Wolfram *SystemModeler*™

User Guide

WOLFRAM

Vertrieb durch:
ADDITIVE GmbH • Max-Planck-Straße 22b • 61381 Friedrichsdorf
http://additive-mathematica.de/ • eShop: http://eshop.additive-net.de
Verkauf: +49-6172-5905-134 mathematica@additive-net.de
Support: +49-6172-5905-20   support@additive-net.de

ADDITIVE
SOFT- & HARDWARE FÜR TECHNIK & WISSENSCHAFT

Contents

Chapter 1: **Installation and Setup**

1.1 Supported Platforms

Wolfram *SystemModeler*™ is available on the platforms listed below.

- Windows
- Mac OS X

Note that there are some minor differences between the Windows version, described in this manual, and the Mac OS X version.

- *Keyboard shortcuts*
  In general the **Ctrl** key in keyboard shortcuts is mapped to the **Command** key, also known as the **Apple** key, on Macintosh keyboards. The only exception is the **Ctrl**+**Tab** and **Ctrl**+**Shift**+**Tab** shortcuts, which are the same on both Windows and Mac OS X.
- *Menus*
  The locations of the **Options** and **About SystemModeler** menu items follow the guidelines for Mac OS X and are found in the **SystemModeler** menu. The **Options** menu item is called **Preferences** on Mac OS X.

1.2 Installing *SystemModeler*

This section contains information on how to install *SystemModeler* on the supported platforms as well as any prerequisites.

1.2.1 Windows

Run the executable file for *SystemModeler* you received. This will start the *SystemModeler* installation wizard. Follow the instructions in the wizard to complete the installation.

1.2.2 Mac OS X

SystemModeler requires Apple's Xcode to be installed. Xcode is available at the Mac App Store or on your Mac OS X installation media.

Mount the disk image you received for *SystemModeler*. To install *SystemModeler*, drag and drop the *SystemModeler* icon to the Application folder. This completes the installation of *SystemModeler*.

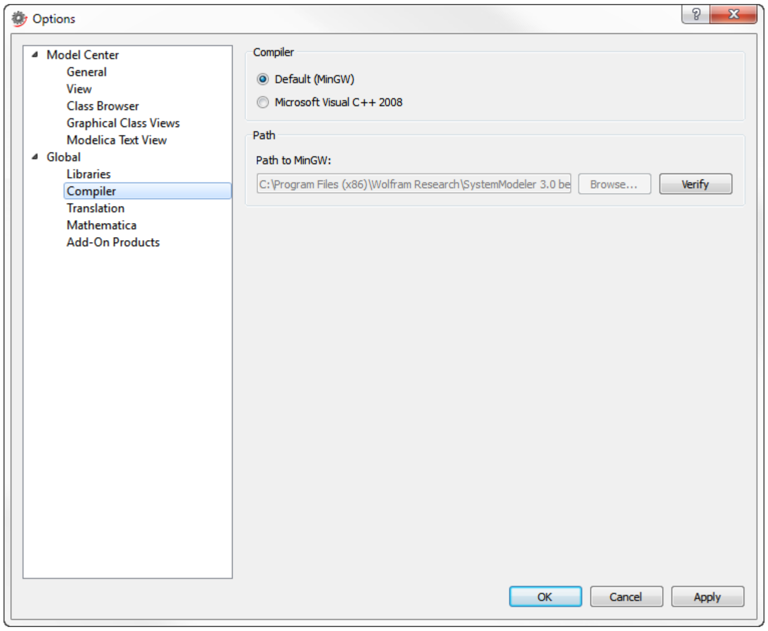
1.3 C/C++ Compiler Settings

To use *SystemModeler*, a C/C++ compiler is needed. *SystemModeler* supports the following compilers:

- MinGW on Windows
- Microsoft Visual C++ 2008 (including the free Express edition) on Windows
- GCC 4.0 on Mac OS X

On Windows, *SystemModeler* is shipped with the MinGW compiler, and that compiler is used by default. On Mac OS X, it is assumed that GCC is installed in /usr/bin.

To change the compiler settings, open the **Options** dialog box by choosing **Tools ▸ Options**. The compiler settings are available in the **Compiler** view. Select the desired compiler. When the selected compiler is Microsoft Visual C++, specify the location of vsvars32.bat. When the selected compiler is GCC, specify the location of g++. It is possible to verify that the new compiler settings are working by clicking on the **Verify** button. This will test the compiler settings by building a small test model.



Compiler

[✓ Default (MinGW)] Microsoft Visual C++ 2008

Path

Path to MinGW: C:\Program Files (x86)\Wolfram Research\SystemModeler 3.0 be

Figure 1-1: The **Compiler** view in the **Options** dialog box.

1.4 Wolfram *SystemModeler Link*

The Wolfram *SystemModeler Link* (*WSMLink*) is a *Mathematica* package that links to *SystemModeler*. If you have *Mathematica* installed, the package will automatically be installed the first time you start *SystemModeler*. If the automatic configuration does not succeed, a dialog is shown for manual configuration. If you do not have *Mathematica,* you can safely click **Cancel** in this dialog.

To manually configure the link, go to **Tools ▸ Options** in the *SystemModeler* menu. Then select *Mathematica* on the left side, under **Global**. Click **Configure SystemModeler Link**. For the configuration to succeed, *Mathematica* cannot be running, so be sure to close it when configuring. Follow the dialogs to complete the configuration.

Chapter 2: **Introduction**

This chapter gives a short introduction to *SystemModeler.* You will learn how to use *Model Center* to build your own model with components from the Modelica Standard Library as well as how to simulate the model and plot variables using *Simulation Center*. At the end of this chapter we will also introduce the *Mathematica* notebook environment and show how *SystemModeler* integrates into *Mathematica*.

The Getting Started document found in the **Help** menu of *Model Center* and *Simulation Center* contains several examples that are useful for getting familiar with modeling in *SystemModeler.*

2.1 How to Start *SystemModeler*

SystemModeler consists of a modeling environment, *Model Center*; a simulation environment, *Simulation Center*; and the Wolfram *SystemModeler Link* package for *Mathematica* that connects to *SystemModeler*.

2.1.1 How to Start the *Model Center*

The natural starting point of *SystemModeler* is the *Model Center*. Choose **Wolfram SystemModeler ▸ Wolfram SystemModeler** from the **Start** menu in Windows. A splash screen will appear while the model editor is loading.

When the *Model Center* is started for the first time, *SystemModeler* tries to configure *SystemModeler Link* for *Mathematica*. If it does not succeed, a dialog for configuring the link is shown. If you do not have *Mathematica,* you can safely skip this step by clicking the **Cancel** button. If you would like to configure the link, see Section 1.4 on page 3 for details.

The menus, tool buttons, and windows available in *Model Center* depend on the installed version. *Model Center* starts with a default configuration, but allows for customization.

