the section "Features under Development" starting on page 44.

Renaming of FMI flags

An increased number of flags related to FMI requires some structuring; the flags have now been structured into two groups: `Advanced.FMI.Export.<name>` and `Advanced.FMI.Import.<name>`.

For flags that are only supported by FMI version 2 or FMI version 3, an additional segment is added to indicate that, that is, `Advanced.FMI.Export.FMI2.<name>`, or `Advanced.FMI.Export.FMI3.<name>`.

If there is no .FMI2 or .FMI3 segment in the flag, then the flag is supported both for FMI version 2 and for FMI version 3.

Important! Because FMI version 1 being deprecated (although still formally supported), this version is not taken into account for the new names, that is, you cannot from the name of an FMI flag know if it supported for FMI version 1 or not.

The complete list of changes can be seen by using the command **Tools > Help Documents**, see the "Miscellaneous" section (the last section).

Important! The new names are considered the only fully valid names. However, you can still use the old flag names, but they will give warnings when used, telling that these names are obsolete. An example of such a warning:

```
Warning: Script using obsolete flag Advanced.FMI.CopyDLLs. Flag
is now called Advanced.FMI.Export.CopyDLLs.
```

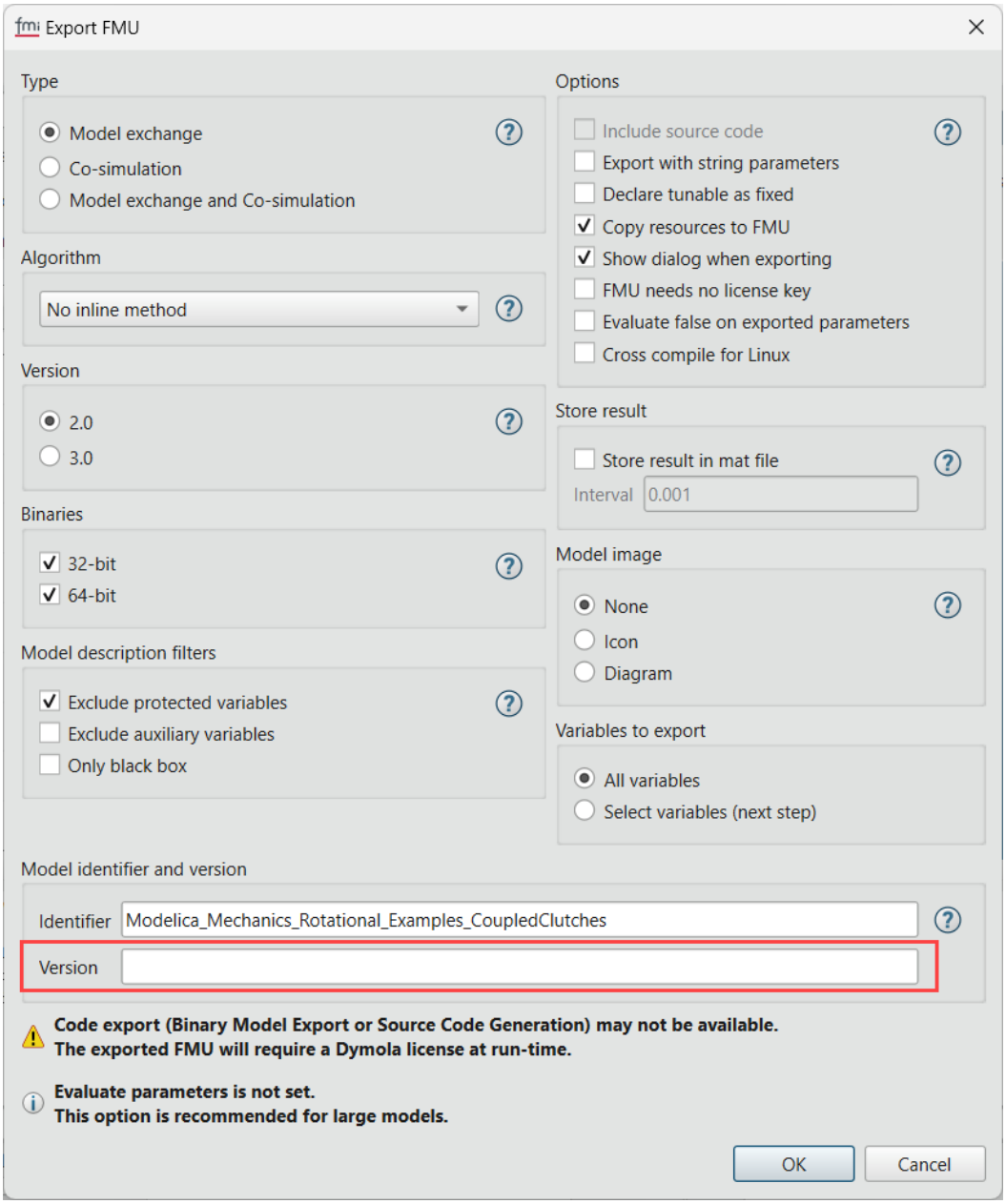
In addition, you can only see the new flag names in the **Flags and Variables** GUI.

