

Dymola

Dynamic Modeling Laboratory

Dymola Release Notes

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

Document version: 1

© Copyright 1992-2023 by Dassault Systèmes AB. All rights reserved.
Dymola® is a registered trademark of Dassault Systèmes AB.
Modelica® is a registered trademark of the Modelica Association.

Other product or brand names are trademarks or registered trademarks of their respective holders.

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	5
2	About this booklet	6
3	Dymola 2024x	7
3.1	Introduction	7
3.1.1	Additions and improvements in Dymola	7
3.1.2	New and updated libraries	8
3.2	Developing a model	9
3.2.1	Minor improvements	9
3.3	Simulating a model	14
3.3.1	Sparse solver for system of equations	14
3.3.2	Plot tab	16
3.3.3	Scripting	16
3.3.4	Minor improvements	17
3.4	Installation	22
3.4.1	Minor improvements	22
3.4.2	Installation on Windows	22
3.4.3	Installation on Linux	24
3.5	Features Under Development	24
3.6	Model Management	29
3.6.1	Support for working with the 3DEXPERIENCE database using the “Design with Dymola” app	29
3.6.2	Minor improvements	43
3.7	Other Simulation Environments	44
3.7.1	Dymola – Matlab interface	44
3.7.2	Real-time simulation	45
3.7.3	Java, Python, and JavaScript Interface for Dymola	46
3.7.4	SSP Support	46
3.7.5	FMI Support in Dymola	46
3.8	eFMI Support in Dymola	50
3.9	New libraries	56
3.9.1	Process Modeling Library	56
3.10	Modelica Standard Library and Modelica Language Specification	58
3.11	Documentation	58
3.12	Appendix – Installation: Hardware and Software Requirements	59
3.12.1	Hardware requirements/recommendations	59

3.12.2 Software requirements 59

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 2024x:

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 59 for more information. **Notes:**

- From Dymola 2020x, Visual Studio C++ compilers older than version 2012 are no longer supported.
- From Dymola 2022x, Visual Studio 2013 is not supported anymore, due to the logistics of supporting multiple old versions for all solvers. (Visual Studio 2012 is however still supported, 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 2024x has limited support for the MinGW GCC compiler, 32-bit and 64-bit. For more information about MinGW GCC, see section “Compilers” on page 59, the section about MinGW GCC compiler.

WSL GCC (Linux cross-compiler)

Dymola 2024x has limited support for the WSL (Windows Subsystem for Linux) GCC compiler, 64-bit. For more information about WLS GCC, see section “Compilers” on page 59, the section about WSL GCC compiler.

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 63 for more information.

2 About this booklet

This booklet covers Dymola 2024x. 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 2024x

3.1 Introduction

3.1.1 Additions and improvements in Dymola

A number of improvements and additions have been implemented in Dymola 2024x. In particular, Dymola 2024x provides:

- Support for working with the 3DEXPERIENCE database using the “Design with Dymola” app (page 29)
- Support for FMI
 - Naming of imported FMU (page 48)
 - Support for FMI 3.0:
 - Terminals and icons (*beta feature*) (page 47)
 - Extended co-simulation support:
 - Hybrid co-simulation (*beta feature*) (page 47)
 - Intermediate update (page 47)
 - Early return (page 47)
 - Event handling (page 47)
 - Variable step size communication (*beta feature*) (page 48) (also for FMI 2.0)
- Support for eFMI (page 50):
 - Support for eFMI Behavioral Model container generation
 - Further convenience functions for common build activities
 - Improved build activity feedback
 - Background build processes
 - Customized build environments
 - Error-safe default initialization
- Sparse solver for system of equations (page 14)
- Support for Microsoft Windows 11 (page 22)
- Support for using Clang as code generator instead of native Visual Studio or, for WSL, (normally) GCC, for Windows (page 22)
- “Features Under Development” – preview of features to be formally released in a later version (page 24)

3.1.2 New and updated libraries

New libraries

- Process Modeling Library

For more information, see “New libraries” starting on page 56.

Updated libraries

The following libraries have been updated:

- Aviation Systems Library, version 1.5.0
- Battery Library, version 2.6.0
- Brushless DC Drives Library, version 1.4.0
- ClaRa DCS Library, version 1.7.2
- ClaRa Grid Library, version 1.7.2
- ClaRa Plus Library, version 1.7.2
- Claytex Library, version 2023.2
- Claytex Fluid Library, version 2023.2
- Cooling Library, version 1.5.1
- Dassault Systemes Library, version 1.11.0
- Design, version 1.2.1
- Dymola Commands Library, version 1.16
- Dymola Embedded Library, version 1.0.2
- Dymola Models Library, version 1.7.0
- eFMI Library, version 1.0.2
- Electric Power Systems Library, version 1.6.3
- Electrified Powertrains Library (ETPL), version 1.8.0
- Fluid Dynamics Library, version 2.16.0
- Fluid Power Library, version 2023.2
- FTire Interface Library, version 1.2.0
- Human Comfort Library, version 2.16.0
- HVAC (Heating, Ventilation, and Air Conditioning) Library, version 3.1.0
- Hydrogen Library, version 1.3.9
- Multiflash Media Library, version 1.1.3
- Pneumatic Systems Library, version 1.6.1
- Testing Library, version 1.7.0
- Thermal Systems Library, version 1.12.0
- Thermal Systems Mobile AC Library, version 1.12.0
- VeSyMA (Vehicle Systems Modeling and Analysis) Library, version 2023.2
- VeSyMA - Engines Library, version 2023.2

- VeSyMA - Powertrain Library, version 2023.2
- VeSyMA - Suspensions Library, version 2023.2
- VeSyMA2ETPL Library, version 2023.2
- Visa2Base, version 1.15
- Visa2Paper, version 1.15
- Visa2Steam, version 1.15

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

Unsupported libraries

For Windows and Linux

From Dymola 2024x, the following libraries are not included in the Dymola distribution:

- PowerTrain library
- SmartElectricDrives library

Note that the VeSyMA - Powertrain Library is still supported; this library replaces the PowerTrain library.

For Linux

In addition to the above, the following libraries are currently not supported on Linux.

- Multiflash Media Library
- Process Modeling Library

3.2 Developing a model

3.2.1 Minor improvements

Improved line attributes handling when making a connection

When you create a connection, certain attributes are copied from the connector's graphical elements to the newly created connection line. This copying has now also include:

- Attributes from an invisible graphical primitive.
- The Smooth attribute.

This means that the following attributes are copied in Dymola 2024x, if the graphical primitive defines it:

Attribute	Comment
Color	
Line pattern	If "None", default solid line is used.

Thickness	If the thickness is > 0.
Smooth	

Improved handling of the context command **Create Connector** when creating a connection

In Dymola 2023x Refresh 1, for components with many connectors close together, you get a dialog to help you select the wanted connector for creating the connection and corresponding connector when dragging a connection line and using the context menu entry **Create Connector**.

In Dymola 2024x, you also get this dialog when components are too close together. A way to test this is to shrink the demo Coupled Clutches and try to create a connection from the lump of components and then right-click and selecting **Create Connector**. You might get a dialog like:





Create Connection

×

Select connector to create connection from below.

Highlight:

Show items In highlighted group

Name	Value	Unit	Description
▼ clutch1			
▼  flange_a			Left flange of compliant 1-dim. rotational component
phi		rad	Absolute rotation angle of flange
tau		N.m	Cut torque in the flange
▼ J1			
▼  flange_a			Flange of left shaft
phi		rad	Absolute rotation angle of flange
tau		N.m	Cut torque in the flange
▼  flange_b			Flange of right shaft
phi		rad	Absolute rotation angle of flange
tau		N.m	Cut torque in the flange
▼ torque			
▼  flange			Flange of shaft
phi		rad	Absolute rotation angle of flange
tau		N.m	Cut torque in the flange

connect(clutch1
);

OKCancel

Notes:

- This feature is also used for the context command **Create Node**.
- What result you get depends on the size of the components and where you click when you start dragging the connection. If you get a hit on a connector, you get the corresponding connection and node directly.

Option to select if you want to have choices (“values”) included with the descriptions for drop-down boxes

You can now select if you want to have the choices (“values”) included with the description of drop-down boxes. If you don’t want the choices included, you can do this in two ways:

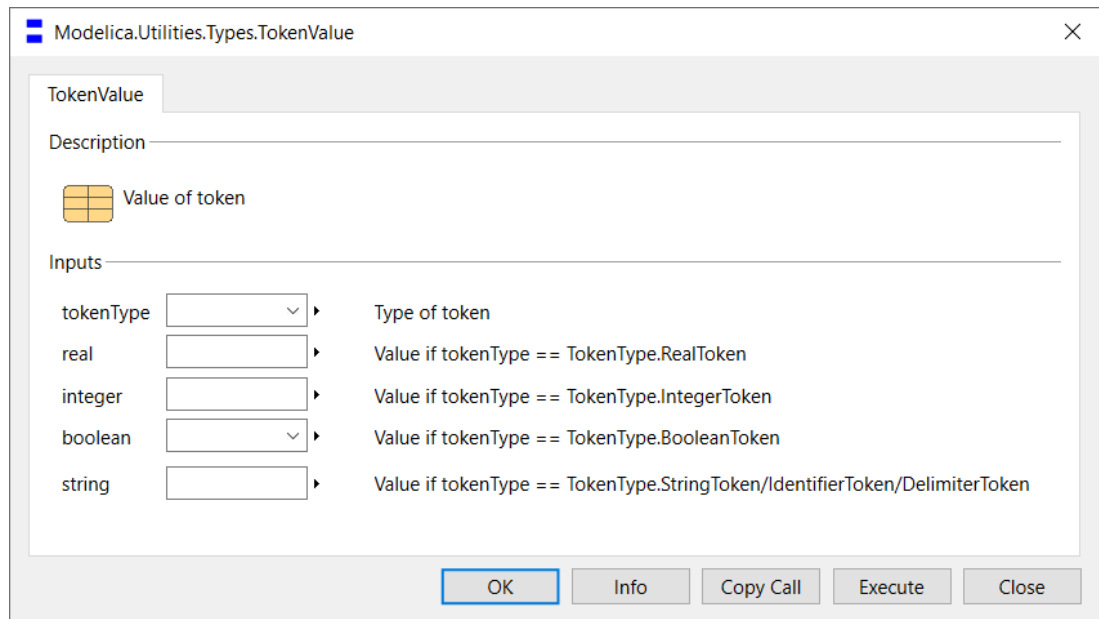
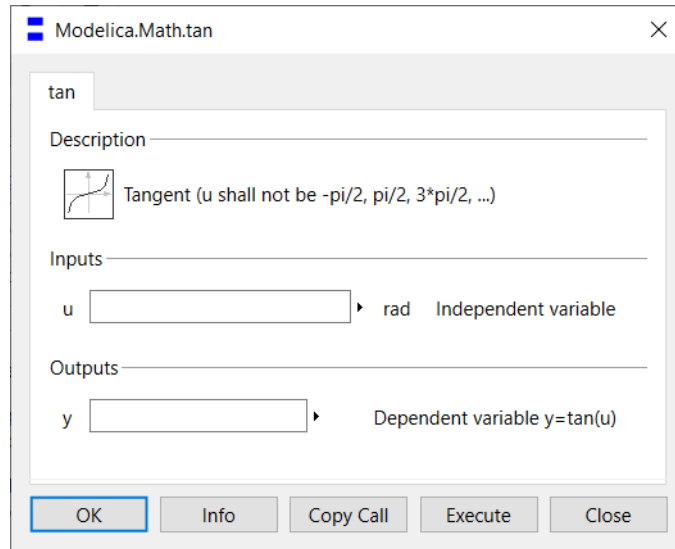
- Using the annotation `annotation(__Dymola_includeChoice=false)`. This annotation works both with `choices` and with `choicesAllMatching`.
- Using the flag `Advanced.Editor.IncludeChoiceWithDescription`. This flag can have three values:
 - 0 – off
 - 1 – Include except for Boolean (This is the default.)
 - 2 – on

(The choice handling in Dymola 2023x Refresh 1 corresponds to the flag having the value 2, in earlier versions the flag having the value 0. Note that this explains why a number of figures in the manuals now by default are different from the corresponding GUI in Dymola 2024x.)

Note that the annotation has precedence over the flag.

Icon displayed (if present) in the function call dialog and the record creation dialog

Function calls and records can have icons symbolizing them in the package browser. These icons (if present) are now displayed to the left of the description text when you call a function or create a record. Two examples:



Improved handling of graphical elements in a protected extends-clause

Graphical annotations (e.g. connector symbols) that are inherited through an extends-clause in the *protected* section will no longer be shown in the icon. They will however be shown in the diagram layer.

Easier interactive testing of libraries that are to be encrypted

Previously, to test interactively a library in its encrypted form, you had to encrypt it and reload it. If anything had to be changed, you had to reload the source version of the library again.

Now you can set the flag `Advanced.Editor.ActAsIfEncrypted = true` to force Dymola to treat the source library as encrypted when testing it. (The flag is by default `false`.)

Typical interactive testing is to use the package, dragging components, showing necessary icons and parameters, etc. As an example, this can reveal if the package is too concealed.

Reading Simulation Resource Meta Data (SRMD) from an external file

The Meta data editor in Dymola has been extended with a command **Read SMRD file** to read data from a Simulation Resource Meta Data (SRMD) external file. The below figure shows the result after reading data from such a file:

The screenshot shows the 'Dymola' window with the 'User meta data of Unnamed.' tab. The 'Category' dropdown is set to 'Simulation Resource Meta Data (resourceMetaData.srmd)'. Below the dropdown is a table with two columns: 'Key' and 'Description'. The table contains the following data:

	Key	Description
1	Type: org.prostep.srmd.mic-core.administrative-data=	
2	model.name	Permanent Magnet Synchronous Machine in abc
3	model.identifier	EMPSM3E01MSEREF
4	model.description	This is a model of a Permanent Magnet Synchronous ...
5	release	20XX.Y.Z
6	release.date	May 1, 2022
7	release.type	internal-release
8	model.supplier	Robert Bosch GmbH, RB-CoC Simulation

At the bottom of the window, there are three buttons: 'Read SRMD file' (highlighted with a red box), 'Read Template', 'OK', and 'Cancel'.

The SRMD file format is specified by the ProSTEP organization.

3.3 Simulating a model

3.3.1 Sparse solver for system of equations

In Dymola 2024x, sparse linear solvers may be used when solving linear and nonlinear systems of equations. When a model contains algebraic loops, Dymola may generate systems of equations to solve such loops. These systems are listed under "Statistics" in the translation log.

Systems of equations detected to be large and sparse enough to use sparse

When a system is large and sparse, using sparse linear solvers may be orders of magnitudes faster compared to using conventional dense solvers. When such systems are present in a model, Dymola will automatically detect them and give a translation log message, including which systems are large and sparse enough, and how large and sparse they are. Size refers to the number of variables to solve for. Whereas density is the number of structurally non-zero Jacobian elements divided by the total number of elements in the Jacobian:

- ▼ ⓘ Sparse solvers enabled: true
 - > ⓘ Enabled for integrator Jacobian: false
 - ▼ ⓘ Enabled for systems of equations: true
 - ⓘ Number of systems sparse enough: 1.
 - ⓘ System: size, density:
simulation.linear[1]: 100, 0.0396
 - ⓘ Using OpenMP, set number of cores using Advanced.NumberOfCores.