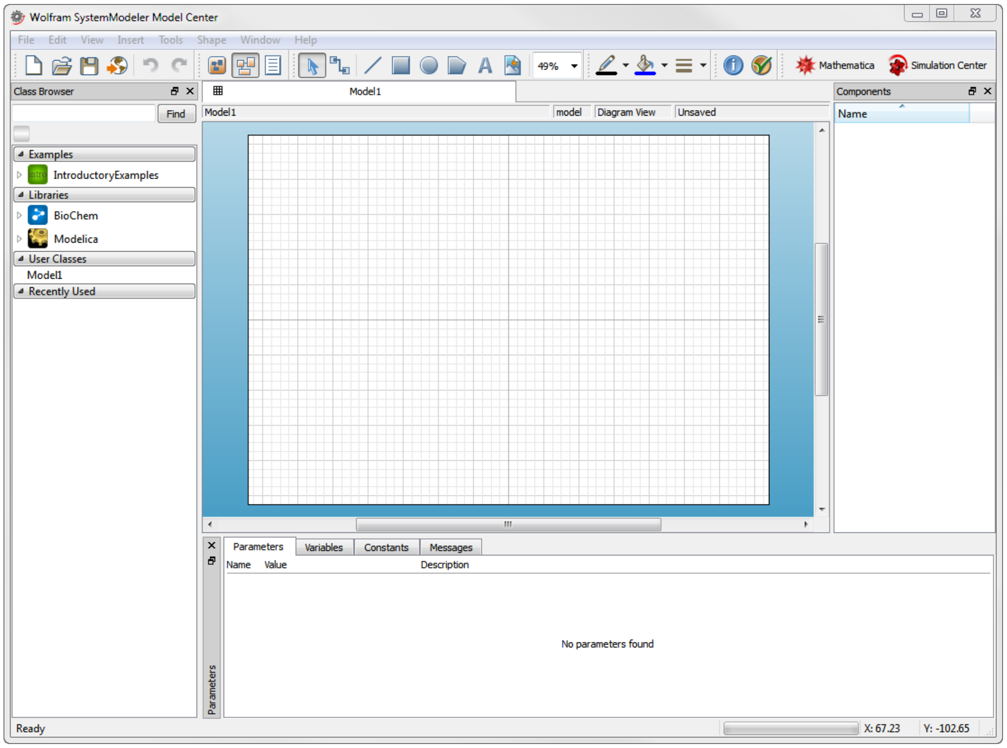


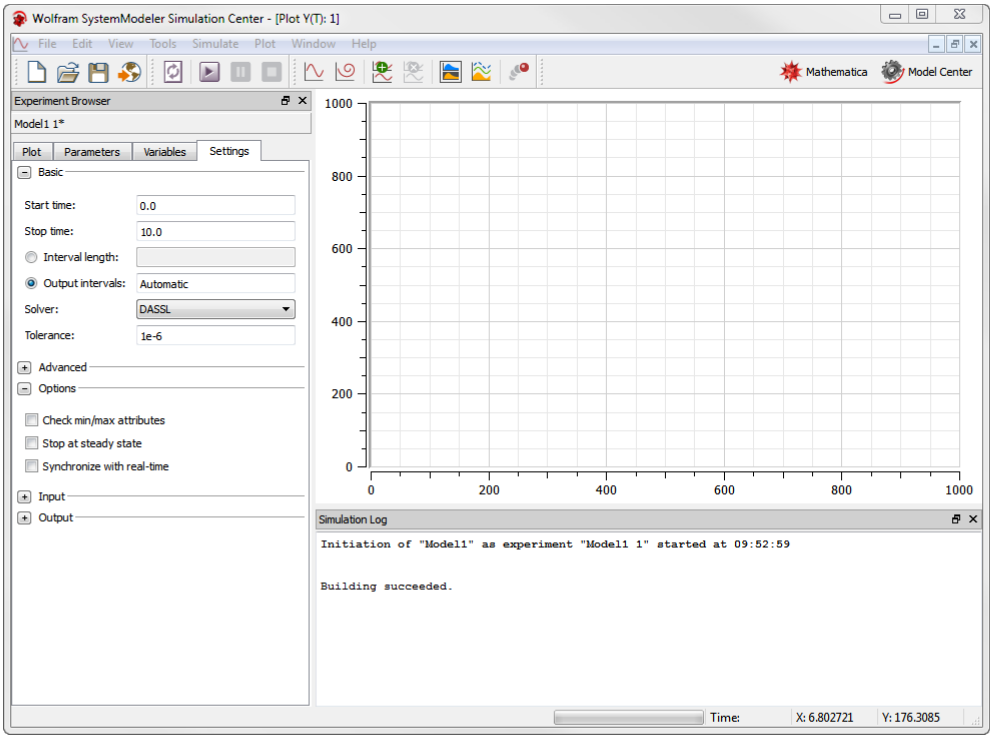
Figure 2-1: The **Model Center** window.

For a detailed description of the *Model Center*, see Chapter 3.

2.1.2 How to Start *Simulation Center*

To start the *Simulation Center,* click the **Simulation Center** button in the top-right corner in the *Model Center*. While *Simulation Center* is loading, a splash screen is shown.

As for the *Model Center*, the menus, tool buttons, and windows available in the *Simulation Center* depend on the installed version.


Wolfram SystemModeler Simulation Center - [Plot Y(T): 1]
File Edit View Tools Simulate Plot Window Help
Mathematica
Model Center
Experiment Browser
Model1 1*
Plot Parameters Variables Settings
Basic
Advanced
Options
Input
Output
Simulation Log
Time:
X: 6.802721
Y: 176.3085

Start time: 0.0
Stop time: 10.0
Interval length:
Output intervals: Automatic
Solver: DASSL
Tolerance: 1e-6
Check min/max attributes
Stop at steady state
Synchronize with real-time

```
Initiation of "Model1" as experiment "Model1 1" started at 09:52:59

Building succeeded.
```

Figure 2-2: The **Simulation Center** window.

See Chapter 4 for a more detailed description on how to use the *Simulation Center*.

2.1.3 How to Start *SystemModeler Link*

For owners of *Mathematica*, it is also possible to load *SystemModeler* into the *Mathematica* environment. Before being able to use *SystemModeler* from within *Mathematica*, the *SystemModeler Link* must be configured. In most cases, the configuration happens automatically when first starting *SystemModeler*. See Section 1.4 on page 3 for more information. Once configured, *Mathematica* can be started by clicking the **Mathematica** button on the top-right of the *SystemModeler* window. This will open the guide page for *SystemModeler Link*. From here, all documentation pages for the link are reachable. To get started, click the link to the Getting Started tutorial on this guide page.

2.2 The Modelica Standard Library

SystemModeler is delivered with a subset of the Modelica Standard Library, a brief overview of which is given here. This library is loaded when the *Model Center* is started and can be browsed using the **Class Browser**, visible on the left-hand side of the *Model Center*.

The **Class Browser** uses a tree view to visualize the contents of a library. Packages are represented as branches of the tree and components as leaves. To show the contents of a package, for instance the Modelica package, expand it by clicking the symbol to the left of its icon. As seen in the figure below, the Modelica Standard Library is hierarchically organized with several levels of subpackages.

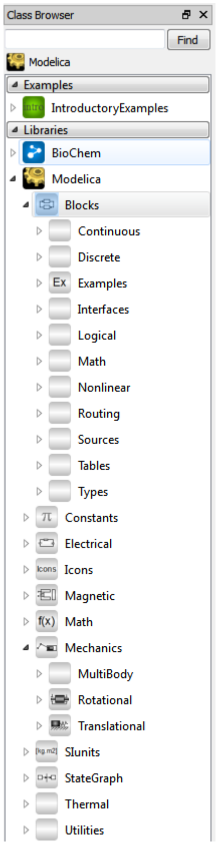


Figure 2-3: Browsing the Modelica Standard Library in the **Class Browser**.

Clicking the symbol to the left of an expanded package will show it as collapsed again, hiding its contents.

To view the contents of a package as a separate group within the **Class Browser**, right-click its name and choose **View as Group**. This can prove useful if you use components from more than one package when designing a model. By opening the packages of interest as separate groups, you can minimize the scrolling up and down needed in order to find the components.

Both components and packages can be opened for editing by right-clicking the name and choosing **Open** from the menu. Components can also be opened for editing by double-clicking their names.

A short presentation of the main packages of the Modelica Standard Library is given below.

- `Blocks`. Contains input/output blocks to build block diagrams.
- `Constants`. Provides frequently needed constants from mathematics, machine-dependent constants, and constants from nature.
- `Electrical`. Includes electrical components to build up analog and digital circuits, as well as machines to model electrical motors and generators.
- `Icons`. Provides the graphical layout for many component icons.
- `Magnetic`. Contains magnetic components to build electromagnetic devices.
- `Math`. Contains basic mathematical functions, as well as functions operating on vectors and matrices.
- `Mechanics`. Contains components to model the movement of 1D rotational, 1D translational, and 3D mechanical systems.
- `SIunits`. Contains type definitions with SI standard names and units.
- `StateGraph`. Provides components to model discrete event and reactive systems in a convenient way.
- `Thermal`. Contains libraries to model heat transfer and fluid heat flow.
- `Utilities`. Contains Modelica functions that are especially suited for scripting.

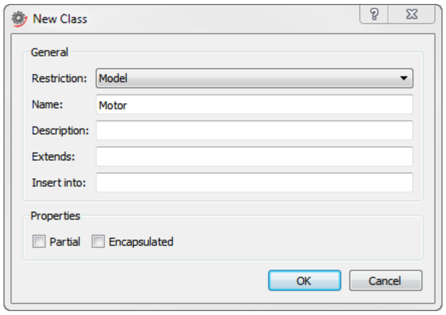
More detailed information about the packages and components can be found in the integrated documentation. Right-click on the name of a package or component in the **Class Browser** and choose **View Documentation** from the popup menu.

2.3 Your First *SystemModeler* Model

We will now introduce the *Model Center* by showing how to build a model of a simple DC motor. Because the DC motor includes both electrical and rotational mechanical components, the example also illustrates multidomain modeling.

2.3.1 Creating a New Model

To create a new model, choose **New Class** from the **File** menu. A dialog box will appear, in which you will be able to specify a name for the new model, among other things. Enter `Motor` as the model name. In most dialog boxes, you can get help with any item in the dialog box by clicking the question mark icon in the title bar and then clicking the item.



New Class

General

Restriction: Model

Name: Motor

Description:

Extends:

Insert into:

Properties

☐ Partial ☐ Encapsulated

Figure 2-4: Creating a DC motor model.

When clicking the **OK** button, the class window of the `Motor` model will automatically open and become the active class window. The class window presents different views of a model. A model consists of two graphical views (icon and diagram), and one text view. The **Diagram View** is the default view for models and is therefore the active view of the new class window.

It is possible to have several class windows open at the same time. The names of all open classes are visible in a row of tabs below the toolbar in the main window. By clicking the name of a class, its class window will become the new active class window. The **Ctrl** + **Tab** and **Ctrl** + **Shift** + **Tab** combination of keys can also be used to switch between open class windows.

Your new `Motor` model will also appear in the **Class Browser** once created. As it was created at top level and not within a package, it will appear as a leaf under **User Classes**.

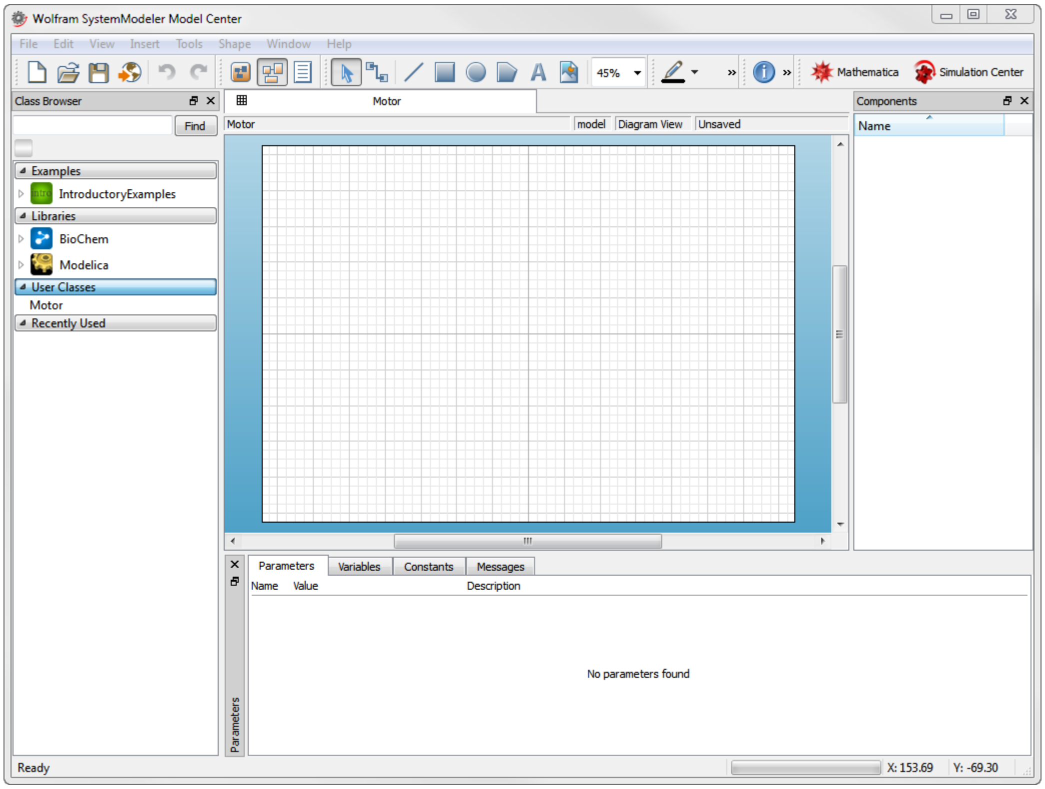


Figure 2-5: The class window for the DC motor model.