To update the flag names in models and scripts, you can use the new built-in function `updateModelicaFlags`. For more information about this function, see section "New built-in function to update Advanced flag names in models and scripts, and simulation setup flag names saved in the model" on page 32.

FMI Export

Specifying the FMI model version when exporting an FMU

You can now specify the model version for an exported FMU, both in the GUI and using scripting. For GUI, the FMI export dialog has been extended:



The version (number) you specify sets the `fmiModelDescription.version` attribute in the file `modelDescription.xml`. If you leave the FMI model version blank, the attribute will be the version extracted from the corresponding Modelica package (if such a version exists).

For scripting, see section “The built-in function translateModelFMU enhanced” on page 32.

For the import of an FMU with model version information, see section "Handling an FMI model version when importing an FMU" on page 86.

Specifying the preferred communicationStepSize for exported Co-simulation FMUs

For certain cases, you can specify the preferred `communicationStepSize` for Co-simulation FMUs when exporting them. This step size is defined as the `stepSize` attribute of the `DefaultExperiment` element in the XML of the FMU.

You can specify the `communicationStepSize` by setting the flag:

```
Advanced.FMI.Export.DefaultExperimentStepSize.
```

The default value of the flag is 0.0. If the value of the flag is 0.0 or lower, the flag is ignored. The flag is saved between sessions.

However, for the following cases of exporting Co-simulation FMUs, you *cannot* use the flag; it is ignored:

- When using a fixed step solver like Euler or Rkfix*.
- When using an inline method with `Advanced.Translation.InlineMethod > 0` and `Advanced.Simulation.InlineFixedStep > 0`.

For these cases, the `stepSize` attribute is set according to the simulation settings, and not the flag.

(Note also that the flag has no meaning for Model Exchange FMUs.)

Documentation of previously undocumented Advanced flags for FMI Export

A number of Advanced flags for FMI Export have been undocumented in the manuals since long. The general name change of Advanced FMI flags activated the need for this documentation. The table below lists documentation for Advanced flags for FMI Export. Note that there is a similar section below for undocumented FMI Import flags.

In many cases, the documentation is actually only the description string of the flag. Flags where this documentation adds more information are market by ***bold italics***.

Flag (Advanced.FMI.Export. in name excluded)	**Default value/saved between sessions**	**Documentation**
AllowMultipleInstances	true/no	Allow multiple instances when instantiating the FMU. This corresponds to the attribute canBeInstantiatedOnlyOncePerProcess in the FMI 2- and FMI 3 standard. Note that when exporting with csSolver then this attribute is always true since our csSolver does not support multiple instances.

AllowPhysicalConnectors	false/no	When exporting an FMU with physical connectors, Dymola will export the variables inside the connectors as local variables. If this flag is set to true, Dymola will try to determine the causality of the variables and export them with input/output causality.
AllowStringParametersForFMU	false/yes	If this flag is set to true, if there are String parameters in the model, when exporting the model as an FMU, the String parameters in the model are exposed, and allowed to be changed through the FMU interface.
CompileFMU32	true/no	Generate 32-bit binaries in FMU. Implicitly false for MinGW and WSL compiler.
CompileFMU64	true/no	Generate 64-bit binaries in FMU, if 64-bit compilation is enabled by the flag Advanced.Translation.CompileWith64.
ConcealFilteredStates	true/no	Conceals variables that must be included in the modelDescription but has been selected to be excluded.
cvodeTolerance	1e-05/no	Specifies the default tolerance when exporting a co-simulation FMU with Cvode as solver.
InlineTypeDefinitions	false/no	By default (false), the types are declared in TypeDefinitions and the variables refer to these types. If set to true, then the type is inline with the variable definition.
SelectVariables	false/no	Corresponds to the **Select Variables** option in FMU export GUI.
ShowExportDialog	true/yes	By default, display the **Export FMU** dialog when exporting an FMU.