Additionally, systems that are close to large and sparse enough are also logged, see below section.

By default, sparse solvers are disabled, but can be enabled by setting the flag `Advanced.Translation.SparseActivate = true`. When enabled, sparse linear solvers will be used for all systems that Dymola determines to be large and sparse enough. Translation log and simulation log messages confirms the selection:

- ▼ ⓘ Sparse solvers enabled: true
 - > ⓘ Enabled for integrator Jacobian: false
 - > ⓘ Enabled for systems of equations: true
 - ⓘ Using OpenMP, set number of cores using Advanced.NumberOfCores.

Using SuperLU sparse solver for 2 systems of equations.

Other than the potential gain in efficiency, enabling sparse normally has a very small effect on linear systems of equations; the only differences are caused by round-off errors.

For nonlinear systems, the differences between sparse and dense linear solvers are bigger. The reason is that the nonlinear solver uses the linear solver in a different way when sparse is enabled. The differences are mostly seen when a nonlinear system is difficult to solve.

Please note that for system of equations with a numeric Jacobian, the beta feature grouping, activated by setting the flag `Advanced.Beta.NonlinearSystemGrouping = true` is required to use sparse solvers. For more information about this feature, see section “Grouping for nonlinear systems of equations with numeric Jacobian” on page 29.

Note that, already in previous releases, sparse linear solvers were available for solving the systems of equations posed inside the integrators. In this case, the sparse solvers handle both analytic and numeric Jacobians without any prerequisites.

Note also that if sparse in general is activated (by the flag `Advanced.Translation.SparseActivate = true`), you may disable sparse specifically for the systems of equations or the integrator Jacobian using the flags:

- For systems of equations: `Advanced.Translation.SparseActivateSystems = false`
- For integrator Jacobian: `Advanced.Translation.SparseActivateIntegrator = false`

(The flags are by default `true`.) If sparse in general is not activated, the above flags are ignored.

Systems of equations detected to be *almost* large and sparse enough to use sparse

Systems that are close to large and sparse enough are also logged:

- ▼ ⓘ Sparse solvers enabled: false
 - > ⓘ Enabled for integrator Jacobian: false
- ▼ ⓘ Enabled for systems of equations: false
 - ⓘ Number of systems sparse enough: 0.
 - ▼ ⓘ Systems close to sparse, but not sparse enough:
 - ⓘ System: size, density:
initialization.linear[17]: 101, 0.186746
simulation.linear[16]: 102, 0.185602
 - ⓘ Set minimum number of variables required using `Advanced.SparseMinimumStates` and maximum density allowed using `Advanced.SparseMaximumDensity`.

In such cases, you may try to experiment a bit with `Advanced.Translation.SparseMaximumDensity`, changing it, for example, from the default value of 5.0 to 20.0 and see what the outcome is. (You also have to set the flag `Advanced.Translation.SparseActivate = true` to get sparse handling.)

3.3.2 Plot tab

Further improvements for predefined plots in annotations for figures

Already Dymola 2021x supported much of the now accepted proposal **Annotation for Predefined Plots**, which is now included in the Modelica Language Specification version 3.6¹, but some features were not implemented. Some of these were included in the previous Dymola 2023x Refresh 1.

However, Dymola could not create clickable link texts, that is, as examples:

- The Modelica link `%[ModelicaBlocks](modelica:///Modelica.Blocks)` was displayed as the non-clickable text `ModelicaBlocks`.
- The arbitrary link `%[Modelica Homepage](http://www.modelica.org)` was displayed as the non-clickable text `Modelica Homepage`.

Now these are clickable links, and displayed as such.

3.3.3 Scripting

The built-in function `Execute` improved

The built-in function `Execute` has been improved to support also synchronous calls and arbitrary commands. For Windows, the option of silent execution is supported, that is, no terminal window is opened when calling the function.

New input argument `wait`

The new Boolean input argument `wait` has the default value `true`, meaning that when calling the function, no other operation can be started in Dymola until the execution of the function has terminated (synchronous call). This is the most common use, since you often want the results of the call to be used in further Dymola processing. If needed, you can stop the execution of the function from within Dymola, by the command **Simulation > Stop**.

If you set `wait` to `false`, the function call will be asynchronous, meaning that the execution of the function call will be in parallel with the Dymola execution. Asynchronous execution should be used with care; you have to ensure that further operations in Dymola do not interfere with the ongoing execution of the function call, causing unexpected results. Note also that asynchronous function calls cannot be stopped from within Dymola, you must use, for example, the Task Manager in Windows, where you can find the called program as a subtask of Dymola.

New input argument `terminal`

The new Boolean input argument `terminal` has the default value `false`, meaning that the function call is silent; no terminal window is opened when the call is executed. If you set

¹ The accepted proposal is now included in section 18.2.2 *Figures* [annotations] and the term “predefined plots” is not used anymore. In the Dymola User Manual, this change has also been implemented.

terminal to true, a new terminal window will be opened for each call. This is only supported on Windows.

Example in Windows

In Windows, the call

```
Execute(file = "cmd /c if exist foo.txt ( test.bat & pause )  
else ( pause )", wait = true, terminal = true)
```

will execute the batch file `test.bat` if the current directory contains the file `foo.txt`. It will always pause at the end of the execution and require the user to press any key in the newly opened terminal window, only thereafter can further operations be started in Dymola (that is, Dymola is synchronized with the program and waits for it to finish).

Note. If you have used `Execute` to, for example, opening documents by adding some own code, you may have to change that to use the general Windows syntax `cmd c/ start`. As an example, you can open the document `MyDoc.pdf` (located in the current directory) by the call

```
Execute(file = "cmd /c start MyDoc.pdf")
```

The built-in function `importFMU` improved

The built-in function `importFMU` now has a new String input argument `modelName`. The argument specifies the name of the generated Modelica model that wraps the imported FMU. The default is formally an empty string, but an empty string means that the name will be taken from the name of the imported FMU file, plus the string “_fmu” in the end of the name. (Importing the FMU file `CoupledClutches.fmu` with `modelName` being the default empty string gives the generated Modelica model name `CoupledClutches_fmu`.)

If you specify a name in `modelName` that name will be used instead.

(The new argument is added last in the list of arguments, not to break compatibility if the built-in function has been called without specifying the argument names when calling the function.)

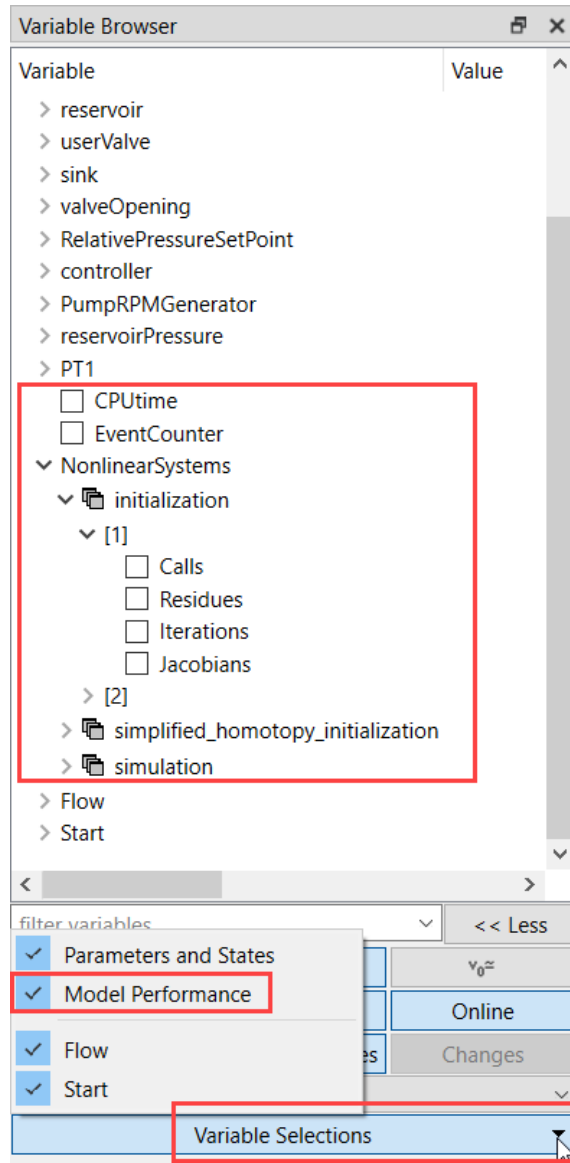
For the corresponding GUI and a note about harmonizing default naming, see section “Naming of imported FMU” on page 48.

3.3.4 Minor improvements

Variable selection: Including variables for CPU time, event counting, and nonlinear solver diagnostics in a new predefined variable selection

Since previous versions, if you, in the simulation setup, the Translation tab, activated the option **Include variables for CPU time, event counting, and nonlinear solver diagnostics**, you got six extra variables in your simulation result file (the nonlinear solve diagnostics includes four of these variables).

In Dymola 2024x, you also have these variables available in a predefined variable selection **Model Performance**, if you have activated the above option. An example:



Improvement of start-value priority when having both initial equations and start values

Some commonly used models, for example, `Modelica.Blocks.Continuous.Integrator` and `Modelica.Blocks.Continuous.PID`, have both initial equations and start-values, and in that case, it makes no sense to use the start-values. This is now by default implemented by the flag:

```
Advanced.Translation.DownPrioritizeStartValueWithInitial = true
```

The flag is by default `true`, you can get the old behavior by setting the flag to `false`.

Note that for the above to happen, the initial equation and the start-value use the same value, and there are no related alias variables with start values.

Minor update of start-value discrimination

The start-value priority created by using start-value confidence number (level) is now used for prioritizing start-values for the simulation problem as described in Modelica Language Specification version 3.6, section 8.6.2. Previously this priority was only used for prioritizing guess-values for systems of equations.

To disable the new behavior, you can set the flag

```
Advanced.Translation.OldStartValueLevelHandling = true
```

(The flag is by default `false`.)

Improved information about selection of default initial conditions

In Dymola 2024x, there is, in the translation log, also information if start values were *not* selected. Consider the simple case:

```
model Unnamed
  Real x(start=1);
  Real y(start=1);
  Real z(start=2, fixed=true)=x+y;
equation
  der(x)=1-x+y;
  der(y)=1-y;
end Unnamed;
```

When translating, the translation log will now look like (the framed message is new):

- ✓ ⓘ Translation of [Unnamed](#)
 - ⓘ The DAE has 3 scalar unknowns and 3 scalar equations.
 - ✓ ⚠ The initial conditions are not fully specified.
Dymola has selected default initial conditions.
 - ⚠ Assuming fixed start value for the continuous states:
y(start = 1)
 - ⚠ There were 1 continuous states with start values that were not selected:
x(start = 1)
 - > ⓘ Statistics
 - ✓ ⓘ Selected continuous time states
 - ⓘ Statically selected continuous time states
 - x
 - y
 - ⓘ Finished
 - ⚠ WARNINGS have been issued.
 - ⓘ Finished

Logging of external function calls during translation

You can log external function calls during translation by setting the flag:

```
Advanced.Translation.Log.FunctionCallEvaluation = true
```

(The flag is by default `false`.)

An example of function calls during translation is setting of scalar parameter values from CSV files. This can mean many calls to read the matrix size, and then to read the matrix, in order to read just one value. The logging enables you to find out if this is an issue, that is, if you have numerous such calls.

Bound checking for variables – present flag now accessible in GUI

In the **Debug** tab of the simulation setup (reached by the command **Simulation > Setup**, the **Debug** tab), a new setting is displayed:

Simulation Setup

General Translation Output **Debug** Compiler Realtime FMI Export FMI Import

General simulation debug information

- ☒ Normal warning messages
- ☐ Debug settings included in translated or exported model
- ☐ Store variables after failed initialization
- ☒ Include function call in error messages

Simulation analysis Timers

- ☐ Events during initialization and simulation
- ☐ Which states that dominate error
- ☐ Provide variable dependencies and equation incidence
- ☐ Generate block timers
- ☐ Generate function and FMU timers

Nonlinear solver diagnostics

- ☐ Detailed logging of failed nonlinear solutions
- ☐ Nonlinear solution summary
- ☐ Nonlinear solution
- ☐ Nonlinear iterations
- ☐ Details
- ☐ Log singular linear

State variable logging

- ☐ Dynamic state selection
- ☐ Final selection of states

Min/Max assertions

- ☐ All variables
- ☐ Non-linear iterations variables

Allowed error 0

Semi-linear error 0.5

- ☒ Use bounds for simplification
- ☐ Log when using bounds for simplification

Store in Model ☒ Automatically store General and Inline integration settings **OK** Cancel

This setting corresponds to the flag

```
Advanced.AssertInsideMinMaxNonReversibleLimit = 0.5
```

This flag (that is present since long), is used to specify an “additional allowed error” when simplifying semiLinear in general. This can be used for, as example, non-reversible flow. By adding it in the GUI as above, it is more visible.

Improved editing of input fields in the variable browser

The editing of the input field in the variable browser is improved. The behavior is now:

- If you click once on a parameter input field, it becomes an input editor, and all the contents is selected. If now you start typing, the entire contents is replaced by what you type.
- If you click on a parameter input field, and then click again (not double-clicking) in the (now) open input editor, you get an insertion position in the editor, without erasing the existing contents.
- If you double-click on a parameter input field, the field becomes an input editor, with the double-clicked item selected.

3.4 Installation

For the current list of hardware and software requirements, please see chapter “Appendix – Installation: Hardware and Software Requirements” starting on page 59.