2.3.2 Saving the Model

To save the model, choose **File ▸ Save**. As the model is saved for the first time, you will be asked to specify a file name. Give the file the same name as your model (`Motor`).

2.3.3 Adding Components

It is possible to assemble the DC motor by drag-and-drop of components from the **Class Browser** to the **Diagram View**. In this example, we need a constant voltage source and a rotational mass representing the motor shaft, as well as a resistor, inductor, and electromagnetic force (EMF). All these components are provided in the Modelica Standard Library and are therefore easily accessible in the **Class Browser**.

In Version 3.1 of the standard library, the components are located in the following packages:

- `Modelica.Electrical.Analog.Sources` (`ConstantVoltage`)
- `Modelica.Electrical.Analog.Basic` (`EMF`, `Ground`, `Inductor`, `Resistor`)
- `Modelica.Mechanics.Rotational.Components` (`Inertia`)

An alternative to browsing the packages in the **Class Browser** to locate components is to search for the components. Type in the name of the component in the text box at the top of the **Class Browser** and click the **Find** button. This is especially useful if you do not know the exact location of the component in the package hierarchy. The result is presented at a separate group at the top part of the **Class Browser**.

To place a component in the **Diagram View** of your `Motor` model, drag the component from the **Class Browser** to the **Diagram View** of the class window.

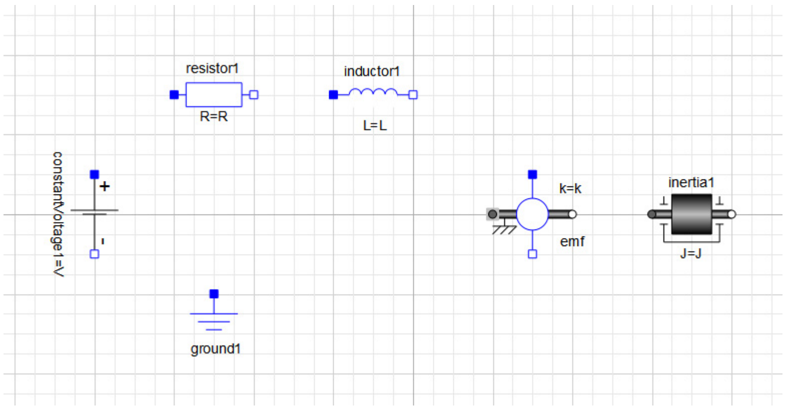


Figure 2-6: Adding components to the DC motor model.

Components placed in the **Diagram View** can be graphically transformed using the mouse and keyboard. To move a component, select it and hold down the left mouse button while moving the mouse. If more than one component is selected, all selected objects will be moved simultaneously.

Scaling of components is done using the green selection handles that are visible when a component is selected. Place the mouse cursor over one of the handles, then click and hold down the left mouse button while moving the mouse.

Components can also be rotated freely using the selection handles. Place the mouse cursor over one of the handles, then click and hold down the left mouse button as well as the **Shift** button on the keyboard while moving the mouse.

Pressing the right mouse button when the mouse cursor is placed over a component brings up a popup menu with applicable operations.

2.3.4 Connecting Components

When the components have all been placed in the **Diagram View**, similar to Figure 2-6 above, the next step is to connect the components. Components are connected using the **Connection Line Tool** from the **Graphic Tools** toolbar.

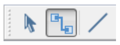


Figure 2-7: The **Connection Line Tool** on the **Graphic Tools** toolbar.

To connect two components, select the **Connection Line Tool** and place the mouse cursor over a connector (usually a square or circular symbol on the side of the components). When the cursor is close enough, it will change in appearance. Click and hold down the left mouse button, drag the cursor to the other connector, and release the mouse button when the mouse cursor changes. The connection line will be drawn as a right-angle connection line.

Continue to connect all components until the model diagram resembles the one in Figure 2-8 below.

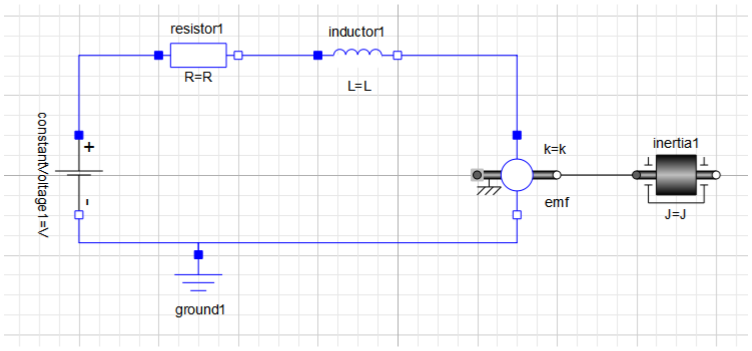
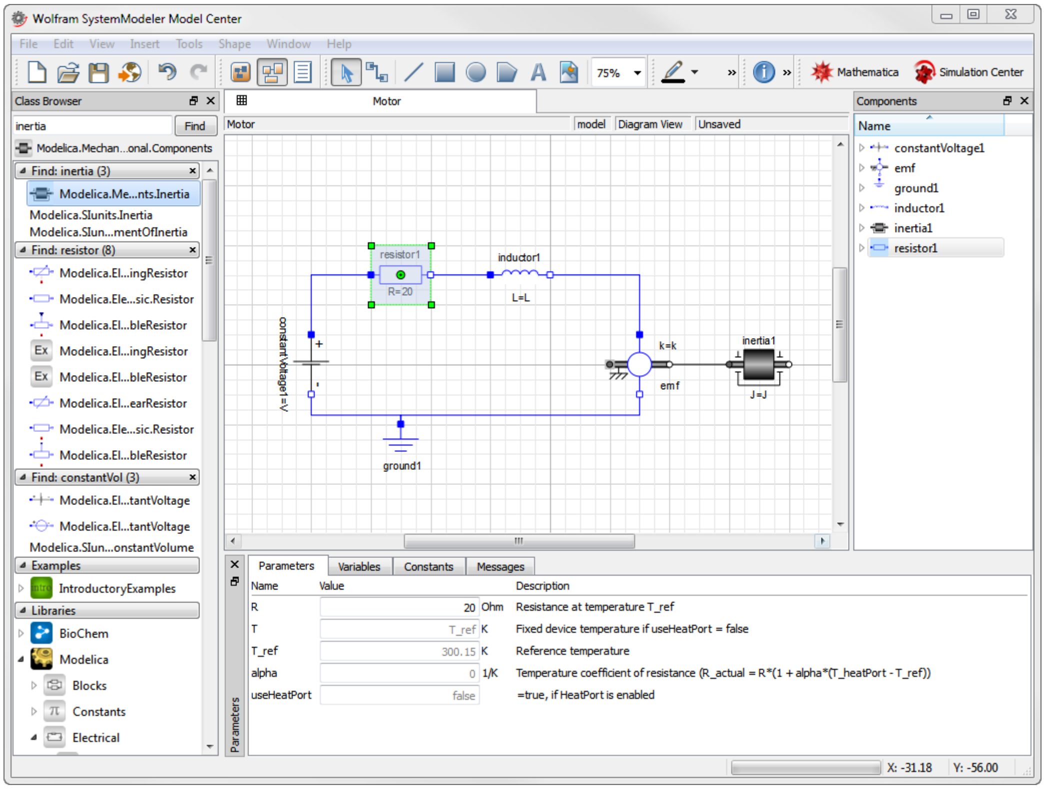


Figure 2-8: Connecting components in the DC motor model.

2.3.5 Changing Parameter Values of Components

To change a parameter value of a component, for example the resistance of the resistor, select the resistor in the **Diagram View** by clicking on it. The parameters of the resistor component will be listed in the **Parameters** view of the window below the class window. Set the value of the parameter `R` to `20` and press the **Return**, **Enter**, or **Tab** key.


Wolfram SystemModeler Model Center
File Edit View Insert Tools Shape Window Help
75%
Mathematica
Simulation Center
Class Browser
inertia
Find
Modelica.Mechan...onal.Components
Find: inertia (3)
Modelica.Me...nts.Inertia
Modelica.SIunits.Inertia
Modelica.SIun...mentOfInertia
Find: resistor (8)
Modelica.El...ingResistor
Modelica.Ele...sic.Resistor
Modelica.El...bleResistor
Modelica.El...ingResistor
Modelica.El...bleResistor
Modelica.El...earResistor
Modelica.Ele...sic.Resistor
Modelica.El...bleResistor
Find: constantVol (3)
Modelica.El...tantVoltage
Modelica.El...tantVoltage
Modelica.SIun...onstantVolume
Examples
IntroductoryExamples
Libraries
BioChem
Modelica
Blocks
Constants
Electrical
Motor
Motor
model
Diagram View
Unsaved
resistor1
R=20
inductor1
L=L
constantVoltage1=V
k=k
emf
inertia1
J=J
ground1
Components
Name
constantVoltage1
emf
ground1
inductor1
inertia1
resistor1
Parameters
Variables
Constants
Messages
X: -31.18
Y: -56.00

Name	Value		Description
R	20	Ohm	Resistance at temperature T_ref
T	T_ref	K	Fixed device temperature if useHeatPort = false
T_ref	300.15	K	Reference temperature
alpha	0	1/K	Temperature coefficient of resistance (R_actual = R*(1 + alpha*(T_heatPort - T_ref))
useHeatPort	false		=true, if HeatPort is enabled

Figure 2-9: Setting the resistance of the resistor component in the DC motor model.