FMI Import

FMI 2: Option to preserve the hierarchical structure of IO

When importing an FMU, you can preserve the hierarchical structure of the IO by setting the flag:

```
Advanced.FMI.Import.FMI2.StructuredIO = true
```

(The flag is by default `false`.)

Notes:

- The flag `Advanced.FMI.Import.StructuralDeclaration` must be `true` for the above to work.
- The flag is only working for FMUs of FMI version 2.

As example of the effect, consider a hierarchical connector

```
A.b
A.c
```

If the flag `Advanced.FMI.Import.FMI2.StructuredIO` is `false`, you will have two inputs when you import the FMU:

```
Modelica.Blocks.Interfaces.RealInput A_b
Modelica.Blocks.Interfaces.RealInput A_c
```

If you set the flag to `true`, and reimport the FMU, you instead get:

```
connector A_con
  input b
  input c
end Acon;
A_con a;
```

(This feature was a beta feature in the previous Dymola version. The flag has changed name in Dymola 2026x Refresh 1.)

FMI 2 and FMI 3: Option to use a standardized package of FMI functions when importing FMUs

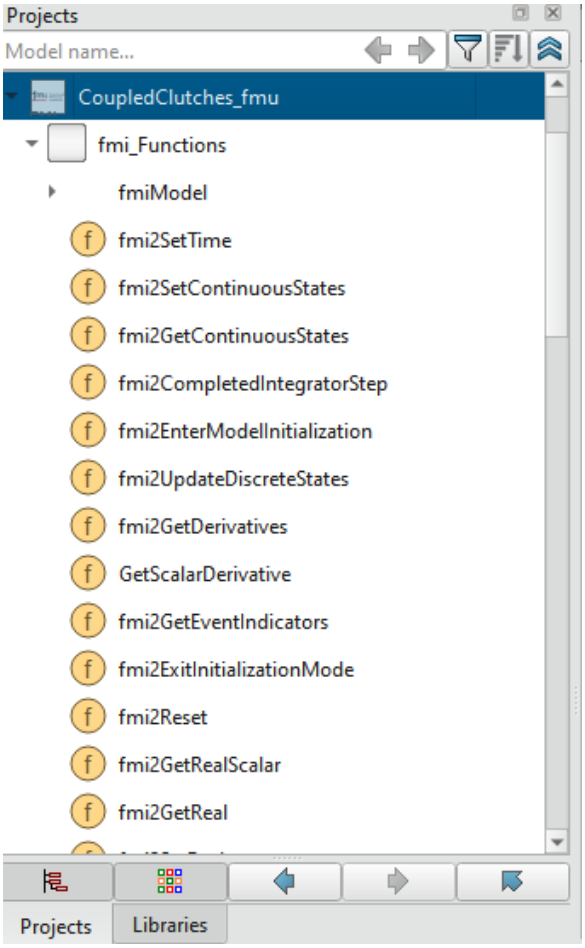
By default, each imported FMU produces fresh C-code. If you set the flag

```
Advanced.FMI.Import.ExternalLibrary = true
```