3.4.1 Minor improvements

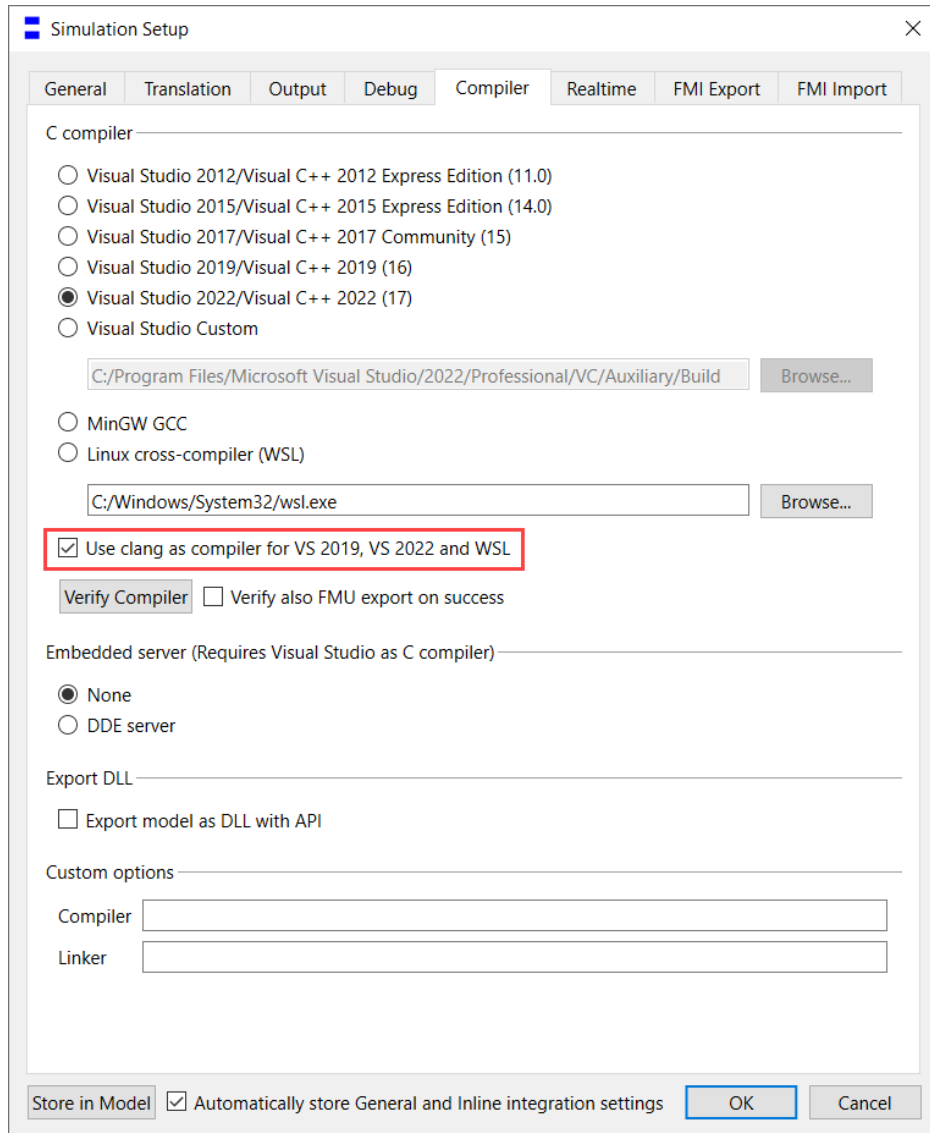
3.4.2 Installation on Windows

Microsoft Windows 11 supported

Dymola 2024x supports Microsoft Windows 11.

Clang compiler supported with Visual Studio and Linux cross-compiler (WSL)

If you first select to use Visual Studio 2019, Visual Studio 2022, or Linux cross-compiler (WSL) as compiler, you can then select to use Clang as code generator instead of native Visual Studio, or, for WSL, (normally) GCC, by the new setting **Use clang as compiler for VS 2019, VS 2022, and WSL**:



Using Clang as code generator can make the simulation less time-consuming, but the improvement is model dependent.

(This option is by default not activated.)

Note that you need to install Clang compiler support to use it.

- For Visual Studio, search and select for components named “clang” in the Visual Studio Installer.
- For WSL, a tip is to select Clang and try to run WSL, you will then get information how to install Clang.

3.4.3 Installation on Linux

Updated Linux version

Dymola 2024x runs on Red Hat Enterprise Linux 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 2024x it is Qt 6.5.1.
- Select Supported platforms)

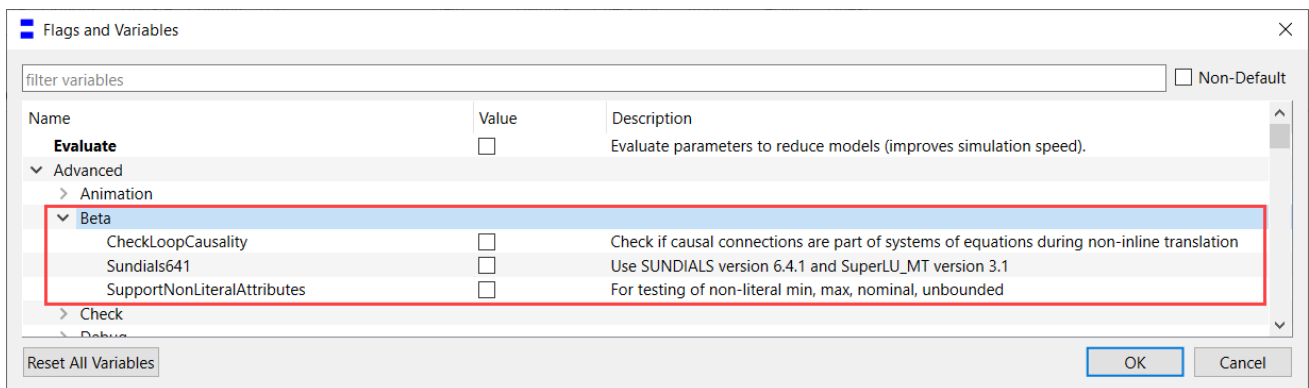
Any later version of gcc is typically compatible. In addition to gcc, the model C code generated by Dymola can also be compiled by Clang.

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. (An exception for Dymola 2024x is that the Grouping feature is also documented in the manuals, since it is so closely related to sparse handling.)

These features are grouped by `Advanced.Beta` flags in **Tools > Options > Variables...**: An example from Dymola 2023x 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.

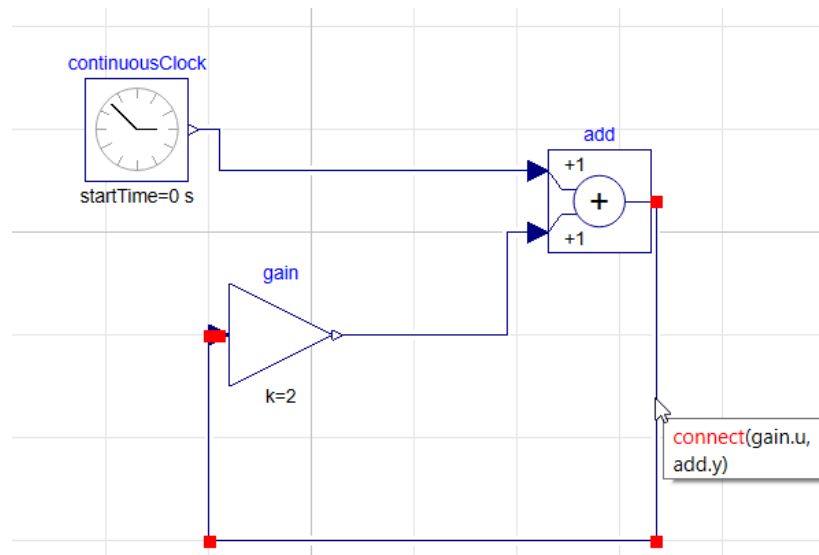
In Dymola 2023x Refresh 1, the following “under development”, features are available:

Option to get diagnostics when reversing causality

To use this option, activate the flag `Advanced.Beta.CheckLoopCausality`. We recommend also setting `Evaluate = false` when using this option, otherwise some case might be missed.

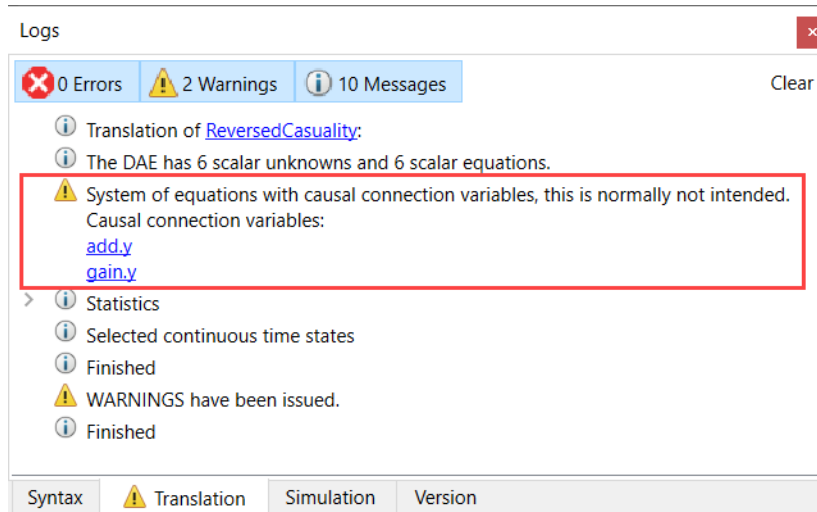
An example model might be:

```
model ReversedCasualty
  Modelica.Blocks.Math.Gain gain(k=2)
    annotation (Placement(transformation(extent={{-16,-30},{4,-10}})));
  Modelica.Blocks.Math.Add add
    annotation (Placement(transformation(extent={{46,-4},{66,16}})));
  Modelica.Blocks.Sources.ContinuousClock continuousClock
    annotation (Placement(transformation(extent={{-44,10},{-24,30}})));
equation
  connect(gain.y, add.u2) annotation (Line(points={{5,-20},{38,-20},{38,0},{
    44,0}}, color={0,0,127}));
  connect(add.u1, continuousClock.y) annotation (Line(points={{44,12},{-18,12},
    {-18,20},{-23,20}}, color={0,0,127}));
  connect(gain.u, add.y) annotation (Line(points={{-18,-20},{-20,-20},{-20,
    -60},{67,-60},{67,6}}, color={0,0,127}));
  annotation (Icon(coordinateSystem(preserveAspectRatio=false)), Diagram(
    coordinateSystem(preserveAspectRatio=false)),
    experiment(StopTime=2, __Dymola_Algorithm="Dassl"));
end ReversedCasualty;
```



Here, the connection between the add block and the gain block constitutes a reversed causality issue.

Activating the flag `Advanced.Beta.CheckLoopCausality`, and setting `Evaluate=false`, and then simulating, you will get, in the translation log:



Support for the IDA solver from SUNDIALS and upgrade to SUNDIALS 6.4.1 and SuperLU_MT 3.1

Upgrade to SUNDIALS 6.4.1 and SuperLU_MT 3.1

By setting the flag `Advanced.Beta.Sundials641 = true`, the SUNDIALS 6.4.1 and SuperLU_MT 3.1 version will be used.

This applies to the following solvers:

- CCode (SUNDIALS 6.4.1)
- CCode sparse (SUNDIALS 6.4.1 and SuperLU_MT 3.1)
- Radau, Esdirk*, Sdirk34hw sparse (SuperLU_MT 3.1)

The option can be used in the following cases:

- Simulation directly in Dymola using any of the above solver options
- FMU export with "Co-simulation using Dymola solvers" using any of the above solver options

Limitations:

- Requires Visual Studio 2015 or newer

Note that the following are currently not supported:

- Dassl, Lsodar sparse (still only uses old version of SuperLU_MT, even with the flag set)
- FMU export with "Co-simulation using Ccode"
- FMU source code export

Support for the IDA solver from SUNDIALS

The SUNDIALS IDA solver is now available as a part of the SUNDIALS 6.4.1 beta option.

IDA is a modern variable-stepsize, variable-order solver of hybrid DAEs. In the core, it implements a BDF integrator and is therefore most similar to Dassl and CVode.

Just as Dassl, IDA allows integration of DAEs. This is a benefit over SUNDIALS CVode, which is a pure hybrid ODE solver.

Among others, the Dymola IDA implementation supports:

- Sparse linear solvers for large-scale systems, by setting `Advanced.SparseActivate = true`.
- Dymola DAE mode for efficient integration of dynamic models with large algebraic loops, by setting `Advanced.Define.DAEsolver = true`. Just as the other Dymola DAE solvers, IDA DAE mode uses the efficient and accurate event handling as described in Henningsson, Olsson, and Vanfretti: *DAE Solvers for Large-Scale Hybrid Models* (<http://dx.doi.org/10.3384/ecp19157491>).

To enable the IDA beta option in Dymola 2023x Refresh 1 you need to do the following:

- Set the flag `Advanced.Beta.Sundials641 = true`
- Set the flag `Advanced.Beta.Sundials641IDA = true`
- In Simulation Setup, reached by the command **Simulation > Setup**, the **General** tab, select the algorithm **Cvode – variable order**.

The IDA solver can be used in the following cases:

- Simulation directly in Dymola
- FMU export with "Co-simulation using Dymola solvers"

Limitations:

- Requires the compiler Microsoft Visual Studio 2015 or newer.

Note that the following are currently not supported:

- FMU export with "Co-simulation using Cvode"
- FMU source code export

Down-sampling plotted curves to improve drawing speed

Drawing plots in Dymola can be considerably slowed down if curves have a large number of data points, tens or even hundreds of thousands. From a plotting perspective this is any case meaningless because we will only have a few hundred pixels to paint (horizontally or vertically), which means that every pixel is painted several times.

Where this slowdown becomes critical varies, but for e.g. a Remote desktop connection it can be as low as 10.000. When this happens, Dymola becomes very hard to use because everything is lagging.

The down-sampling has been done with some care, as just picking points at regular intervals might hide important behavior. Instead, points that either overlap completely, or intermediate points of a straight-line segment are deleted. This is not completely undetectable due to certain drawing artifacts when the lines are rendered (pixel rounding, anti-aliasing).

The flag `Advanced.Beta.Plot.DownsamplingLimit` has been introduced to set the limit where curves are down-sampled to improve drawing speed. The default is zero, which means down-sampling is off (backward compatible). Note that if you zoom in on a curve, the number of points drawn will eventually fall below this limit and any down-sampling artifacts will thus disappear.

FMI 3: Support for hybrid co-simulation

For this feature, see section “Hybrid co-simulation” on page 47.

FMI 3: Support for terminals and icons

For this feature, see section “Terminals and icons” on page 47.

FMI: Variable step size co-simulation

For this feature, see section “Variable step co-simulation” on page 48.

Differentiating discrete expressions

As an example, consider the model:

```
model Test
  Real x;if time>=0.5 and time<0.9 then 1 else 0.1;
  Real y;
equation
  der(x)=der(y);
end Test;
```

If you try to simulate this model, it will fail, and you will get the message:

```
Model error - differentiated if-then-else was not continuous:
(if time >= 0.5 and time < 0.9 then 1 else 0.1)
Value jumped from 0.1 to 1.
There is Beta-support for using impulses to handle this; please
refer to release notes.
```

If you set the flag `Advanced.Beta.ImpulseWhenDifferentiatingDiscontinuity = true` and then simulate, you will succeed, with the messages:

```
Introducing impulse to handle differentiated expression that
was not continuous:
(if time >= 0.5 and time < 0.9 then 1 else 0.1)
Value jumped from 0.1 to 1.
```

```
Introducing impulse to handle differentiated expression that
was not continuous:
(if time >= 0.5 and time < 0.9 then 1 else 0.1)
Value jumped from 1 to 0.1.
```

The problem is solved by generating an impulse so that `y` also changes discontinuously. (The flag is by default `false`.)

The reasons that this feature is currently a Beta feature are:

- Impulses currently only work when simulating normally, and not in Matlab/Simulink, FMU etc.
- The "integration" of the discontinuity is currently not fully implemented.
- The feature only handles first derivatives.
- If there are multiple relevant things changing at an event it might not be currently possible to handle.

Grouping for nonlinear systems of equations with numeric Jacobian

When a model contains algebraic loops, Dymola may generate systems of equations to solve such loops. The solving procedure involves calculations of the system Jacobian. Normally, Dymola will find these Jacobians analytically by differentiating the residual equations. However, sometimes such differentiation fails, and then Dymola falls back on using numerical approximations.