When you have done this, study the icon of the resistor component. It should show `R=20` instead of `R=R`. The resistance of the resistor is now 20 ohm. Most parameters of components in the standard library have a default value that is used if you do not specify a value explicitly.

2.3.6 Simulating the Model

You have now created your first model in *SystemModeler* and you are ready to simulate it. Open the *Simulation Center* by clicking the **Simulation Center** button on the **Applications** toolbar.

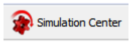
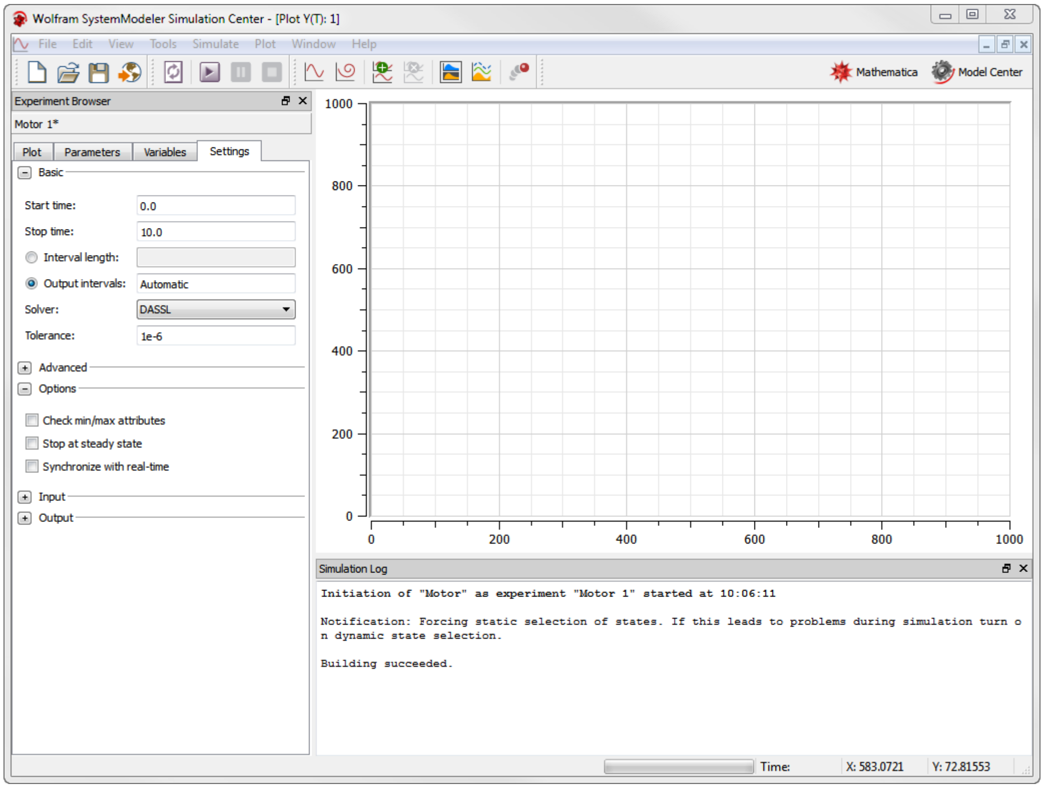


Figure 2-10: The **Simulation Center** button on the **Applications** toolbar.

When the *Simulation Center* is started, the active model in the model editor is converted into an experiment in which you can set parameter values, initial values for state variables, and simulation settings. When configured, the model can be simulated.



Start time: 0.0
Stop time: 10.0
Interval length:
Output intervals: Automatic
Solver: DASSL
Tolerance: 1e-6

- [ ] Check min/max attributes
- [ ] Stop at steady state
- [ ] Synchronize with real-time

```
Initiation of "Motor" as experiment "Motor 1" started at 10:06:11

Notification: Forcing static selection of states. If this leads to problems during simulation turn o
n dynamic state selection.

Building succeeded.
```

Figure 2-11: The **Simulation Center** window.

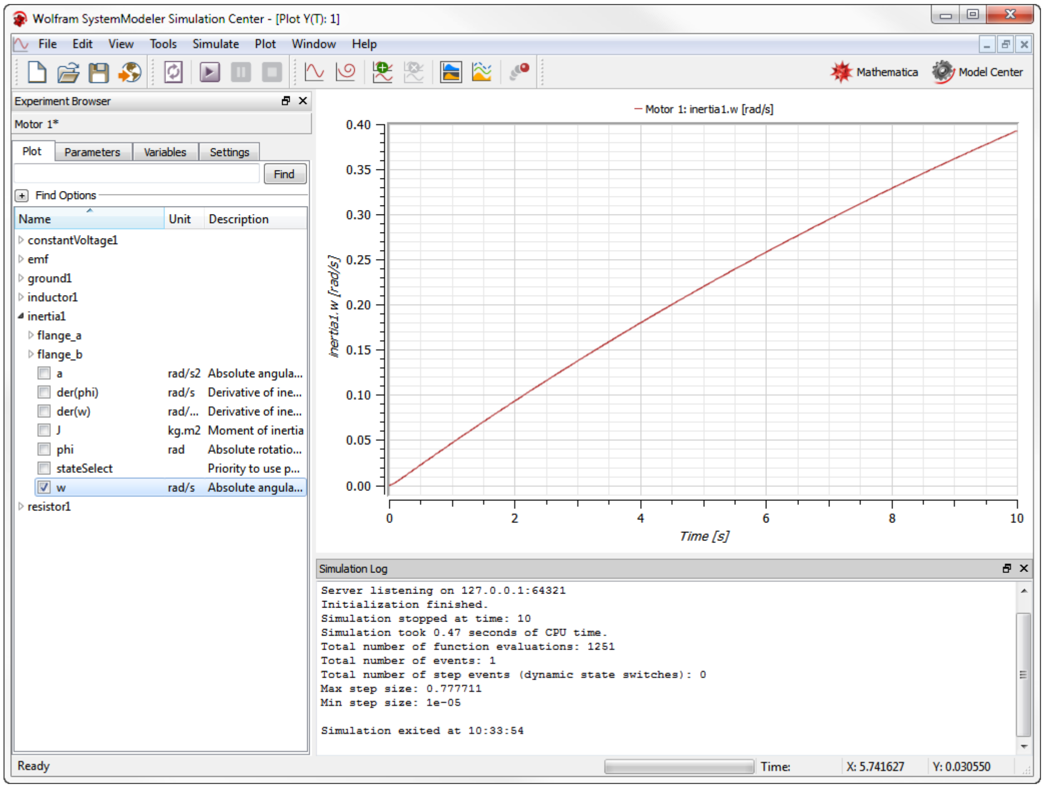
To perform a simulation, choose **Simulate** from the **Simulate** menu, or click the button on the toolbar shown in Figure 2-12 below.

Figure 2-12: The **Simulate** button on the toolbar.

2.3.7 Plotting Variables

When the simulation is completed, a tree view of variables that can be plotted will appear under the **Plot** tab of the **Experiment Browser**. The **Experiment Browser** is the window on the left-hand side in the *Simulation Center*.

Browse the tree and select the variable in which you are interested. In the example depicted in Figure 2-13, the absolute angular velocity, `w`, of the `inertia1` component is selected by checking the box in front of `w`. When the box is checked, a window appears with a plot of the selected variable versus time.



Name	Unit	Description
constantVoltage1		
emf		
ground1		
inductor1		
inertia1		
flange_a		
flange_b		
a	rad/s2	Absolute angula...
der(phi)	rad/s	Derivative of ine...
der(w)	rad/...	Derivative of ine...
J	kg.m2	Moment of inertia
phi	rad	Absolute rotatio...
stateSelect		Priority to use p...
✓ w	rad/s	Absolute angula...
resistor1		

```
Server listening on 127.0.0.1:64321
Initialization finished.
Simulation stopped at time: 10
Simulation took 0.47 seconds of CPU time.
Total number of function evaluations: 1251
Total number of events: 1
Total number of step events (dynamic state switches): 0
Max step size: 0.777711
Min step size: 1e-05

Simulation exited at 10:33:54
```

Figure 2-13: A plot of the absolute angular velocity, w, versus time.

As you can see, the time interval that was chosen by default was not enough to observe the time at which the rotational speed settles. Therefore, we want to extend the simulation interval. This is done by clicking on the **Settings** tab of the **Experiment Browser** located in the upper left-hand corner as seen in Figure 2-14 below.