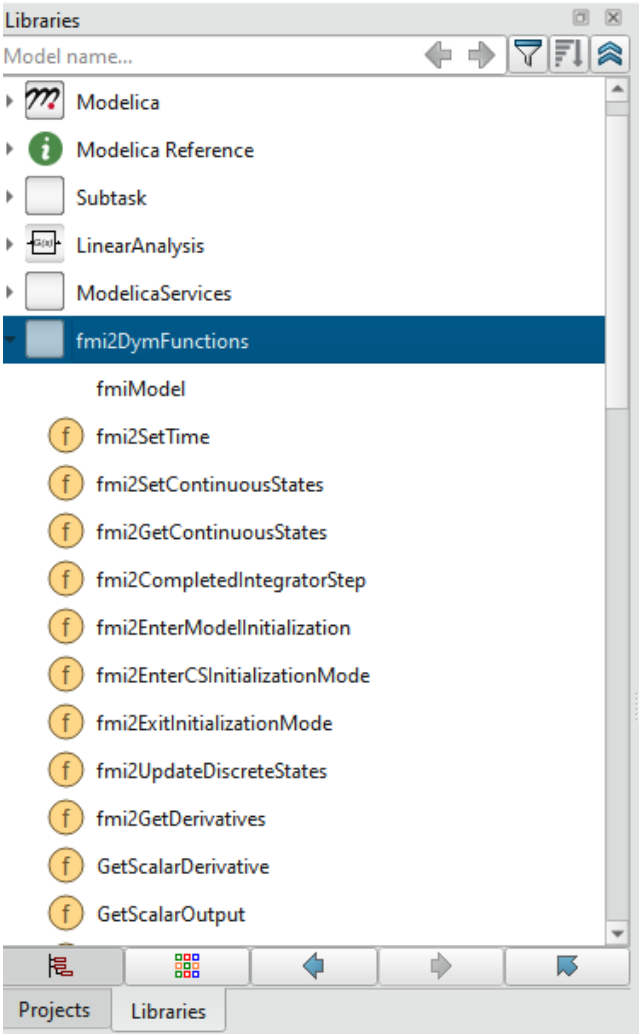
a standardized import package with all FMI2 and FMI3 functions that are common between FMUs is loaded and unitized in the Modelica wrapper for the FMU for both co-simulation and model exchange.

(The flag is by default false.) The flag is not saved between sessions.

For an example, start by keeping the default value `false` for the above flag and export and import the Coupled Clutches demo as a model exchange FMU using FMI 3. In this case, all FMI functions are present inside the Modelica wrapper for the FMU. To see this, you must first activate the setting **Show protected classes** using the command **Tools > Options**, in the **Package Browser tab**. Now you can look at the imported FMU in the package browser:



Now set the flag `Advanced.FMI.Import.ExternalLibrary = true` and re-import the FMU. The FMI functions are now included in the external library under the **Libraries** tab of the FMU.



All imported FMI 2 FMUs will share the C-functions from the **fmi2DymFunctions** library. For FMI 3, the external library is called **fmi3DymFunctions**.

Important! FMU source code import is currently not supported using such external libraries.

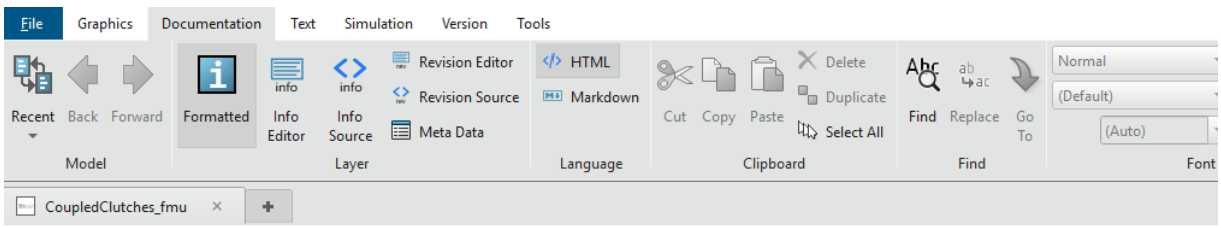
(This feature was a beta feature in the previous Dymola version. The flag has changed name in Dymola 2026x Refresh 1.)

Handling an FMI model version when importing an FMU

When importing an FMU that contains a `version` attribute under `fmiModelDescription`, the version information is added to the documentation view under “ModelDescription Attributes”. An example:

```
<fmiModelDescription
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  fmiVersion="2.0"
  modelName="CoupledClutches"
  guid="{f30c525a-f796-465e-b028-cc5c48391cba}"
  description="Drive train with 3 dynamically coupled clutches"
  version="1.0.2 Alpha"
```

Importing this FMU into Dymola, you can find this information in the documentation view (as “version = 1.0.2 Alpha”):



Drive train with 3 dynamically coupled clutches

Information

ModelDescription Attributes

- fmiVersion = 2.0
- modelName = CoupledClutches
- version = 1.0.2 Alpha
- generationTool = Dymola Version 2026x Refresh 1 Beta 1, 2026-02-27 (Co-simulation using Cvode)
- generationDateAndTime = 2026-02-25T10:25:47Z

Documentation of previously undocumented Advanced flags for FMI Import

A number of Advanced flags for FMI Import have been undocumented in the manuals since long. The general name change of Advanced FMI flags activated the need for this documentation. The table below lists documentation for Advanced flags for FMI Import. Note that there is a similar section below for undocumented FMI Export flags.