In Dymola 2024x, it is possible to speed up the numeric Jacobian approximation using grouping to reduce the number of residual evaluations. This beta feature is enabled by setting the flag:

```
Advanced.Beta.NonlinearSystemGrouping = true
```

(The default value of the flag is `false`.) For large and sparse systems, also consider enabling sparse solvers, to also speed up the Jacobian factorization.

Other than the potential efficiency gain, enabling grouping should normally have a very small effect on the overall solution procedure for the algebraic loop. In general, the Jacobian approximation should be more exact as all structural zeros are guaranteed to be zero also in the approximation.

Note. For the approximation of the ODE Jacobian, grouping is always used when applicable, independent of the value of the above flag.

3.6 Model Management

3.6.1 Support for working with the 3DEXPERIENCE database using the “Design with Dymola” app

General

From Dymola 2024x, you can install the app “Design with Dymola” in 3DEXPERIENCE. This app enables you to launch Dymola from 3DEXPERIENCE, and to use the 3DEXPERIENCE database to store and retrieve your files.

For information about installing and handling the “Design with Dymola” app, see the corresponding 3DEXPERIENCE documentation. To reach the top level of this documentation, use the link <https://www.3ds.com/support/documentation>. Note that you need a 3DEXPERIENCE ID to access the documentation.

From this top level, select **USER ASSISTANCE**, then, under 3DEXPERIENCE Platform, select the relevant language under 3DEXPERIENCE ON THE CLOUD or 3DEXPERIENCE ON PREMISE. You are now at the top level of the documentation pages.

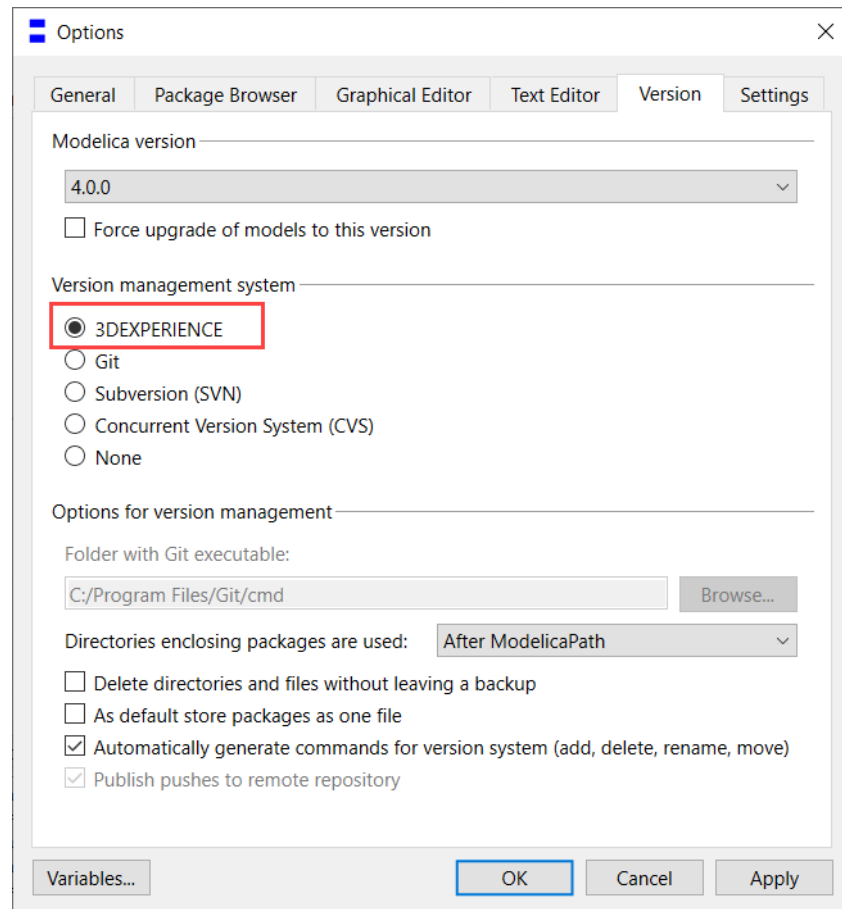
In the left pane, expand **Social and Collaborative > Multidiscipline Configured Product Development (Collaborative Design) > Design with Dymola**.

Here we assume that this app is launched, that is, Dymola is started using the “Design with Dymola” app. (It is also possible to start Dymola separately and connect to an accessible 3DEXPERIENCE versioning system, see section “Starting Dymola separately and connecting to the 3DEXPERIENCE versioning system” on page 43.)

Note that Design with Dymola is only supported for Dymola on Windows.

Selecting the new versioning system

Looking at the command **Tools > Options**, the **Version** tab, you can now select a new version management system **3DEXPERIENCE**:

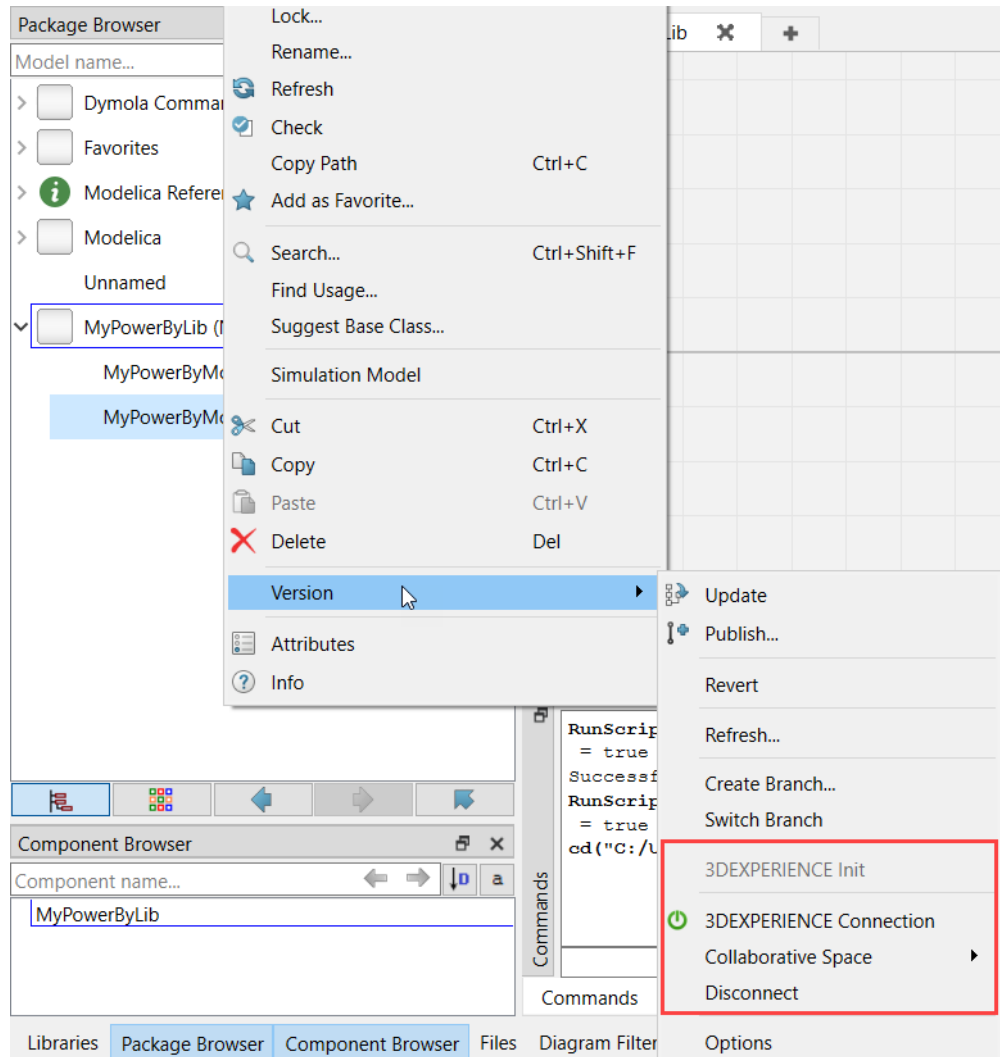


If this option is not selected, select it, and click **OK**.

In addition, clear the option **As default store packages as one file** if that option is active. (By default, it is deactivated if you select any versioning system.)

New commands



This selection gives some additional commands for 3DEXPERIENCE version handling, seen by, for example, right-clicking a file in the package browser and selecting **Version**:



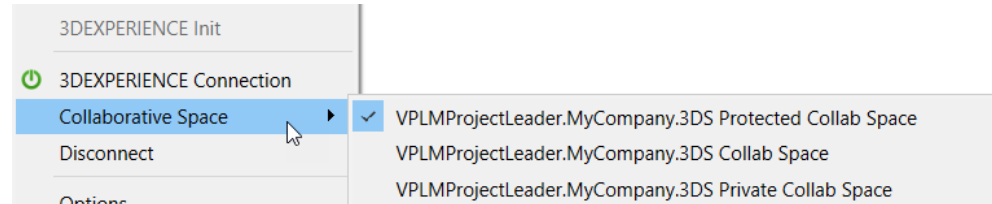
The commands are:

3DEXPERIENCE Init – creates a new 3DEXPERIENCE module; the result is that the library is marked for version management.

3DEXPERIENCE Connection – This entry starts with any of the icons:

-  - You are connected. The entry is now an information only; clicking it does not change anything.
-  - You are disconnected. You can click this entry to connect; you have to enter user and password to connect.

Collaborative Space – For selection of collaborative space. A collaborative space is an area where people with different responsibilities can work together to produce and deliver content.



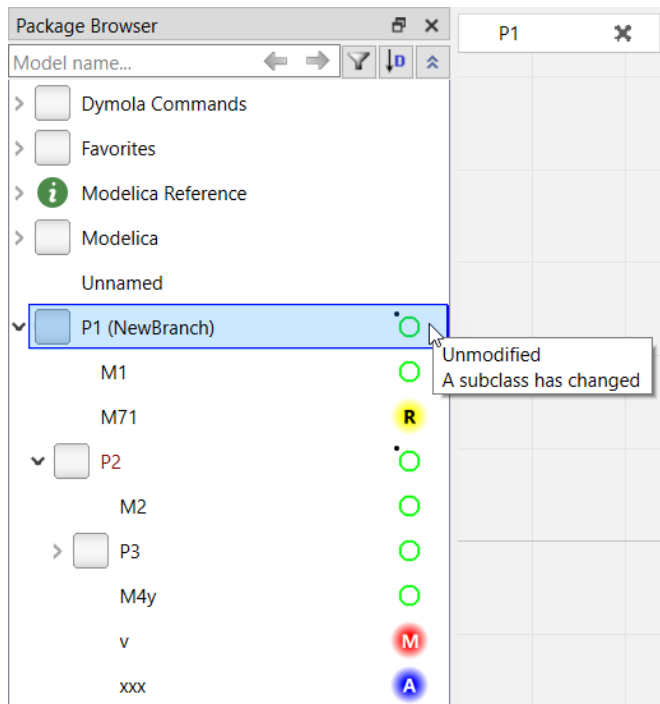
Disconnect – Disconnects from 3DEXPERIENCE. This entry is dimmed if you are disconnected.

The rest of the commands are very similar to the corresponding GIT commands; it is possible to see the support of the 3DEXPERIENCE versioning system as an extension to the Git support.

3DEXPERIENCE status visible in the package/project browser

The current 3DEXPERIENCE status is displayed in the package/project browser.

An example:



The presentation of the status:

Status	Icon	Comments
Unmodified, and all subclasses unmodified		
Unmodified, but at least one subclass modified, renamed, or added		
Modified		
Added		
Renamed		Renaming can currently not be tracked.
Untracked		
(Deleted)	<no icon>	A deleted file is not displayed in the package browser.

Examples

Init and first commits

As already mentioned above, we assume that Dymola is started using “Design with Dymola”, that you have selected **3DEXPERIENCE** as versioning system, and that you have cleared the option **As default store packages in one file**.

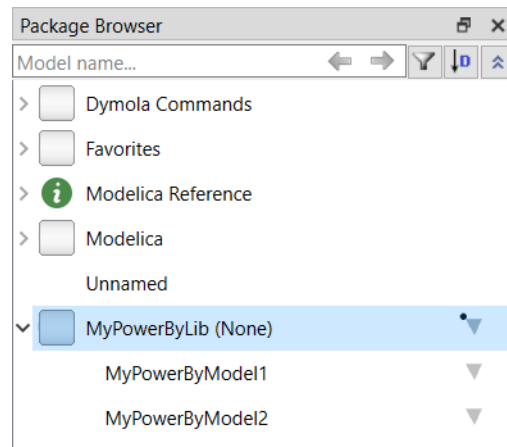
Start by creating a Modelica package **MyPowerByLib**. Notes:

- The setting **Save contents of package in one file** should already be unchecked, since you cleared the corresponding default option in the Version tab above. The content of the package should be saved as individual files.
- It is a good idea to add a description, this description can be seen by pausing over the library in the package browser, but is also displayed when searching the library in the 3DEXPERIENCE database.
- Folders containing standalone Modelica files are not yet supported.

In this library, create two models, **MyPowerByModel1** and **MyPowerByModel2**.

Save the package.