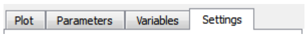
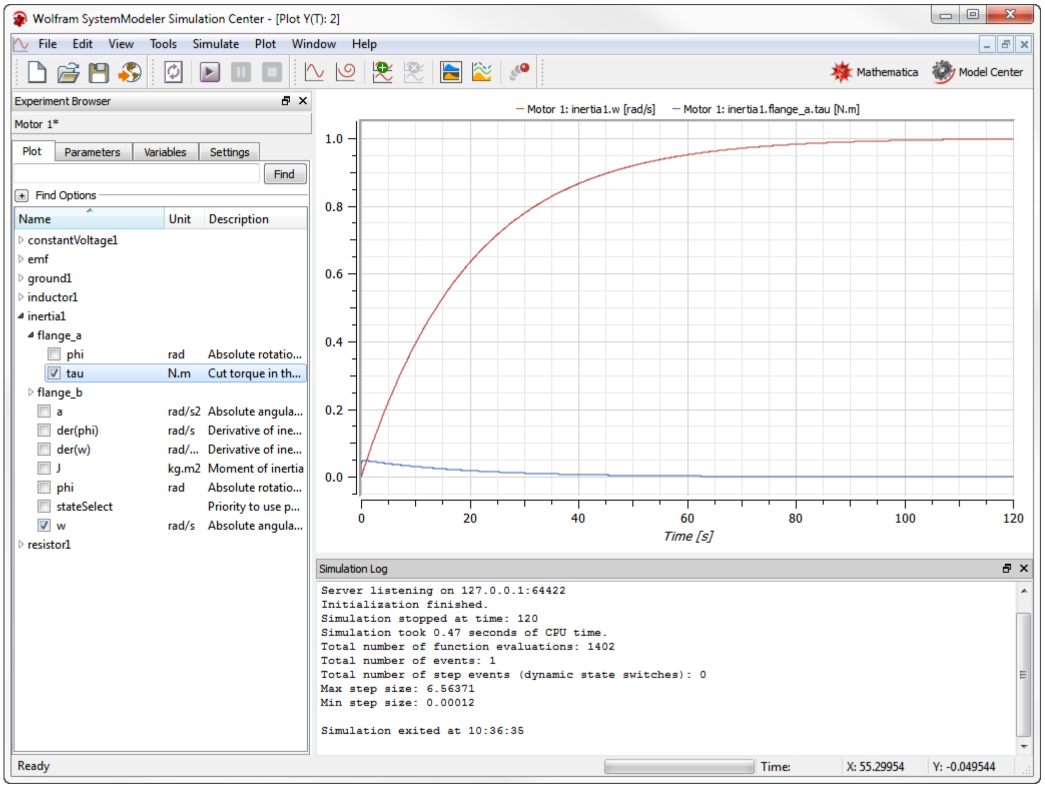


Figure 2-14: The tabs of the **Experiment Browser**.

Set the start time to `0` and the end time to `120` and simulate the model again. To compare the torque that affects the inertia with the rotational speed, we select `tau` under `flange_a`. The plot window updates automatically and will appear as in Figure 2-15.


Wolfram SystemModeler Simulation Center - [Plot Y(T): 2]
File Edit View Tools Simulate Plot Window Help
Mathematica
Model Center
Experiment Browser
Motor 1*
Plot Parameters Variables Settings
Find
Find Options
— Motor 1: inertia1.w [rad/s] — Motor 1: inertia1.flange_a.tau [N.m]
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120
Time [s]
Simulation Log
Ready
Time:
X: 55.29954
Y: -0.049544

Name	Unit	Description
constantVoltage1		
emf		
ground1		
inductor1		
inertia1		
flange_a		
phi	rad	Absolute rotatio...
✓ tau	N.m	Cut torque in th...
flange_b		
a	rad/s2	Absolute angula...
der(phi)	rad/s	Derivative of ine...
der(w)	rad/...	Derivative of ine...
J	kg.m2	Moment of inertia
phi	rad	Absolute rotatio...
stateSelect		Priority to use p...
✓ w	rad/s	Absolute angula...
resistor1		

```
Server listening on 127.0.0.1:64422
Initialization finished.
Simulation stopped at time: 120
Simulation took 0.47 seconds of CPU time.
Total number of function evaluations: 1402
Total number of events: 1
Total number of step events (dynamic state switches): 0
Max step size: 6.56371
Min step size: 0.00012

Simulation exited at 10:36:35
```

Figure 2-15: The line-plot for the variable `w` of `inertia1` and `tau` of `inertia.flange_a`, simulated from `0` to `120`.

It is possible to change parameter values as well as state variable initial values in the **Experiment Browser**. Choosing **Simulate** after editing any values allows the model to be simulated with the new parameter settings.

2.3.8 Analyzing in *Mathematica*

With *SystemModeler* it is also possible to interact with the *Mathematica* notebook environment using the *SystemModeler Link* in *Mathematica*. Click the **Mathematica** button in either the *Model Center* or the *Simulation Center* toolbar. If the *Mathematica* icon is gray, the link needs to be configured; please see Section 1.4 on page 3.

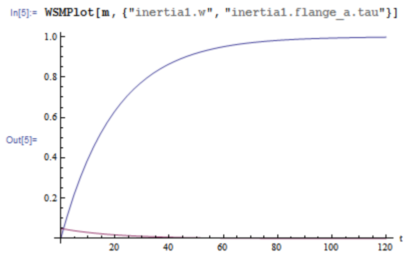
Mathematica will open with the guide page for Wolfram *SystemModeler Link* (*WSMLink*). Open a new notebook by selecting **File ▸ New ▸ Notebook** in *Mathematica*. Load *WSM-Link* by evaluating the following command in the notebook.

```
Needs["WSMLink`"]
```

Mathematica commands are evaluated by pressing **Shift** + **Enter**. When the link is loaded, it is possible to simulate the motor model using the `WSMSimulate` command.

```
m=WSMSimulate["Motor",{0,120}]
```

This command simulates the model `Motor` in the range `0` to `120` and returns a `WSMSimulationData` object with all the available output signals. The object contains all simulation parameters and variables. These can be plotted using the `WSMPlot` command as illustrated in Figure 2-16.



```
In[5]:= WSMPlot[m, {"inertia1.w", "inertia1.flange_a.tau"}]
Out[5]=
```

Figure 2-16: The line plot for the variable `w` of `inertia1` and `tau` of `inertia.flange_a`, simulated from `0` to `120`.

This is the same variable that we plotted in Figure 2-15 using *Simulation Center*, hence we obtain the same result.

Chapter 3: ***Model Center***

This chapter describes the *SystemModeler* environment for developing models.

3.1 Introduction

By default *Model Center* starts with four windows (see figure below): the **Class Browser** (A), a class window (B), the **Components** window (C), and a window with the **Parameters** view, **Variables** view, **Constants** view, and **Messages** view (D); see Figure 3-1.

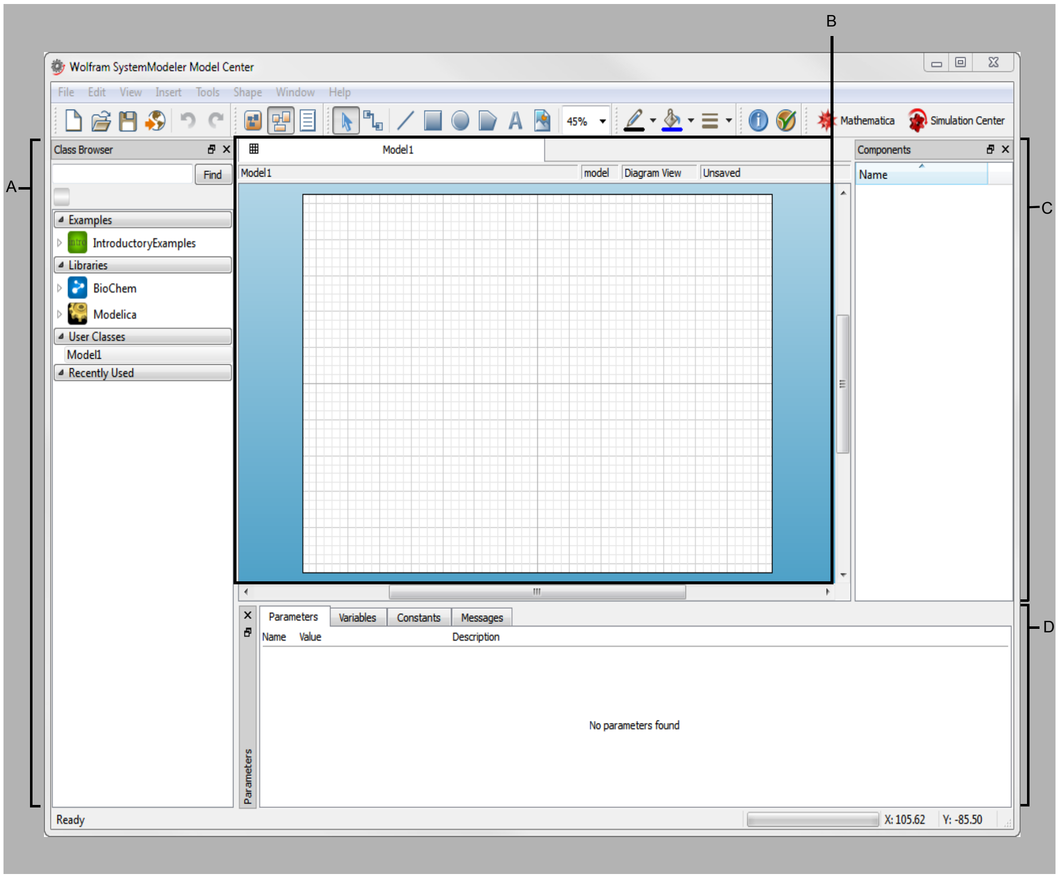
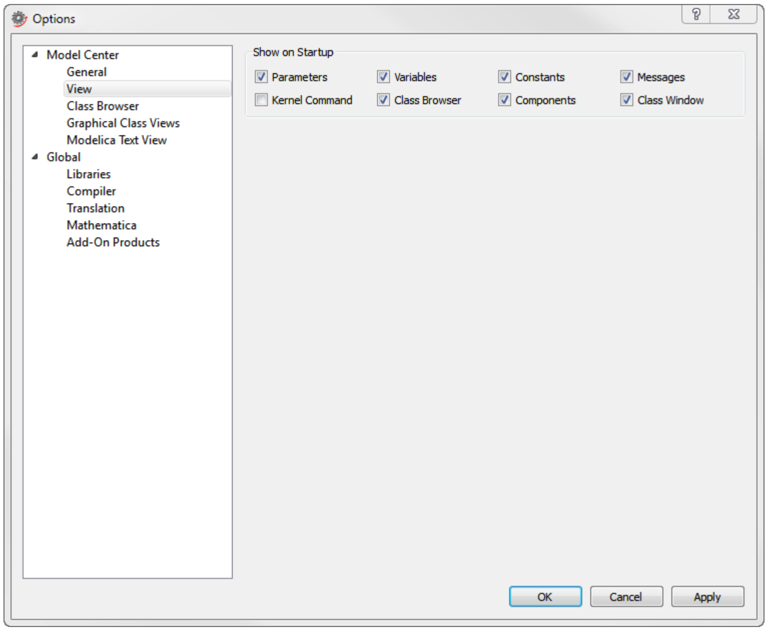


Figure 3-1: An overview of *Model Center*.