In many cases, the documentation is actually only the description string of the flag. Flags where this documentation adds more information are market by ***bold italics***.

Flag (Advanced.FMI.Import. in name excluded)	Default value/saved between sessions	Documentation
LockFMUInstance	false/no	When importing FMUs, set this flag to true to lock each FMU instance as an external object, which will allow multiple parallel FMU-objects of the same FMU. As a general note on parallelizing FMUs: For best performance when parallelizing the co-simulation in Dymola, the following best practices are currently recommended: -- When possible, use the default option of `fmi_UsePreOnInputSignals = true` to allow for evaluation of FMUs in parallel in most cases. -- The calculation time for each step of the FMU must be significant (or at least for some of the FMUs). -- The communication interval should be the same for all significant FMUs. -- When several instances of the same FMU is used, set Advanced.FMI.Import.LockFMUInstance = true.
ModelicaVersionInImport	“”/no	Adds a “uses(Modelica(version=<flag value>))” to the imported FMU wrapper.
FMI2.RecreateArrays	true/yes/	When importing an FMU, Dymola will, when possible, group elements of an array in the model description into an array in the Modelica wrapper. This flag assumes that the Structural Declaration is activated for the import.
ShowImportDialog	true/yes	By default, display the **Import FMU** dialog when importing an FMU.

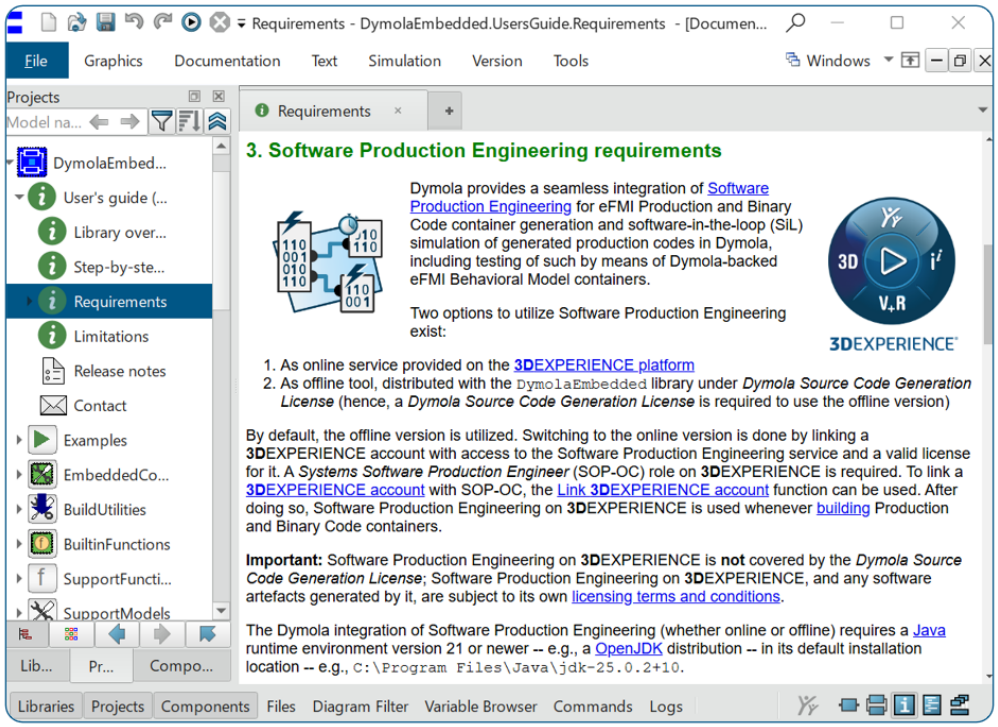
3.9.6 eFMI Support in Dymola

A number of improvements have been implemented:

Major improvements

Integrated production code generation

Dymola now ships with an offline version of Software Production Engineering available under *Dymola Source Code Generation License*. Software Production Engineering on **3D**EXPERIENCE (SOP-OC) is still supported and used if a **3D**EXPERIENCE account is linked; otherwise the new offline version is used. For details, see the requirements documentation (`DymolaEmbedded.UsersGuide.Requirements`).