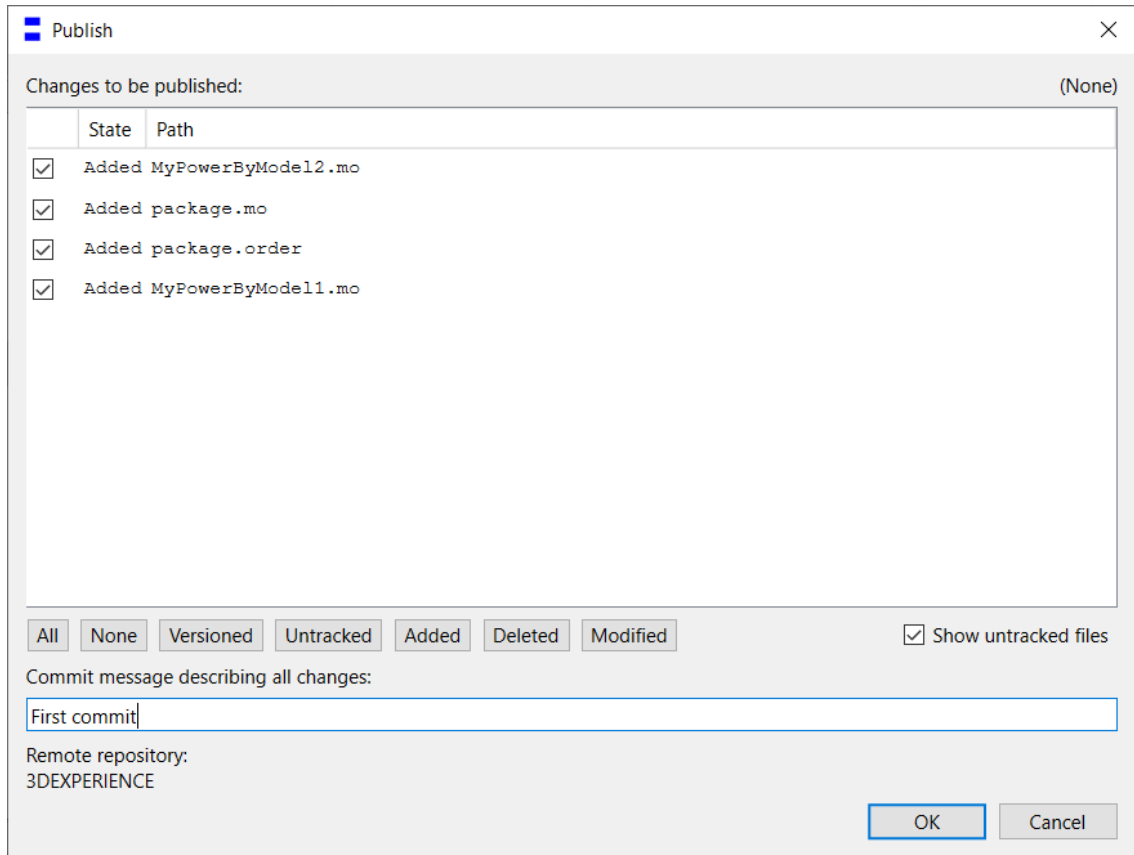
Right-click the root package **MyPowerByLib** and select **Version > 3DEXPERIENCE Init**. This command activates the package for version management. The result in the package browser is:



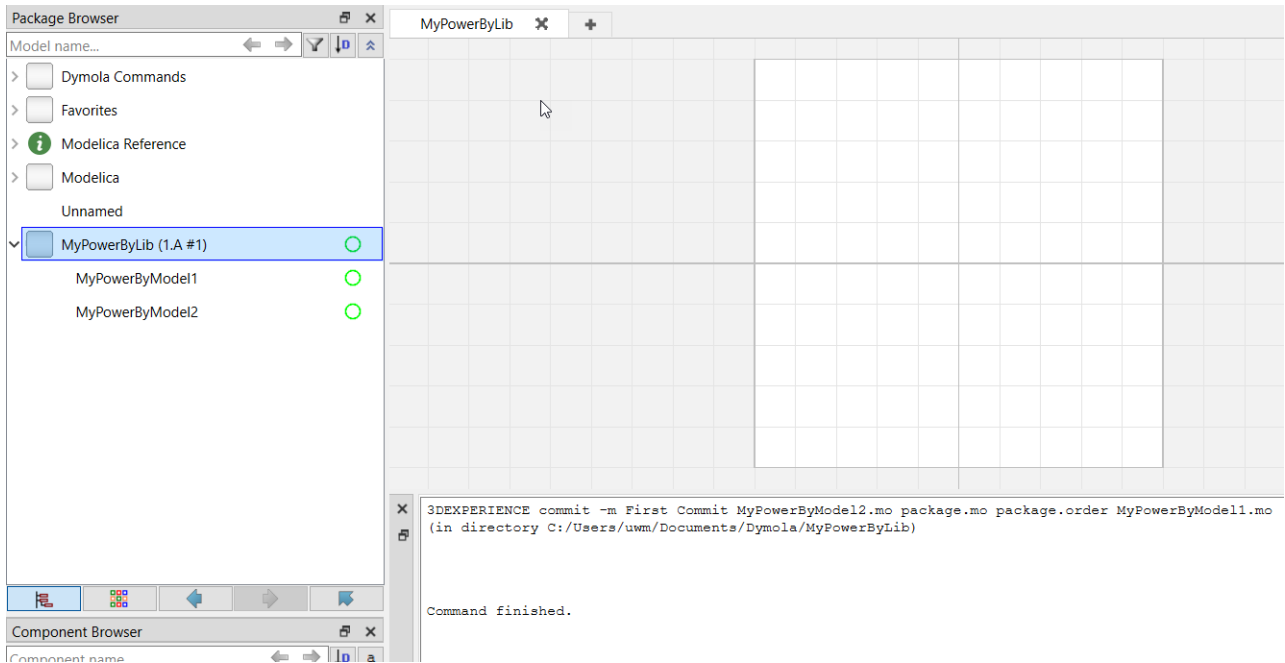
The icons that appear to the right tells that the package and models are untracked.

Now, to publish the package, right-click **MyPowerByLib** and select **Version > Publish....**

- First, an Update dialog appears. Select **Save All**.
- Then a Publish dialog appears – enter a commit message **First Commit** like below:



After clicking **OK**, the Dymola window will look like:



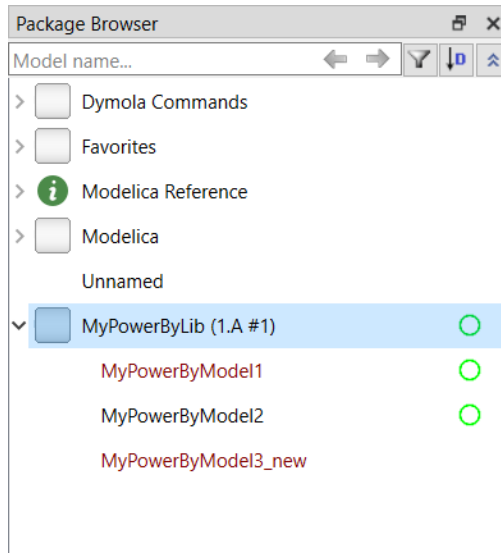
Notes:

- The icons to the right in the package browser now tells that the library and included models are unmodified, that is, they are saved and there are no local changes noted.
- For **MyPowerByLib** in the package browser there is now, in brackets, information about the name of the branch created: **1.A** and the iteration number: **#1**.
- In the commands window there is information about the versioning command performed, including information that the command was performed successfully.

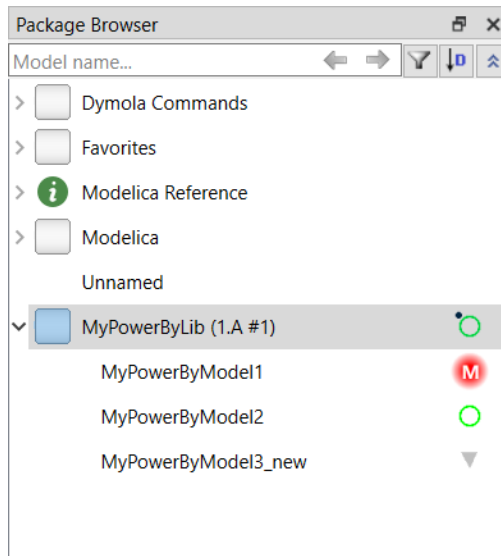
Now, having performed the first commit successfully, do the following:

- Modify **MyPowerByModel1**, by, for example, adding a graphical object in it.
- Create, in the library, a new model **MyPowerByModel3_new**.

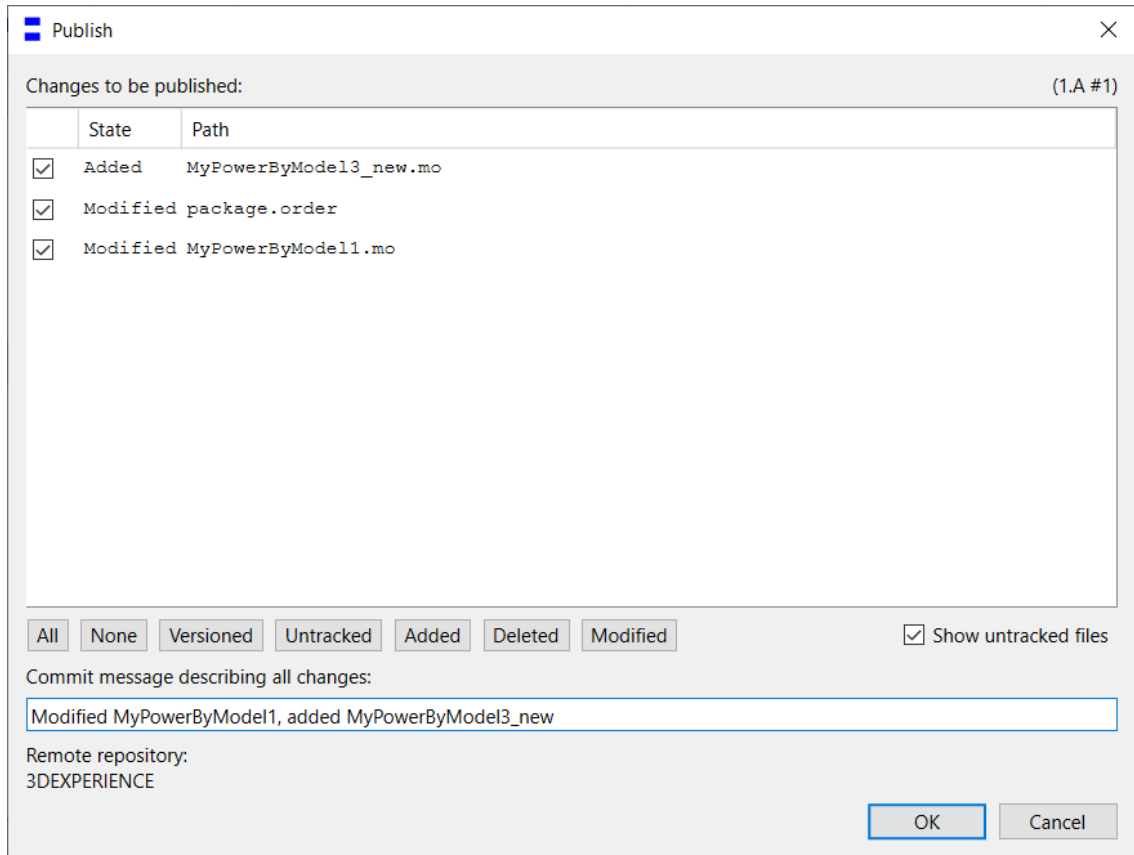
After the changes, the package browser looks like:



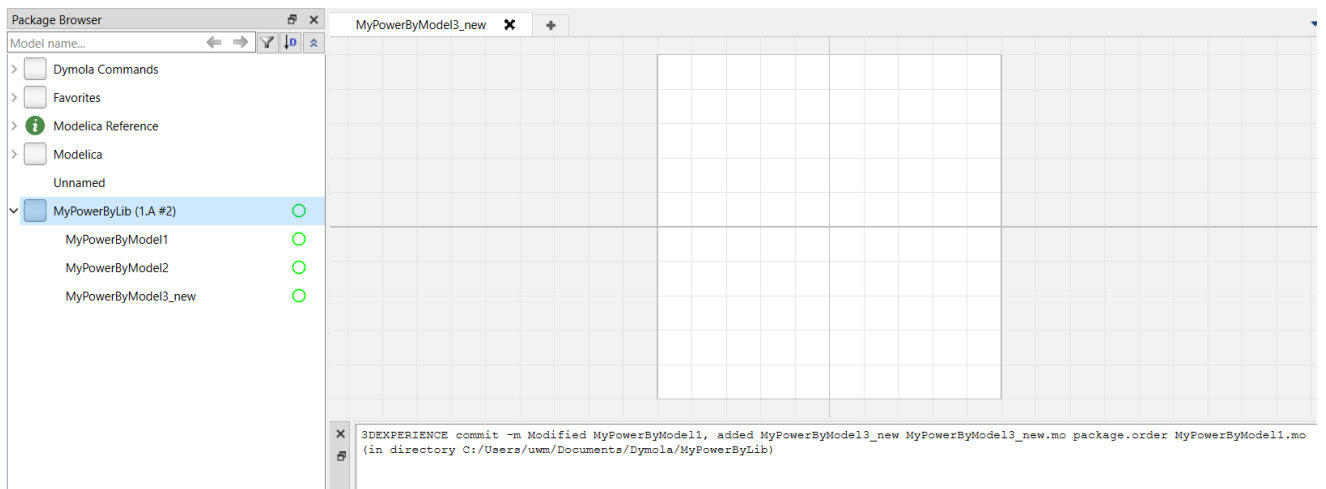
Note that the status (the icons to the right) is not updated. However, to be able to commit you must save the changes locally, and after having done this, the status is updated:



Now, perform a second commit by right-clicking **MyPowerByLib** and select **Version > Publish....** The following dialog appear (you have to add the commit message):



After clicking **OK**, the Dymola window will look like:

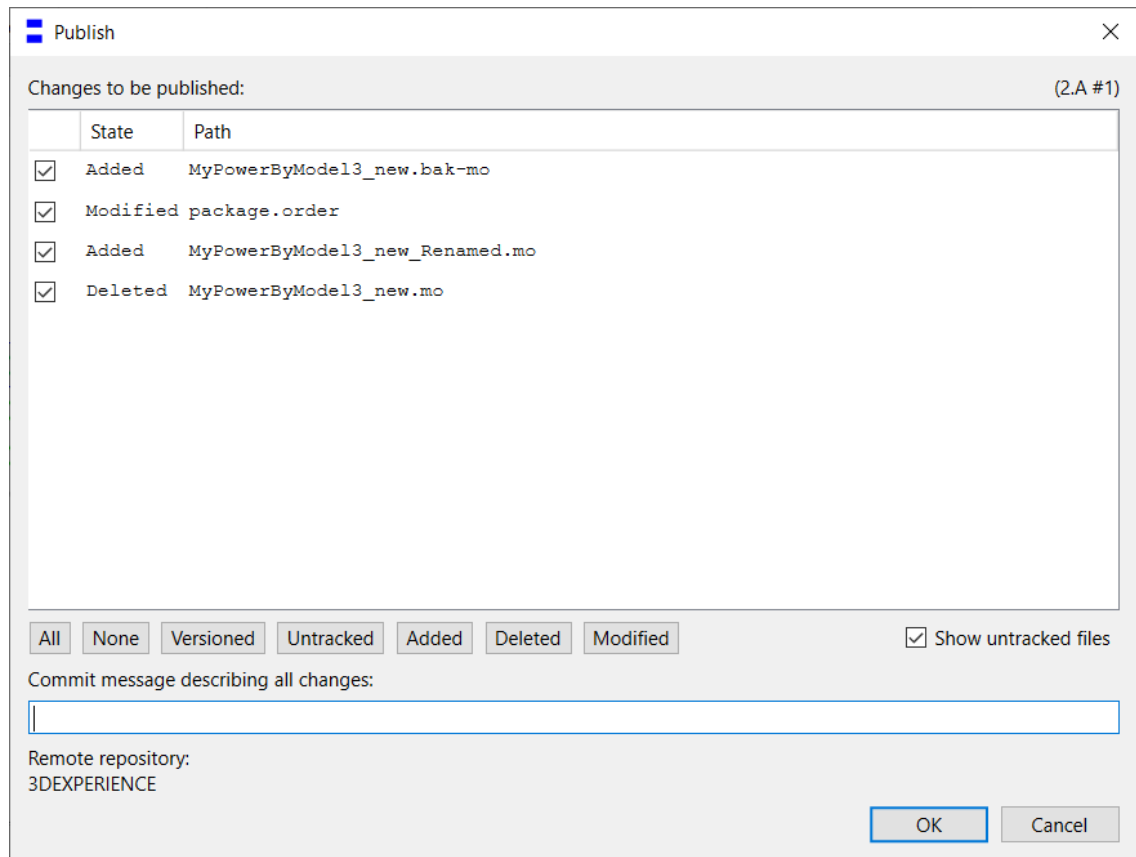


In the package browser, you can now see that for **MyPowerByLib**, a new iteration has been created, **#2**.