3.1.1 Customizing the Windows

All *Model Center* windows, except the class window, can be dragged and dropped anywhere within the main window or outside it as floating windows.

The windows appearing when starting *Model Center* can be specified in the **View** options of the **Options** dialog box. To open the **Options** dialog box, choose **Tools ▸ Options**. All settings in the **Options** dialog box are automatically saved when closing the dialog box and restored the next time *Model Center* is started.



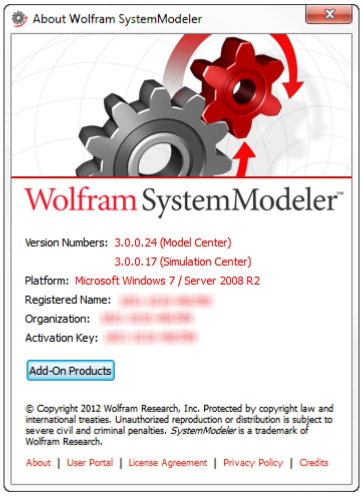
Show on Startup

✓ Parameters	✓ Variables	✓ Constants	✓ Messages
☐ Kernel Command	✓ Class Browser	✓ Components	✓ Class Window

Figure 3-2: Specifying what windows to open when starting *Model Center*.

3.1.2 Version and License Details

The version of *Model Center* can be found by choosing **About SystemModeler** from the **Help** menu.



Version Numbers: 3.0.0.24 (Model Center)
3.0.0.17 (Simulation Center)
Platform: Microsoft Windows 7 / Server 2008 R2
Registered Name: [redacted]
Organization: [redacted]
Activation Key: [redacted]

© Copyright 2012 Wolfram Research, Inc. Protected by copyright law and international treaties. Unauthorized reproduction or distribution is subject to severe civil and criminal penalties. *SystemModeler* is a trademark of Wolfram Research.

About | User Portal | License Agreement | Privacy Policy | Credits

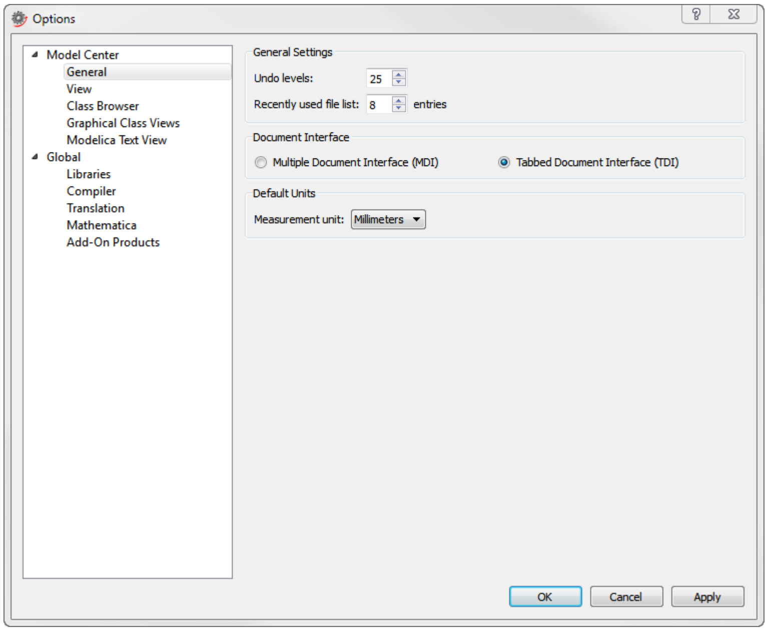
Figure 3-3: The **About SystemModeler** dialog box.

The dialog will also show the name of the licensee and the activation key. Click the **Add-On Products** button to get information about available add-ons.

3.1.3 Document Interface

The default configuration of *Model Center* is to use a tabbed document interface (TDI), where classes can be viewed and edited in a tabbed environment. If you prefer to use a multiple document interface (MDI), where classes are viewed and edited in windows, you can switch to MDI in the **Options** dialog box. Open the **Options** dialog box by choosing **Tools ▸ Options**. The document interface settings are located in the **General** view.

One of the main benefits of TDI is the overview of open classes that it provides. The tabbed environment also makes it easier to switch back and forth between classes. What it does not allow is to view multiple classes side by side. This can only be accomplished by switching to the MDI, which gives you control over the size and position of the windows.



General Settings

Undo levels: 25

Recently used file list: 8 entries

Document Interface

Multiple Document Interface (MDI) [✓ Tabbed Document Interface (TDI)]

Default Units

Measurement unit: Millimeters

Figure 3-4: Switching between MDI and TDI environments.

3.1.4 Default Units

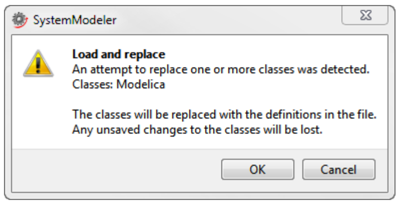
The default units used in *Model Center* can be changed in the **Options** dialog box. Open the **Options** dialog box by choosing **Options** from the **Tools** menu. The default units setting is located in the **General** view.

3.1.5 Loading Classes

At any time you can load one or more Modelica files (*.mo) in *Model Center* by choosing **File ▸ Open**. Alternatively, you can drop Modelica files anywhere on the *Model Center* window. Recently used files are available in the **File ▸ Recent Files** menu. Click one of the menu items to load the specified file. The number of items available in the **Recent Files** menu can be specified in the **General** view of the **Options** dialog box, accessible from the **Tools** menu.

As soon as *Model Center* is finished reading the contents of the files, the loaded classes will appear in the **Class Browser**.

Any attempts to redefine existing classes when loading a file will be detected and you will be given a chance to abort the operation. If you choose to proceed, the existing classes will be replaced with the class definitions in the file.



Load and replace
An attempt to replace one or more classes was detected.
Classes: Modelica

The classes will be replaced with the definitions in the file.
Any unsaved changes to the classes will be lost.

Figure 3-5: An attempt to redefine the Modelica Standard Library when loading a file was detected.

3.1.6 Refreshing Classes

In some rare cases it may be necessary to force a refresh of a class in *Model Center*. When refreshing a class, the class definition is refetched from the *SystemModeler* kernel and all elements of the class, including child classes, are updated in *Model Center*.

To refresh a class, right-click its name in the **Class Browser** and choose **Refresh** from the popup menu, or if the class is open, right-click an empty area in a graphical view and choose **Refresh** from the menu.

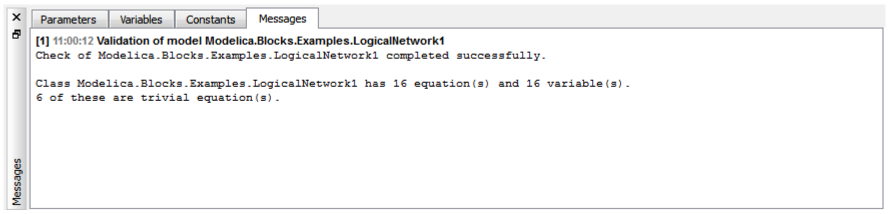
3.1.7 Validating Classes

Click the **Validate Class** button on the toolbar to validate the class of the active class window. You may also right-click any class in the **Class Browser** and choose **Validate** from the menu.

Figure 3-6: The **Validate Class** button on the **Tools** toolbar.

All syntactic and many semantic errors in the class will be detected and reported. Note that the semantic check cannot be performed if the class has syntax errors. A report of the semantic check is shown in the **Messages** view of the window at the bottom of *Model Center*.

If the semantic check is successfully completed, the total number of equations and variables, as well as the number of trivial equations of the class, will be listed at the end of the report.



```
[1] 11:00:12 Validation of model Modelica.Blocks.Examples.LogicalNetwork1
Check of Modelica.Blocks.Examples.LogicalNetwork1 completed successfully.

Class Modelica.Blocks.Examples.LogicalNetwork1 has 16 equation(s) and 16 variable(s).
6 of these are trivial equation(s).
```

Figure 3-7: The validation report of the `Modelica.Blocks.Examples.LogicalNetwork1` model.

3.1.8 Simulating Classes

Classes are simulated using *Simulation Center*. *Simulation Center* can be started from within *Model Center* by clicking the **Simulation Center** button on the **Applications** toolbar, or by choosing **Simulation Center** from the **Tools** menu.

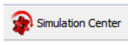


Figure 3-8: The **Simulation Center** button on the **Applications** toolbar.

For more information on simulation, please see Chapter 4.

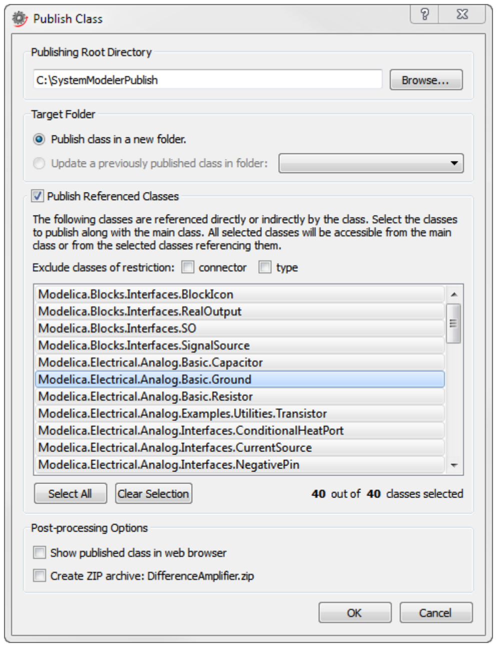
3.1.9 Publishing Classes

Using the publisher tool it is possible to generate documentation for Modelica classes that can be viewed in any web browser. The generated documentation is standalone and does not require *SystemModeler*.