Offline production code generation under Dymola Source Code Generation License.

Improved event handling of GALEC code generator

The GALEC code generator now supports (1) `pre` operators, including the required event iteration and initialization, (2) `whenelse` branches for reinitialization with `reinit`, and (3) `if` statements, including optional `elseif` and `else` branches, in algorithm sections (not functions).

Export of production code as FMI 3.0 source code co-simulation FMU

The actual production code can now be exported as source code FMUs (see `DymolaEmbedded.EmbeddedConfiguration.ProductionCode.build_FMU`). The exported source code FMUs are self-contained (no external library dependencies), small, high-quality (MISRA C:2025, SEI CERT C, etc.), safety-critical code suited for hard real-time. They are suited for many real-time platforms with source code FMU import, like ConfigurationDesk (dSPACE), TwinCAT (Beckhoff), Silver (Synopsys), etc.

Minor improvements:

- The previous FMU export provided by `EmbeddedConfiguration.BinaryCode.build_FMU()` is deprecated; it is replaced by the new FMU source code export, please see the previous section, the third item.
- eFMU SiL-stubs and experiment-packages now support Modelica enumerations. Such are mapped to GALEC block-interface variables of type `Integer`.
- Unlinking 3DEXPERIENCE credentials now creates a backup that can be used for relinking (For more information, please see the unlink (`DymolaEmbedded.UsersGuide.Requirements.unlink_3DEXPERIENCE_account`) and relink (`DymolaEmbedded.UsersGuide.Requirements.relink_3DEXPERIENCE_account`) functions).
- Software Production Engineering now requires a Java runtime environment version 21 or newer (before it has been Java 17 or newer).
- Added support for Microsoft Visual Studio 2026 when building Binary Code containers with Software Production Engineering.
- Updated check code scripts to work with latest Cppcheck version (open source 2.20.0, premium 26.3.0).

New library versions:

- DymolaEmbedded 1.0.8
- eFMI_TestCases_EmbeddedConfigurations 1.0.8

3.9.7 Model structure: Improved model editing API

Introduction

The functions to create and edit Modelica models using function calls, located in the package `ModelManagement.Structure.AST` have been slightly improved. Below the changed functions are listed, in alphabetical order in each subpackage.

New AST functions

For editing classes (in the **Classes** subpackage):

Name	Description
ConnectorsInClass	Returns a list of connectors of the given class. There is an option to get all connectors including those inherited from a base class.

Changed AST functions

Miscellaneous functions (in the **Misc** subpackage):

Name	Description of change
ClassExists	An optional argument to check if the class is writeable has been added.

3.10 Modelica Standard Library and Modelica Language Specification

The current version of the Modelica Standard Library is version 4.1.0. The current version of the Modelica Language Specification is 3.6.

3.11 Documentation

General

In the software, distribution of Dymola 2026x Refresh 1 Dymola User Manuals of version “March 2026” will be present; these manuals include all relevant features/improvements of Dymola 2026x Refresh 1 presented in the Release Notes, except the “under development” ones (if present).

3.12 Appendix – Installation: Hardware and Software Requirements

Below the current hardware and software requirements for Dymola 2026x Refresh 1 are listed.

3.12.1 Hardware requirements/recommendations

Hardware requirements

- At least 2 GB RAM
- At least 1 GB 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.

3.12.2 Software requirements

Microsoft Windows

Dymola versions on Windows and Windows operating systems versions

Dymola 2026x Refresh 1 is supported, as 64-bit application, on Windows 11. 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, or a GCC compiler, must be installed separately. The following compilers are supported for Dymola 2026x Refresh 1 on Windows:

Microsoft C/C++ compilers, free editions:

Note. When installing any Visual Studio, make sure that the option "C++/CLI support…" is also selected to be installed. (Also, note that Bundled Visual Studio toolsets are not supported. The workaround is to install the older build tools separately instead of bundled in e.g. a Visual Studio 2022 installation.)