Renaming models

When you rename a model stored in the database (by, for example, right-clicking it in the package browser and select the command **Rename**), then the original file is deleted, and a new file is created for the file with the new name. (A backup file is also created.)

As an example, renaming the file **MyPowerByModel3_new** to **MyPowerByModel3_new_Renamed** and then publish, gives the dialog:



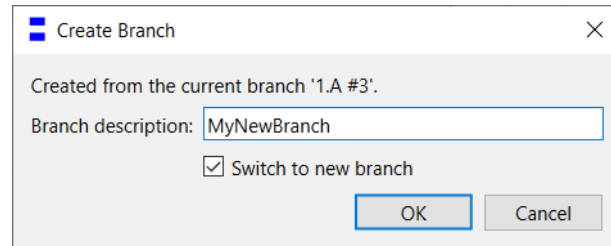
Note that this means that there is currently no traceability of renaming.

Working with branches

The following describes how to work with branches from Dymola, note however, that in 3DEXPERIENCE, the Collaborative Lifecycle app can be used to handle versioning, and a Dymola branch is seen as a revision in 3DEXPERIENCE. Thus, branches can consequently

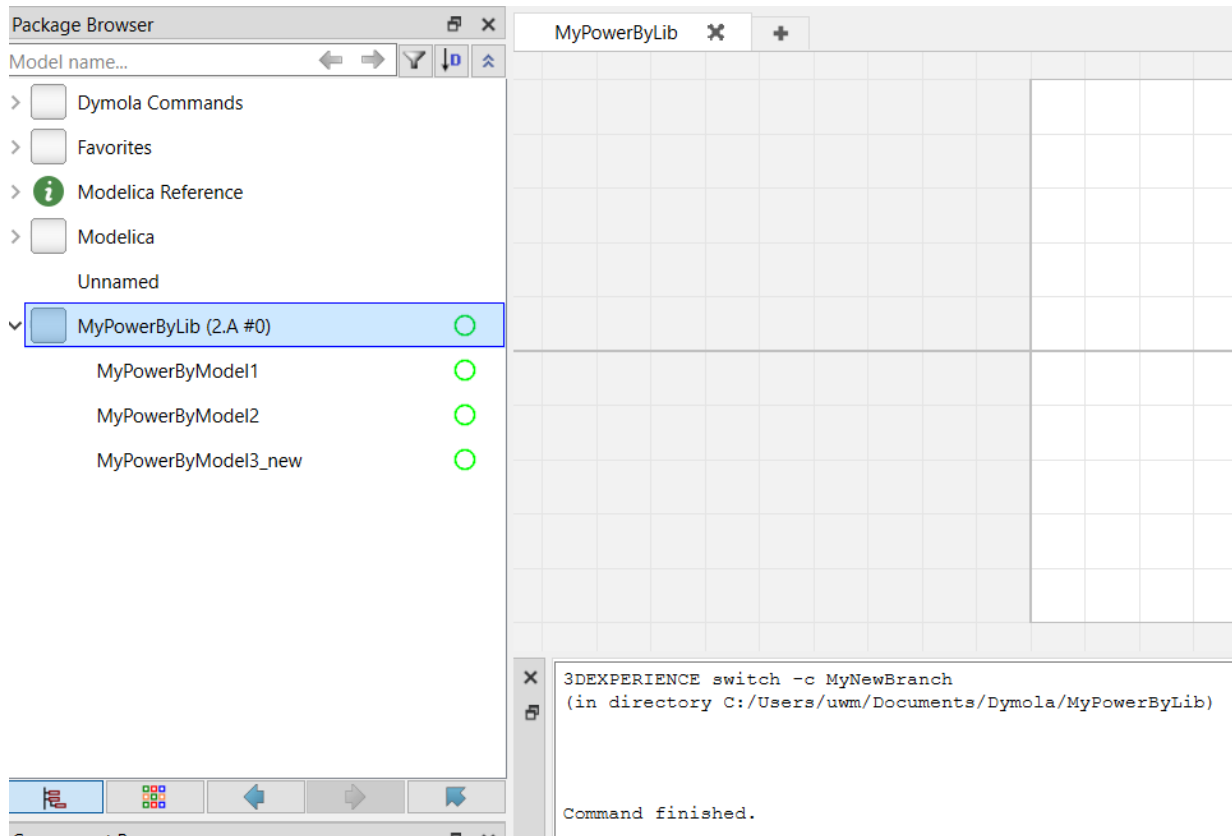
also be handled by the Collaborative Lifecycle app. For more about using this app to handle Dymola branches, see the corresponding 3DEXPERIENCE documentation [\[\[link needed\]\]](#).

To create a new branch in Dymola, you can right-click the root package **MyPowerByLib** and use the command **Version > Create Branch....** After you have performed a save, the following dialog appears. You have to enter the description of the new branch (in this example **MyNewBranch**):



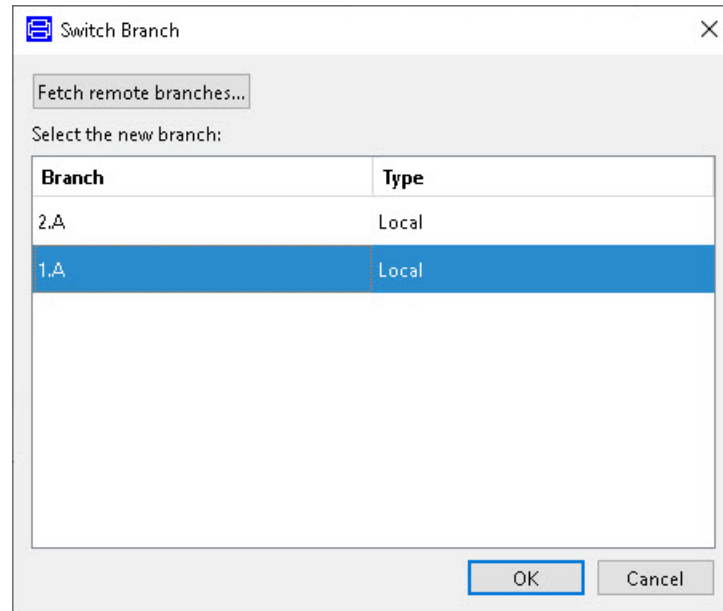
By default, you switch to the new branch when you create it, but you can clear the setting **Switch to new branch** to keep working with the present branch.

The result of the operation is:



The current branch is displayed in brackets for the relevant root package in the package browser (in this example the package **MyPowerByLib**).

To switch branch, you can right-click the root package **MyPowerByLib** and use the command **Version > Switch Branch**. After a save operation, this command displays a dialog where you can select from the available branches:



Note that in Dymola, by default, only the branches that have already been displayed in Dymola are listed. If branches have been created using the Collaborative Lifecycle app in 3DEXPERIENCE without being displayed in Dymola, you can fetch them to this dialog by clicking the button **Fetch remote branches....**

Displaying the log of the versioning

To display the versioning log of the complete library, right-click the root package (in our example the package **MyPowerByLib**), and select **Version > Log**. The log is displayed in the commands window, an example:

```
X 3DEXPERIENCE log
  (in directory C:/Users/uwm/Documents/Dymola/MyPowerByLib)

* commit #1PowerBy Dymola library for testing
| A package.order
| A package.mo
| A MyPowerByModel1.mo
| A MyPowerByModel2.mo
| A MyPowerByModel3_new.mo
*****

Command finished.
```

To display the versioning log of a model, right-click the model and select **Version > Log**. An example for **MyPowerByModel2**:

```
X 3DEXPERIENCE log MyPowerByModel2.mo
  (in directory C:/Users/uwm/Documents/Dymola/MyPowerByLib)

* commit #1PowerBy Dymola library for testing
| A MyPowerByModel2.mo
*****

Command finished.
```

Handling concurrent modifications

In a given branch, if the current iteration in Dymola is not the most recent iteration (that is, another user has created later iterations), then it is mandatory to update the model before publishing the local modification. A warning message is displayed if you are trying to publish the iteration that is not the most recent one.

In case concurrent modifications are made by two users to the same file, when update is performed, then the local file is copied to a file with extension terminated with “~”, before the conflicting file is uploaded from the database. Thus, you can (manually) compare the files and solve the merge, with your preferred tool for comparison/merge.

Handling model maturity

Maturity is a 3DEXPERIENCE attribute that can only be handled by the Collaborative Lifecycle app. If maturity for a model has been changed in such a way that modifications are not allowed (that is, the maturity is frozen, released, or obsolete), a message is displayed in Dymola if you try to commit a local modification for such a file.

Starting Dymola separately and connecting to the 3DEXPERIENCE versioning system

If the 3DEXPERIENCE versioning system is accessible, you can also start Dymola separately and connect to the 3DEXPERIENCE versioning system. Once you have started Dymola, you can connect by first selecting **3DEXPERIENCE** as version management system in the **Version** tab of the **Options** menu (reached by the command **Tools > Options**, the **Version** tab), and then selecting the command **Tools > More > 3DEXPERIENCE Connection**. A login panel will appear.

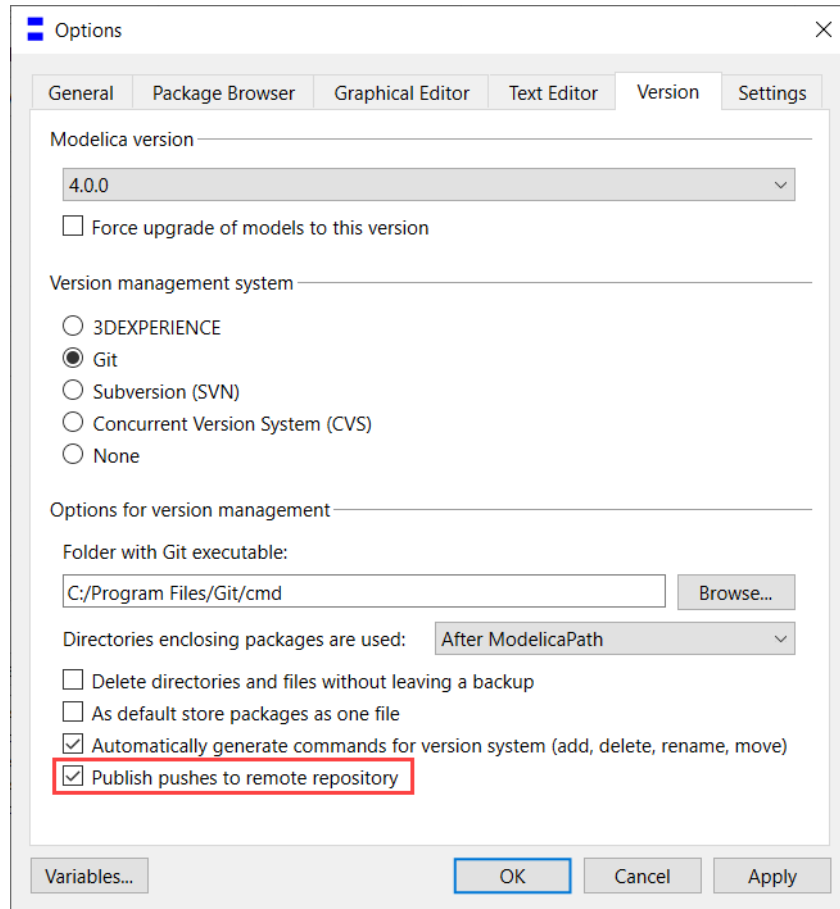
Note. When selecting the **3DEXPERIENCE** as versioning system in the **Version** tab, remember to deselect **As default store packages as one file**, if that setting is selected.

Important! Dymola must first be started once from the 3DEXPERIENCE Compass, in order to set the settings for connection to 3DEXPERIENCE.

3.6.2 Minor improvements

New option to publish pushes to remote directory

If you select to use Git as your versioning system, by the command **Tools > Options**, the **Version** tab, you have now a new option for Git, to publish pushes to remote directory:



The option is by default activated.

The use case is really rather the option to, in rare cases, e.g. for testing purposes, deselect the option to work only locally, without publishing. In all other cases, the option should be activated.

3.7 Other Simulation Environments

3.7.1 Dymola – Matlab interface

Compatibility

The Dymola – Simulink interface now supports Matlab releases from R2018a (ver. 9.4) up to R2023a (ver. 9.14). 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.

3.7.2 Real-time simulation

Compatibility – dSPACE

Dymola 2024x officially supports the DS1006, MicroLabBox, and SCALEXIO systems for HIL applications. For these systems, Dymola 2024x generated code has been verified for compatibility with the following combinations of dSPACE and Matlab releases:

- dSPACE Release 2018-A with Matlab R2018a
- dSPACE Release 2018-B with Matlab R2018b
- dSPACE Release 2019-A with Matlab R2019a
- dSPACE Release 2019-B with Matlab R2019b
- dSPACE Release 2020-A with Matlab R2020a
- dSPACE Release 2020-B with Matlab R2020b
- 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 R2022a and R2022b
- dSPACE Release 2023-A with Matlab R2022a, R2022b, and R2023a

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_rtmp_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_rtmp_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_usrtrcmmerge` 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 R2018a (Simulink Real-Time ver. 6.8) to R2023a (Simulink Real-Time ver. 8.2). Only Microsoft Visual C compilers have been tested.

3.7.3 Java, Python, and JavaScript Interface for Dymola

New or improved built-in functions available

A number of new and improved built-in functions are available in the interfaces.

For more information, see the corresponding sections in “Scripting” starting on page 16.

3.7.4 SSP Support

SSP export

SSP export has been modified for Modelica parameters and constants. They are now represented in the system structure description as **connectors**, with `kind="parameter"`, as specified in the SSP specification (previously they were a special kind of component element).

3.7.5 FMI Support in Dymola

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

Support for FMI 3.0

General

Dymola 2024x includes limited support for FMI version 3.0, with support for the new features described in the below sections. Note that all existing import and export features in Dymola are implemented in FMI 3.0.

Note!

The declaration order of alias variables in FMI 3.0 will not match the order of the original variables, due to a restriction in the specification. Thus, multibody animations of imported FMUs will not work.

For general information about FMI 3.0, see:

- [The FMI 3.0 specification](#)
- [A paper describing the new features of FMI 3.0](#)

These links are also available using **Tools > Help Documents**.

Terminals and icons

***Note!** This feature is in the beta-stage and performance and reliability will later be improved. As beta feature, it is not documented in the manual.*

FMI 3.0 introduces a new file to support terminals and icons. Dymola can use this to store input and output variables. You can export terminals and icons in FMUs by setting the flag `Advanced.Beta.FMI3.ExportTerminalsAndIcons = true`. (The flag is by default `false`.)

(This feature was already available in Dymola 2023x Refresh 1.)

Hybrid co-simulation

***Note!** This feature is in the beta-stage and performance and reliability will later be improved. As beta feature, it is not documented in the manual.*

An *exported* FMI 3.0 FMU supports early return for events.

An *imported* FMI 3.0 FMU supports hybrid co-simulation if the following is satisfied:

- The flag `Advanced.Beta.FMI3.HybridCoSim` is set to `true`. (The default value of the flag is `false`.)
- The imported FMU indicates that it supports:
 - Returning early for events
 - Getting and setting the states
 - Variable size communication step size.

The imported FMU has some additional hooks, since it is necessary to propagate the potential early return to other parts of the model (including other co-simulation FMUs).