When publishing a class, pages with information about the class, such as parameters, variables, constants, and components are automatically generated. The publisher extracts information from the class in order to create these pages, so the more details you provide in the form of component descriptions, documentation annotations, and so on, the more information will be available in the generated documentation.

The publisher also generates an interactive graphical representation of the **Diagram View** of each published class. In order to visualize the graphics in a web browser, the web browser needs to have the Microsoft Silverlight plugin installed.

To publish a class, open the class and choose **Publish Class** from the **File** menu. This will open a dialog box that will let you configure the publisher.



Publishing Root Directory: C:\SystemModelerPublish

Target Folder
- [✓] Publish class in a new folder.
- [ ] Update a previously published class in folder:

[✓] Publish Referenced Classes

The following classes are referenced directly or indirectly by the class. Select the classes to publish along with the main class. All selected classes will be accessible from the main class or from the selected classes referencing them.

Exclude classes of restriction: [ ] connector [ ] type

Modelica.Blocks.Interfaces.BlockIcon
Modelica.Blocks.Interfaces.RealOutput
Modelica.Blocks.Interfaces.SO
Modelica.Blocks.Interfaces.SignalSource
Modelica.Electrical.Analog.Basic.Capacitor
Modelica.Electrical.Analog.Basic.Ground
Modelica.Electrical.Analog.Basic.Resistor
Modelica.Electrical.Analog.Examples.Utilities.Transistor
Modelica.Electrical.Analog.Interfaces.ConditionalHeatPort
Modelica.Electrical.Analog.Interfaces.CurrentSource
Modelica.Electrical.Analog.Interfaces.NegativePin

40 out of **40** classes selected

Post-processing Options
- [ ] Show published class in web browser
- [ ] Create ZIP archive: DifferenceAmplifier.zip

Figure 3-9: The publisher configuration dialog box.

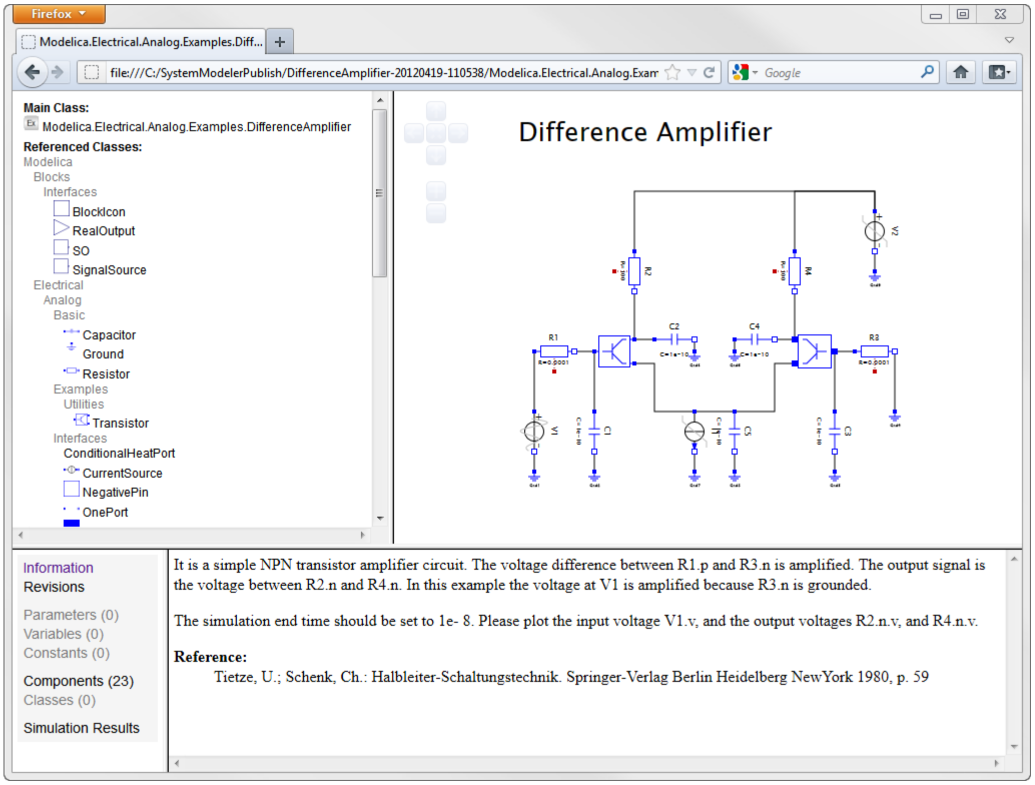
The different sections of the dialog box are described in detail below.

- **Publishing Root Directory**
  The directory in which all generated files and folders are saved.
- **Target Folder**
  Select an existing folder to update a previously published class.
- **Publish Referenced Classes**
  Select classes to publish along with the main class to enable hierarchical browsing of the published class.
- **Post-processing Options**
  Specifies the post-processing options.
  - **Show published class in web browser**. Shows the published class in the default web browser.
  - **Create ZIP archive**. Archives all generated files and folders of the published class in a single ZIP file.

The published version of the class `Modelica.Electrical.Analog.Examples.DifferenceAmplifier` can be seen in Figure 3-10 below.

As seen in the figure, the view of a published class is divided up into four sections. In the top-right section the **Diagram View** of the class is found. This view is also interactive. Click a component to select it and view related information in the bottom-right section. When no component is selected, information about the published class is shown in the bottom-right section. If the class of a component was published along with the main class, it is possible to view the component class by double-clicking the component. In the top-left corner of the **Diagram View**, you will also find a control panel that is used to pan and zoom the **Diagram View**. The control panel becomes visible when the mouse cursor is within its area. Finally, the top-left view is a tree view of all referenced classes that were published along with the published class.

The bottom-left section controls what information is shown about the published class or any selected component of the class. For information on how to publish simulation results, see Section 4.4.14 on page 126.


Firefox
Modelica.Electrical.Analog.Examples.Diff...
file:///C:/SystemModelerPublish/DifferenceAmplifier-20120419-110538/Modelica.Electrical.Analog.Exam
Google
Difference Amplifier
R1
R2
R3
R4
C1
C2
C3
C4
V1
V2

Main Class:
Modelica.Electrical.Analog.Examples.DifferenceAmplifier
Referenced Classes:
Modelica
Blocks
Interfaces
BlockIcon
RealOutput
SO
SignalSource
Electrical
Analog
Basic
Capacitor
Ground
Resistor
Examples
Utilities
Transistor
Interfaces
ConditionalHeatPort
CurrentSource
NegativePin
OnePort

Information
Revisions
Parameters (0)
Variables (0)
Constants (0)
Components (23)
Classes (0)
Simulation Results

It is a simple NPN transistor amplifier circuit. The voltage difference between R1.p and R3.n is amplified. The output signal is the voltage between R2.n and R4.n. In this example the voltage at V1 is amplified because R3.n is grounded.

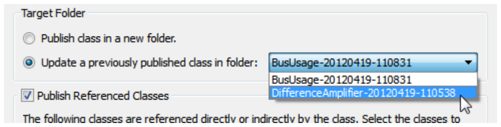
The simulation end time should be set to 1e- 8. Please plot the input voltage V1.v, and the output voltages R2.n.v, and R4.n.v.

Reference:
Tietze, U.; Schenk, Ch.: Halbleiter-Schaltungstechnik. Springer-Verlag Berlin Heidelberg NewYork 1980, p. 59

Figure 3-10: `Modelica.Electrical.Analog.Examples.DifferenceAmplifier` published.

Updating a Previously Published Class

If you have made changes to a class that you already have published, you may update the published result by changing the target folder in the dialog box to the folder containing the previously published class. The folders presented in the drop-down box are the folders found in the specified publishing root directory.


Target Folder

Publish class in a new folder.

Update a previously published class in folder: BusUsage-20120419-110831

BusUsage-20120419-110831

DifferenceAmplifier-20120419-110538

Publish Referenced Classes

The following classes are referenced directly or indirectly by the class. Select the classes to

Figure 3-11: Updating a previously published class.

When updating a previously published class, any published simulation results will remain intact and accessible from the updated version of the published class.

Limitations

Due to technical reasons, there are a few limitations that should be taken into consideration when publishing classes.

- Text items: the font name and font style attributes of text items are not supported.
- Rectangle items: the border style attribute of rectangle items is not supported.
- Ellipse items: the begin and end angle attributes of ellipse items are not supported.
- Filled shapes: the tiled fill patterns (**Horizontal**, **Vertical**, **Cross**, **Forward**, **Backward**, and **Cross diagonal**) are not supported.

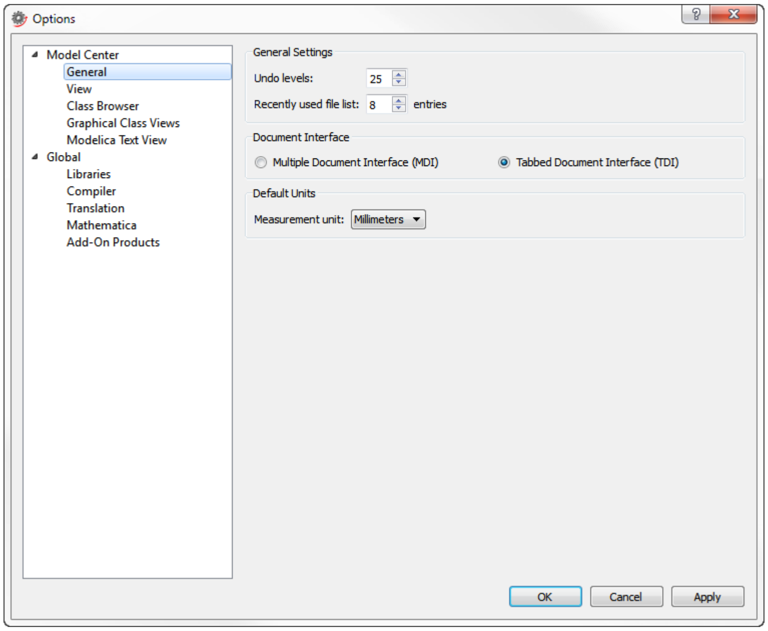
3.1.10 Undoing Mistakes

In the event of a mistake, you can undo the most recent actions performed. Choose **Edit ▸ Undo** to undo the very last undoable action you performed. If you later decide you did not want to undo a certain action, choose **Edit ▸ Redo**. Please note that it is not possible to undo all actions. The **Undo** and **Redo** operations can also be reached from the **Standard** toolbar.