- 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 2017 Community 2017 (15)
- Visual Studio 2017 Build Tools **Notes:**
    - The recommended selection to run Dymola is the workload "Visual C++ build tools" + the option "C++/CLI Support…"
    - Installing this selection, no IDE (Integrated Development Environment) is installed, only command line features
    - This installation is not visible as a specific selection when later selecting the compiler in Dymola, the alternative to select is the same as for any Visual Studio 2017 alternative: **Visual Studio 2017/Visual C++ 2017 Express Edition (15)**.
    - For more information about installing and testing this compiler with Dymola, see the document Installing Visual Studio Build Tools for Dymola.pdf.
- Visual Studio 2019 Community (16). Note that you can select to use Clang as code generator for this compiler.
- Visual Studio 2019 Build Tools **Notes:**
    - The recommended selection to run Dymola is the workload "C++ build tools" + the option "C++/CLI Support…"
    - Installing this selection, no IDE (Integrated Development Environment) is installed, only command line features
    - This installation is not visible as a specific selection when later selecting the compiler in Dymola, the alternative to select is the same as for any Visual Studio 2019 alternative: **Visual Studio 2019/Visual C++ 2019 (16)**.
    - For more information about installing and testing this compiler with Dymola, see the document Installing Visual Studio Build Tools for Dymola.pdf.
    - You can select to use Clang as code generator for this compiler.
- Visual Studio 2022 Community (17). Note that you can select to use Clang as code generator for this compiler.
- Visual Studio 2022 Build Tools **Notes:**
    - The recommend selection to run Dymola is the workload "Desktop development with C++" + the option "C++/CLI Support…"

    - Installing this selection, no IDE (Integrated Development Environment) is installed, only command line features
    - This installation is not visible as a specific selection when later selecting the compiler in Dymola, the alternative to select is the same as for any Visual Studio 2022 alternative: **Visual Studio 2022/Visual C++ 2022 (17)**.
    - For more information about installing and testing this compiler with Dymola, see the document Installing Visual Studio Build Tools for Dymola.pdf.
    - You can select to use Clang as code generator for this compiler.
- Visual Studio 2026 Community (18). Note that you can select to use Clang as code generator for this compiler.
- Visual Studio 2026 Build Tools **Notes:**
    - The recommend selection to run Dymola is the workload "Desktop development with C++" + the option "C++/CLI Support…"
    - Installing this selection, no IDE (Integrated Development Environment) is installed, only command line features
    - This installation is not visible as a specific selection when later selecting the compiler in Dymola, the alternative to select is the same as for any Visual Studio 2026 alternative: **Visual Studio 2026/Visual C++ 2026 (18)**.
    - For more information about installing and testing this compiler with Dymola, see the document Installing Visual Studio Build Tools for Dymola.pdf.
    - You can select to use Clang as code generator for this compiler.

Microsoft C/C++ compilers, professional editions:

Note. When installing any Visual Studio, make sure that the option "C++/CLI support…" is also selected to be installed. (Also, note that Bundled Visual Studio toolsets are not supported. The workaround is to install the older build tools separately instead of bundled in e.g. a Visual Studio 2022 installation.)

- Visual Studio 2015 (14.0)
- Visual Studio Professional 2017 (15)
- Visual Studio Enterprise 2017 (15)
- Visual Studio Professional 2019 (16). Note that you can select to use Clang as code generator for this compiler.
- Visual Studio Enterprise 2019 (16). Note that you can select to use Clang as code generator for this compiler.
- Visual Studio Enterprise 2022 (17). Note that you can select to use Clang as code generator for this compiler.
- Visual Studio Professional 2022 (17) Note that you can select to use Clang as code generator for this compiler.

- Visual Studio Enterprise 2026 (18). Note that you can select to use Clang as code generator for this compiler.
- Visual Studio Professional 2026 (18) Note that you can select to use Clang as code generator for this compiler.

Clang compiler

If you first select to use Visual Studio 2019, Visual Studio 2022 or Visual Studio 2026 as compiler, you can then select to use Clang as code generator instead of native Visual Studio.

Intel compilers

Note!

Important. The support for Intel compilers are discontinued from the previous Dymola 2022 version.

MinGW GCC compiler

Dymola 2026x Refresh 1 has limited support for the MinGW GCC 64-bit compiler. The versions 7.3, 8.1, and 15.1.0 have been tested. Normally, later versions will also be compatible.

To download any of these vesrions, please see the manual "*Dymola User Manual 1: Introduction, Getting Starting, and Installation*", the chapter "Appendix – Installation" 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.