(This feature was already available in Dymola 2023x Refresh 1.)

Co-simulation: Intermediate update

Dymola supports intermediate update for co-simulation FMUs. Setting the flag `Advanced.FMI.SetInputDerivatives=true` interpolate inputs for FMI 3 co-simulation, similarly as input derivatives for FMI 2. (The default value of the flag is `false`.)

Input interpolation is supported by FMI 3.0 FMUs generated by Dymola.

(This feature was already available in Dymola 2023x Refresh 1.)

Co-simulation: Early return

Dymola 2024x supports early return for an *exported* FMU, that is, to stop the execution of `fmi3DoStep` and return without reaching the predefined communication time.

Co-simulation: Event handling

Dymola 2024x supports event handling for an *exported* FMU.

Co-simulation: Variable step communication interval

See “Variable step co-simulation” below.

Support for FMI 1.0 will be discontinued in a future release

The support for FMI 1.0 will be discontinued in a future release of Dymola.

FMI import

Variable step co-simulation

Note! This feature is in the beta-stage and performance and reliability will later be improved. As beta feature, it is not documented in the manual.

There is a need for variable communication interval for co-simulation, to, for example, use small steps in the beginning of a process to handle transients, and then apply longer steps. In Dymola 2024x, you can change `dostepTrigger` to a variable that can be externally triggered. You do this by setting the flag:

```
Advanced.Beta.FMI.ExternalDoStepTrigger = true
```

(The flag is by default `false`.)

Note that the feature `canHandleVariableCommunicationStepSize` must be activated in the FMU that is imported to be able to activate variable step communication. In Dymola, this feature is by default activated in an exported FMU.

This feature is available for FMI 2.0 and FMI 3.0.

Naming of imported FMU

When you import an FMU, you can change the name of the Modelica model wrapping the imported FMU. You can use the command **File > Open > Import FMU...** to import an FMU:

fmi Import FMU

FMU file

Name of imported FMU's model

Preferred type
☒ Model exchange
☐ Co-simulation

Options
☐ Prompt before replacing an existing Modelica model
☐ Generate graphics for the diagram layer
☐ Translate value reference to variable name
☒ Structured declaration of variables

Insert in package

Variables to import
☒ All variables
☐ Black box (parameters, inputs, outputs)
☐ These variables:

In the above figure, the default name is **CoupledClutches_fmu**, like in previous Dymola versions. The user has added **My** in the beginning of the name.

Note that in previous Dymola versions, the default name of a “black box” import (that is, when you selected, in the **Variables to import** group, the alternative **Black box (parameters, inputs, outputs)** and imported) was **CoupledClutches_fmu_black_box**. That default is now changed to **CoupledClutches_fmu** also for this alternative.

For scripting, a new corresponding input argument has been added in the built-in function `importFMU`. See section “The built-in function `importFMU` improved” on page 17.

3.8 eFMI Support in Dymola

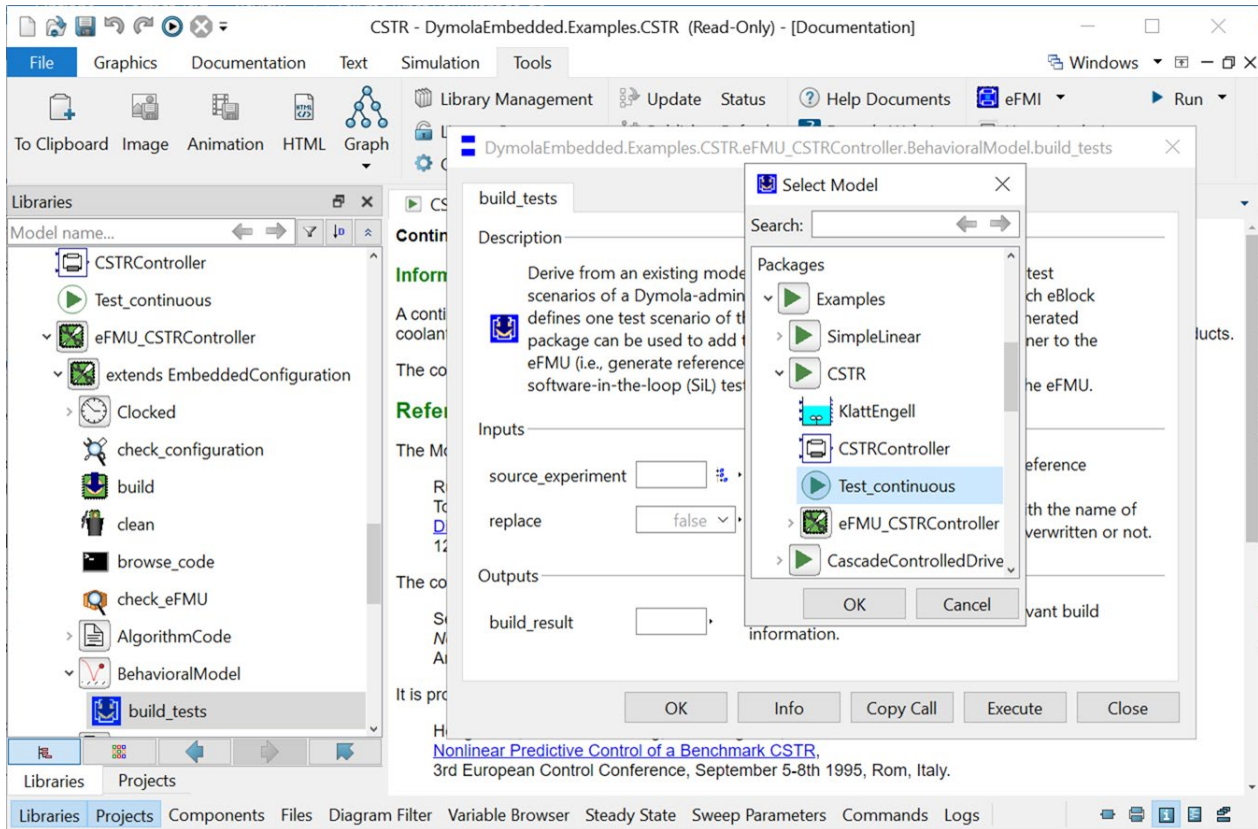
The new eFMI main features are: (1) support for eFMI Behavioral Model container generation; (2) further convenience functions for common build activities, (3) improved build activity feedback, (4) background build processes, (5) customized build environments and (6) error-safe default initialization. A user documentation is available in the “User’s guide” of the `DymolaEmbedded` library available from the “Tools” ribbon → “eFMI” button → “Load Libraries ...” entry.

Support for eFMI Behavioral Model container generation

Dymola now supports eFMI Behavioral Model container generation, including convenient means to generate experiment packages providing model-in-the-loop (MiL) and software-in-the-loop (SiL) tests of clocked system parts that are subject to eFMU code generation. Generated experiment packages ease the configuration of test scenarios and their tolerances for varying floating-point precisions and conducting respective tests of the production code generated by CATIA ESP. The actual testing facilities are based on the `Testing` library, leveraging on its convenient user-interface and logging. eFMI Behavioral Model containers can be completely automatically generated from the test scenarios modeled in experiment packages.

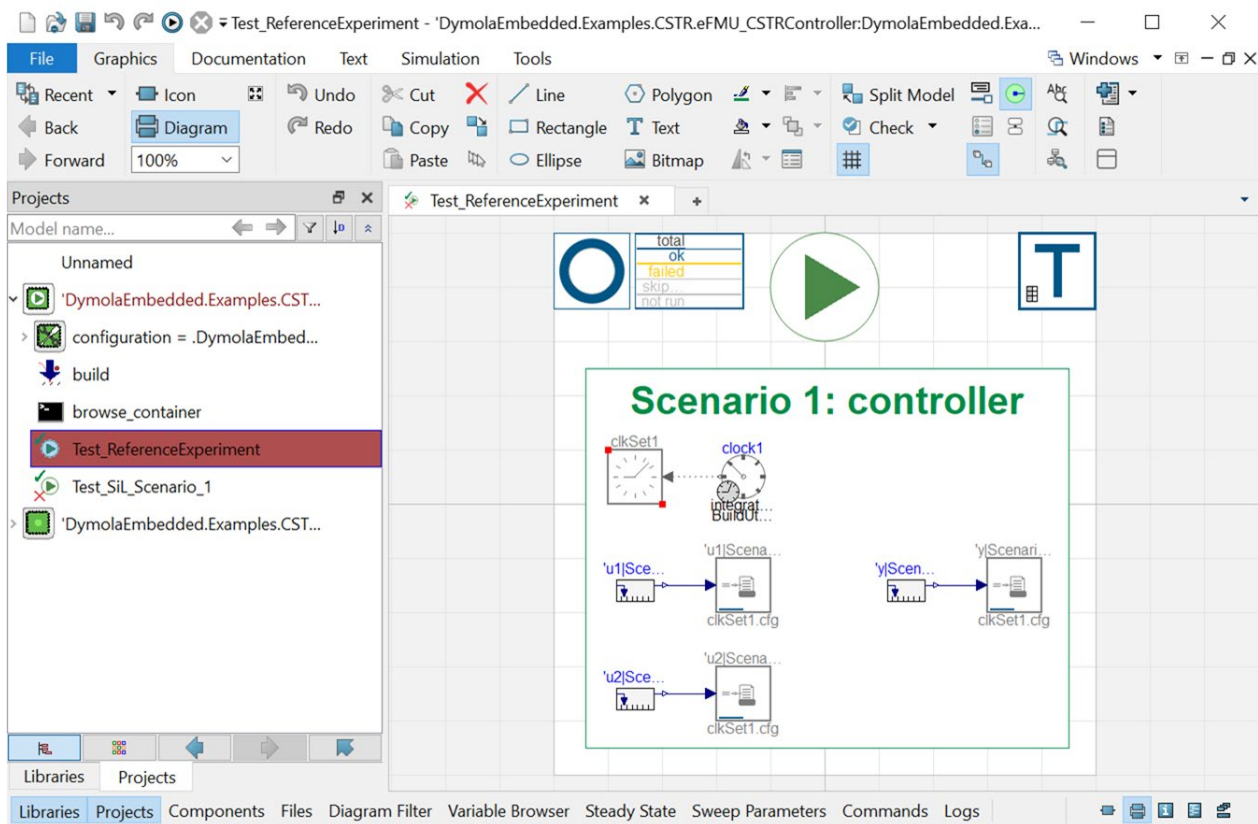
As an example, consider the continuous stirred-tank reactor (CSTR) of `DymolaEmbedded.Examples.CSTR.CSTRController`. To derive an experiment package from the closed loop continuous experiment `DymolaEmbedded.Examples.CSTR.Test_continuous`, generate a respective eFMI Behavioral Model container, and conduct software-in-the-loop tests of CATIA ESP generated Production Code containers, the following steps are required:

- Generate the eFMU with Algorithm Code and CATIA ESP Production Code containers by calling
`DymolaEmbedded.Examples.CSTR.eFMU_CSTRController.build()`,
with `load_binary_stub=true`.
- Call
`DymolaEmbedded.Examples.CSTR.eFMU_CSTRController.BehavioralModel.build_tests()` and pick the original closed loop experiment as `source_experiment` (Figure 1).



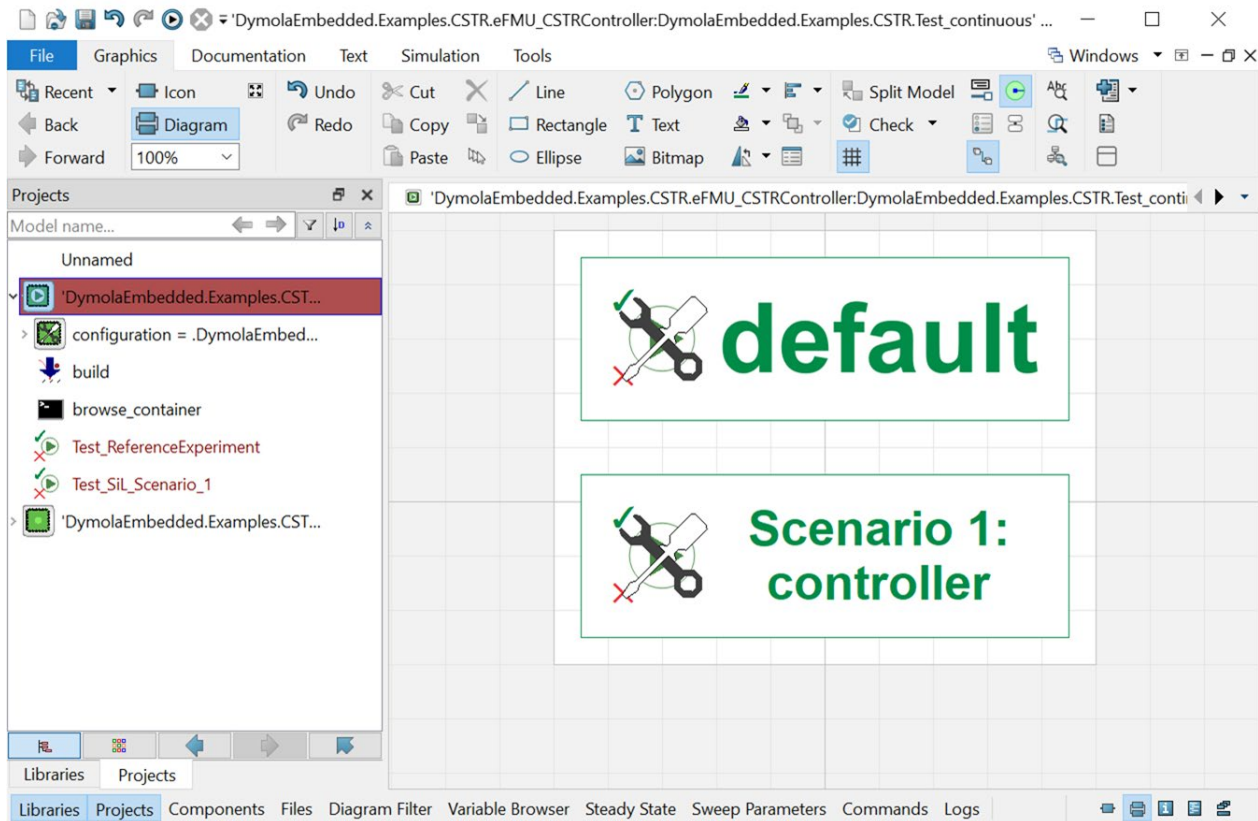
The `experiment` package `'DymolaEmbedded.Examples.CSTR.eFMU_CSTRController:DymolaEmbedded.Examples.CSTR.Test_continuous'` will be generated.

- Configure the sampling of the `controller` component (the controller model subject to eFMU code generation, i.e., the eBlock). This step is only required if the eBlock of the original experiment is not already sampled: Open the reference experiment `Test_ReferenceExperiment` of the generated experiment package in the “Graphics” ribbon’s “Diagram” layer, double click the `clkSet1` component and set `clockType` to `External` (Figure 2).



Doing so will introduce properly sampled reference trajectories with the `sample_period` of the eFMU generation configuration.