Also, note that to be able to use other solvers than Lsodar, Dassl, and Euler, you must also add support for C++ when installing the MinGW GCC compiler. Usually, you can select this as an add-on when installing GCC MinGW.

Current limitations with 64-bit Min GW GCC:

- Embedded server (DDE) is not supported.
- Support for external library resources is implemented, but requires that the resources support GCC, which is not always the case.
- FMUs must be exported with the code export option[1] enabled.
- Compilation may run out of memory also for models that compile with Visual Studio.

(Note that the support for the 32-bit GCC compiler is removed from this version, Dymola 2026x Refresh 1.)

WSL GCC compiler (Linux cross-compiler)

Dymola on window supports cross-compilation for Linux via the use of Windows Subsystem for Linux (WSL) GCC compiler. The default WSL setup is 64-bit only and Dymola adopts this limitation. Notes:

- WSL is usually not enabled on Windows, so you need to enable WSL on your computer and install needed software components.

[1] Having the code export options means having any of the license features **Dymola Binary Model Export** or the **Dymola Source Code Generation**.

- You must download and install a suitable Linux distribution, including a C compiler. We recommend Ubuntu 20 since it is the most tested version for Dymola. In particular, the integration algorithms RadauIIa, Esdirk23a, Esdirk34a, Esdirk45a, and Sdirk34hw have been confirmed to work with Ubuntu 20, but not with Ubuntu 18.
- The WSL Linux environment can compile the generated model C code from Dymola in order to produce a Linux executable dymosim or a Linux FMU. (To generate Linux FMUs, you must use a specific flag as well.)
- Note that you can select to use Clang as code generator for this compiler.

Dymola license server

For a Dymola license server on Windows, all files needed to set up and run a Dymola license server on Windows using FLEXnet, except the license file, are available in the Dymola distribution. (This includes also the license daemon, where Dymola presently supports FLEXnet Publisher version 11.19.6. This version is part of the Dymola distribution.)

As an alternative to FLEXnet, Dassault Systèmes License Server (DSLS) can be used. Dymola 2026x Refresh 1 supports DSLS R2026x. Earlier DSLS versions cannot be used.

Note that running the Dymola license server on virtual machines is not a supported configuration, although it might work on some platforms.

Linux

Supported Linux versions and compilers

Dymola 2026x Refresh 1 runs on Red Hat Enterprise Linux (RHEL) version 8.6, 64-bit, with gcc version 11.2.1, and compatible systems. (For more information about supported platforms, do the following:

- Go to https://doc.qt.io/
- Select the relevant version of Qt, for Dymola 2026x Refresh 1 it is Qt 6.10.2.
- Select Supported platforms)

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 be able to select Clang, it must be installed, e.g. on Red Hat Enterprise Linux (RHEL): `sudo yum install clang` – on Ubuntu: `sudo apt install clang`.)

You can use a dialog to select compiler, set compiler and linker flags, and test the compiler by the **Verify Compiler** button, like in Windows. This is done by the command **Simulation > Setup**, in the **Compiler** tab.

Dymola 2026x Refresh 1 is supported as a 64-bit application on Linux. Corresponding support for 64-bit export and import of FMUs is included.

Notes:

- Dymola is built with Qt 6.10.2 and inherits the system requirements from Qt. However, since Qt 6.10.2 no longer supports embedding of the XCB libraries, these must now be present on the platform running Dymola. To know what to download and install, see the table in https://doc.qt.io/qt-6/linux-requirements.html for the list of versions of the ones

starting with “libxcb”. Note that the development packages (“-dev”) mentioned outside the table are not needed.

- For FMU export/import to work, zip/unzip must be installed.
- Support for 32-bit simulation on Linux (including 32-bit export and import of FMUs) is discontinued from Dymola 2025x.

Note on libraries

- The following libraries are currently not supported on Linux:
    - Process Modeling Library
    - Thermodynamics Connector Library
    - UserInteraction Library

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 11.19.6.)

As an alternative to FLEXnet, Dassault Systèmes License Server (DSLS) can be used. Dymola 2026x Refresh 1 supports DSLS R2026x. Earlier DSLS versions cannot be used.

Note that running the Dymola license server on virtual machines is not a supported configuration, although it might work on some platforms.

' "