- Configure acceptable tolerances for the various floating-point precisions of available production codes: Open the generated experiment package (the actual root) in the “Graphics” ribbon’s “Diagram” layer, double click the `tolerances_default` component (labeled **default**) and set tolerances as follows (Figure 3):
`absolute_x32(y=21)`
`absolute_x64(y=21)`



- Generate the eFMU Behavioral Model container by calling the `build()` function of the generated experiment package. You can browse its content, in particular the manifest file `manifest.xml`, by calling the `browse_container()` function of the experiment package.
- Software-in-the-loop (SiL) test the generated CATIA ESP production code in the given scenario with the selected tolerances by simulating the `Test_SiL_Scenario_1` model of the experiment package. The test should pass; if the tolerances are set tighter, it will fail. Note, that the rather high tolerances are required because the original test scenario used to define reference results is continuous; whereas the SiL test is sampled (the reference trajectories stored are sampled, but the actual simulation of the eBlock, from which the trajectories are recorded, is purely continuous); and the sampling period of 5s is rather long for the output signal range of $[-750, -3000]$.

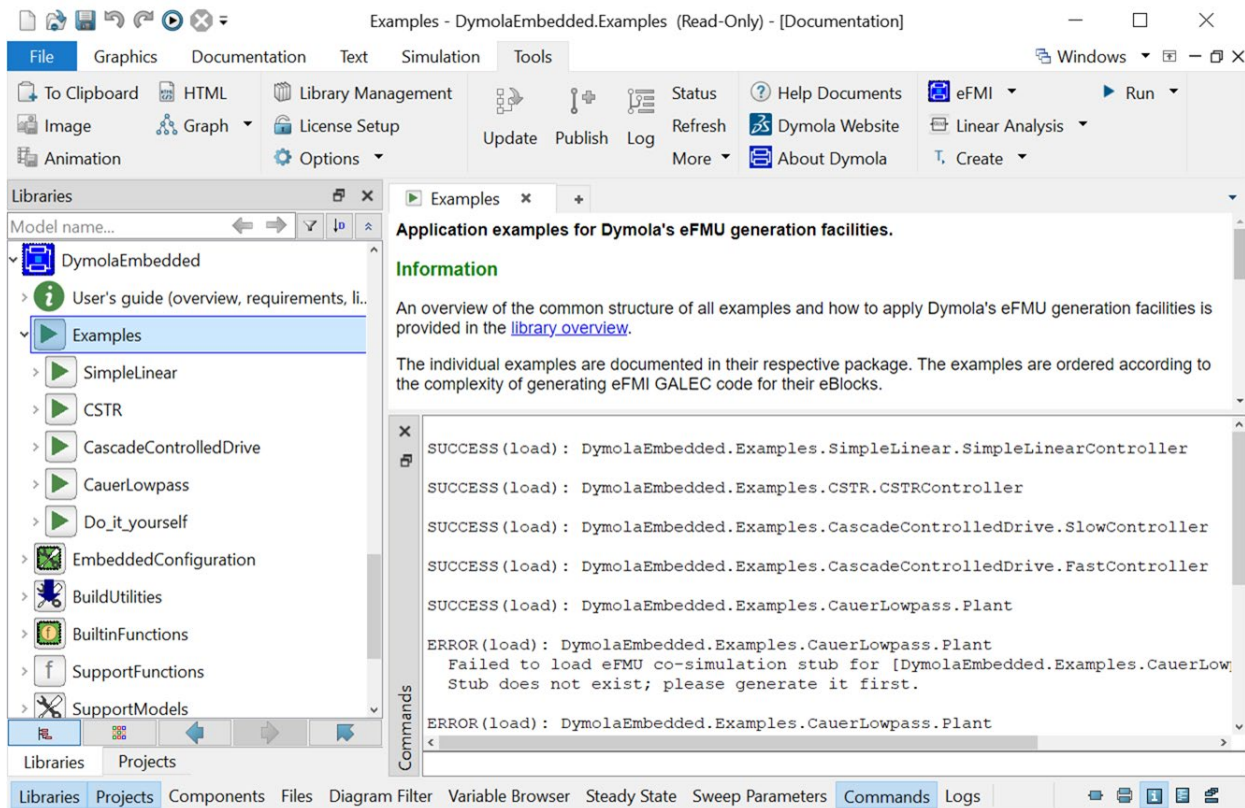
Further convenience functions for common build activities

- New function to cleanup a generated eFMU, i.e., delete everything (`.DymolaEmbedded.EmbeddedConfiguration.clean()`).

- New function to cleanup all generated eFMUs of a package hierarchy (`.DymolaEmbedded.BuildUtilities.clean_all()`). Likewise `build_all()`, the function is also available from the “eFMI” button in the “Tools” ribbon.
- Generated eFMU co-simulation stubs now provide the new `resolve_code_configuration()` function, which can be used to retrieve the original CATIA ESP configuration used to build one of the stub's production codes.
- Generated eFMU co-simulation stubs now provide the new `resolve_interface()` function, which can be used to retrieve the stub's GALEC/eBlock interface.

Improved build activity feedback

Much improved feedback and error messages in the Commands window on build activities like building Production Code containers, cleanup of eFMUs, loading eFMU co-simulation stubs etc. Figure 4 shows a typical output in the “Commands” window for “Load eFMI Co-simulation Stubs...” of the “eFMI” button in the “Tools” ribbon.



Background build processes

All build actions are now run as background processes without annoying pop-up terminal windows; they can be aborted from within Dymola (“Simulation” ribbon → “Stop” button).

Customized build environments

Build scripts can now use a customized build environment instead of the default (e.g., to pick a certain Microsoft Visual C++ compiler version, Cppcheck version, Java runtime environment etc.). To do so, the command line environment variable `DYMOLA_eFMI_BUILD_ENVIRONMENT` has to be set to the full-qualified path of the customization script. The respective *.bat script is executed before each build action (like generating or checking production code, compiling binary code etc.) and can therefore set environment variables like compiler paths etc. to bypass the default resolution for such.

For example, to pick a user-defined Microsoft Visual C++ 12 compiler for building production code with CATIA ESP, a respective `my-setup.bat` script could be

```
set "PATH= "C:\my\path;C:\Program Files (x86)\Microsoft Visual
Studio 12.0\VC;%PATH%"
if defined tgtInstrSetArch (
    call vcvarsall.bat %tgtInstrSetArch%
)
exit /b 0
```

whereas `tgtInstrSetArch` will be defined by the `DymolaEmbedded` library when building production code with CATIA ESP. To use above script when building eFMUs, set the environment variable `DYMOLA_eFMI_BUILD_ENVIRONMENT` to the full qualified path of `my-setup.bat` before starting Dymola, e.g., by starting a new Windows Command line (cmd), typing

```
set "DYMOLA_eFMI_BUILD_ENVIRONMENT=C:\my\path\my-setup.bat"
```

and starting Dymola from that very cmd.

Error-safe default initialization

Default initialization values are now also correctly computed – and listed in Algorithm Code manifests – if any not-a-number (NaN) values or error signals are encountered throughout initialization.

3.9 New libraries

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

3.9.1 Process Modeling Library

The Process Modeling Library is a commercial and licensed Modelica library for the Dymola modeling and simulation environment.

This library provides models to simulate thermal separation processes based on multiphase multicomponent equilibria. Such models are mostly separations processes, like rectification, extraction or gravitational separation and related systems.

The library uses the Multiflash software for the calculation of physical properties.

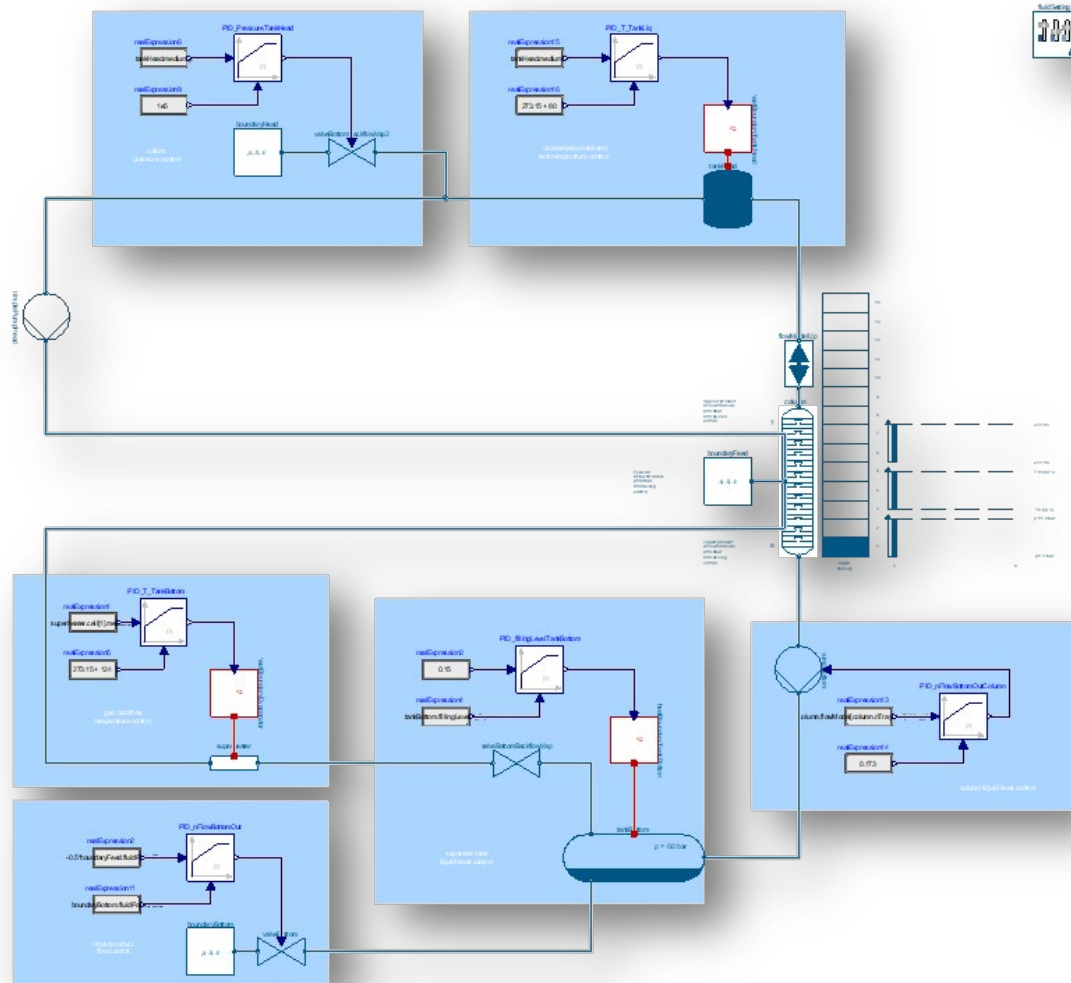
All models are thread safe and it is recommended to use parallelization. This is obtained by setting the flag

```
Advanced.Translation.ParallelizeCode = true;
```

The library requires a working license, the installation of the Multiflash Media Library (the product “Systems Thermodynamics Connector”) of Dassault Systemes, and the installation of the Multiflash software package. (Note that the Multiflash software package is not available from Dassault Systemes.)

Note that neither Process Modeling Library nor Multiflash Media Library are currently supported for Linux.

An example of a model:



3.10 Modelica Standard Library and Modelica Language Specification

The current version of the Modelica Standard Library is version 4.0.0. The current version of the Modelica Language Specification is 3.6.

3.11 Documentation

General

In the software, distribution of Dymola 2024x Dymola User Manuals of version “September 2023” will be present; these manuals include all relevant features/improvements of Dymola 2024 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 2024x 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 2024x is supported, as 64-bit application, on Windows 10 and 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 2024x 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 2012 Express Edition (11.0)
- Visual Studio 2015 Express Edition for Windows Desktop (14.0)
- Visual Studio 2017 Desktop Express (15) **Note!** This compiler only supports compiling to Windows 32-bit executables.
- Visual Studio 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 www.Dymola.com/compiler.
- Visual Studio 2019 Community (16). Note that you can select to use Clang 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 www.Dymola.com/compiler.
 - 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 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 2019 alternative: **Visual Studio 2022/Visual C++ 2022 (17).**

- For more information about installing and testing this compiler with Dymola, see www.Dymola.com/compiler.
- 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 2012 (11.0)
- 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 code generator for this compiler.
- Visual Studio Enterprise 2019 (16). Note that you can select to use Clang code generator for this compiler.
- Visual Studio Enterprise 2022 (17). Note that you can select to use Clang code generator for this compiler.
- Visual Studio Professional 2022 (17) Note that you can select to use Clang code generator for this compiler.

Clang compiler

If you first select to use Visual Studio 2019 or Visual Studio 2022 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 2024x has limited support for the MinGW GCC compiler. The following versions have been tested and are supported:

- For 32-bit GCC: version 6.3 and 8.2
- For 64-bit GCC: version 7.3 and 8.1

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

To download any of these free compilers, please visit <http://www.Dymola.com/compiler> 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 GCC compiler. Usually, you can select this as an add-on when installing GCC.

Current limitations with 32-bit and 64-bit 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 option² enabled.
- For 32-bit simulation, parallelization (multi-core) is currently not supported for any of the following algorithms: RadauIIa, Esdirk23a, Esdirk34a, Esdirk45a, and Sdirk34hw.
- Compilation may run out of memory also for models that compile with Visual Studio. The situation is better for 64-bit GCC than for 32-bit GCC.

In general, 64-bit compilation is recommended for MinGW GCC. In addition to the limitations above, it tends to be more numerically robust.

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.
- 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.
 - **Note** however that if you want to exchange FMUs with the Systems Simulation Design app (sometimes referred as “SID”), you should use Ubuntu 18.04 instead.
- 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 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 11.16.2.1. This version is part of the Dymola distribution.)

As an alternative to FLEXnet, Dassault Systèmes License Server (DSLS) can be used. Dymola 2024x supports DSLS R2024x. 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.

² Having the code export options means having any of the license features **Dymola Binary Model Export** or the **Dymola Source Code Generation**.

Linux

Supported Linux versions and compilers

Dymola 2024x runs on Red Hat Enterprise Linux 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 2024x it is Qt 6.5.1.
- 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.

You can use a dialog to select compiler, set 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.

You can however still change the compiler by changing the variable CC in /opt/dymola-<version>-x86-64/bin/dsbuild.sh. As an example, for a Dymola 2024x application:

```
/opt/dymola-2024x-x86_64/bin/dsbuild.sh
```

Dymola 2024x is supported as a 64-bit application on Linux.

Notes

- 32-bit compilation for simulation might require explicit installation of 32-bit libc. E.g. on Ubuntu: `sudo apt-get install g++-multilib libc6-dev-i386`
- Dymola is built with Qt 6.5.1 and thereby inherits the system requirements from Qt. This means:
 - Since Qt 6.5.1 no longer supports embedding of the XCB libraries, these must now be present on the platform running Dymola. 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.
 - The library `libxcb-xinput.so.0` might require explicit installation.
- For FMU export/import to work, zip/unzip must be installed.

Note on libraries

- The library UserInteraction is not supported on Linux.

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 11.16.2.1.)

As an alternative to FLEXnet, Dassault Systèmes License Server (DSLS) can be used. Dymola 2024x supports DSLS R2024x. 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.

"

'