Figure 3-12: The **Undo** and **Redo** buttons on the **Standard** toolbar.

Setting the Number of Undo Levels

The maximum number of consecutive actions you can undo is by default 100. This number can be changed to any number between 0 and 999. Open the **Options** dialog box by choosing **Tools ▸ Options**. The setting is available in the **General** view and is automatically saved when closing the dialog box and restored the next time *Model Center* is started.



General Settings
Undo levels: 25
Recently used file list: 8 entries

Document Interface
Multiple Document Interface (MDI) [✓ Tabbed Document Interface (TDI)]

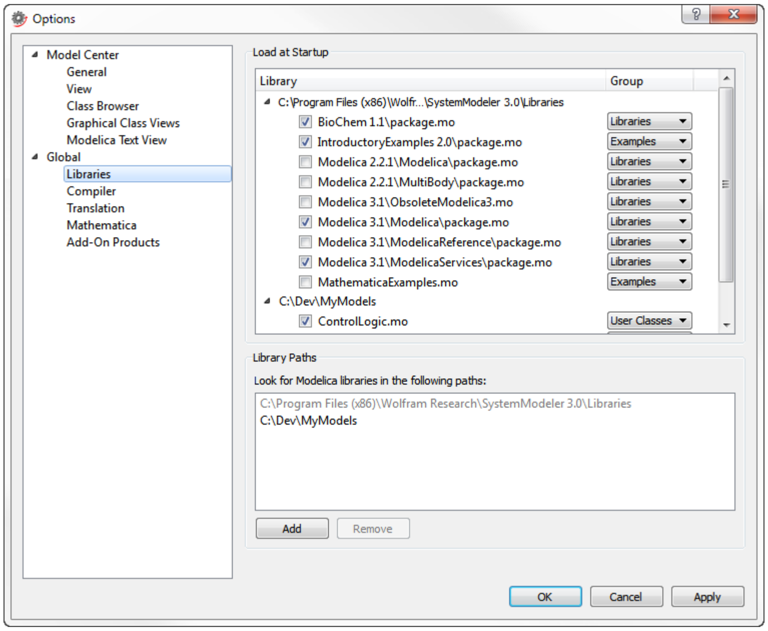
Default Units
Measurement unit: Millimeters

Figure 3-13: The **General** view of the **Options** dialog box.

3.1.11 Specifying Libraries to Load at Startup

It is possible to customize the set of libraries that is loaded at the startup of *SystemModeler*. By default the Modelica Standard Library, the *SystemModeler* `IntroductoryExamples`, the *SystemModeler* `MathematicaExamples`, and the `BioChem` libraries are loaded.

The settings that control this are available in the **Options** dialog box. Open the **Options** dialog box by choosing **Options** from the **Tools** menu and go to the **Libraries** section. You will find that the right-hand side of the dialog box is divided up into two parts. The lower part lists all paths in which Modelica (*.mo) files are searched for, while the upper part lists the files found in those paths.



Load at Startup

Library	Group
C:\Program Files (x86)\Wolfr...\SystemModeler 3.0\Libraries	
✓ BioChem 1.1\package.mo	Libraries
✓ IntroductoryExamples 2.0\package.mo	Examples
☐ Modelica 2.2.1\Modelica\package.mo	Libraries
☐ Modelica 2.2.1\MultiBody\package.mo	Libraries
☐ Modelica 3.1\ObsoleteModelica3.mo	Libraries
✓ Modelica 3.1\Modelica\package.mo	Libraries
☐ Modelica 3.1\ModelicaReference\package.mo	Libraries
✓ Modelica 3.1\ModelicaServices\package.mo	Libraries
☐ MathematicaExamples.mo	Examples
C:\Dev\MyModels	
✓ ControlLogic.mo	User Classes

Library Paths

Look for Modelica libraries in the following paths:

```
C:\Program Files (x86)\Wolfram Research\SystemModeler 3.0\Libraries
C:\Dev\MyModels
```

Figure 3-14: Adding a custom library to automatically load at startup.

To add your own libraries or models, begin with adding the paths to the files in the **Library Paths** section. Once added, any files found within those paths will appear in the upper section of the dialog box. Check the files you want to be loaded at startup.

The setting is automatically saved when closing the dialog and restored the next time *Model Center* is started.

Setting the Logical Group for a Library

Each library that is loaded at startup is associated with a logical group. This group controls which section of the **Class Browser** the library will appear in when loaded. By default new libraries will be associated with the **User Classes** group. To change the associated group for a library, click the corresponding drop-down box and choose a different group.

The setting is automatically saved when closing the dialog and restored the next time *Model Center* is started.

3.1.12 Setting the Screen Resolution

To make the size of graphic objects on screen match their actual specified size, you need to set the screen resolution in the **Options** dialog box. Choose **Options** from the **Tools** menu and edit the **Screen resolution (DPI)** box in the **Display** view of the **Graphical Class Views** options.

For instance, the screen resolution (in dots per inches) for a monitor measuring 19 inches diagonally, and with a resolution of 1280x1024 pixels, equals

sqrt(1280x1280 + 1024x1024) / 19.

This leaves us with a screen resolution of approximately 86 dpi.

The setting is automatically saved when closing the dialog and restored the next time *Model Center* is started.

3.2 Class Browser

The **Class Browser** is used to browse packages and components of libraries, and to perform operations on classes, such as renaming, saving, and more.

3.2.1 The Modelica Class Concept

In Modelica, classes are used to model systems. From a class it is possible to create any number of objects that are known as instances of that class. An instance of a class is also often referred to as a component. As a class is very general and does not give any information about what it defines; several kinds of restricted classes are available. The class restriction of a class specifies the constraints that apply to the class.

- *model*. The model class is by far the most commonly used kind of class for modeling purposes. The only restriction is that a model may not be used in connections.
- *package.* A package is primarily used to organize Modelica code and may only contain declarations of other classes and constants. No variable declarations are allowed within a package.
- *function*. The function concept in Modelica corresponds to mathematical functions without external side effects and may be recursive.
- *connector*. Connector classes are typically used as templates for connectors. A connector is a connection point used for communicating data between objects.

- *record*. The record class is used for specifying data without behavior.
- *block*. A block is a class for which the causality (whether the data-flow direction is input or output) of each of its variables is known.
- *type*. Classes of restriction type are used to define custom types, extending a predefined type, enumeration, array of type, or a class extending from another type.

3.2.2 Browsing Classes

Modelica classes and their hierarchical organization are visualized using grouped tree views in the **Class Browser**. Classes are grouped into packages and components, where the items of each group are sorted alphabetically. Packages are represented as branches of the tree and components as leaves. To show the contents of a package, expand it by clicking the symbol to the left of its icon (or name if the package has no icon). Double-clicking also expands packages.

Clicking the symbol to the left of an expanded package will show it as collapsed again, hiding its contents.

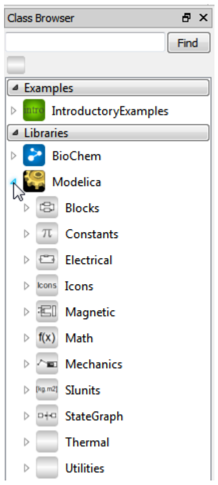


Figure 3-15: Browsing the Modelica Standard Library.

To view the contents of a package as a separate group within the **Class Browser**, right-click its name and choose **View as Group**. This can prove useful if you use components from more than one package when designing a model. By opening the packages of interest as separate groups, you can minimize the scrolling up and down needed in order to find the components.

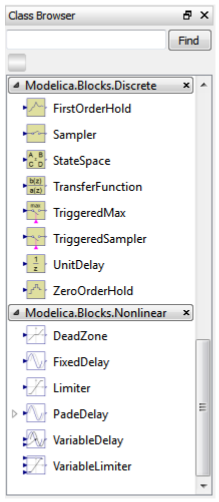


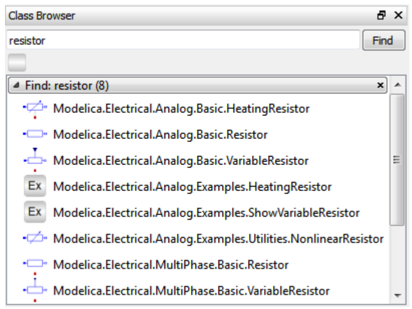
Figure 3-16: Viewing the contents of two packages in separate group views.

The icon of a package or class can be copied to the clipboard as an image by right-clicking its name and choosing **Copy Icon as Image**.

It is possible to view balloon help (on-screen information) for a class by moving the mouse cursor over its icon and holding it still for a short period of time. The class name and description (if available) are shown.

3.2.3 Finding Classes

If you know the name of the class you are looking for or parts of its name, you can use the search feature of the **Class Browser** to find it. This is especially useful if you do not know the exact location of the class. Type in the text to search for in the text box at the top of the **Class Browser** and click the **Find** button. The result of the search operation is presented as a separate group at the top part of the **Class Browser**. Any class whose name contains the text searched for will be present in the group. The number of hits that the search resulted in is displayed within parentheses in the title bar of the group.



Modelica.Electrical.Analog.Basic.HeatingResistor
Modelica.Electrical.Analog.Basic.Resistor
Modelica.Electrical.Analog.Basic.VariableResistor
Modelica.Electrical.Analog.Examples.HeatingResistor
Modelica.Electrical.Analog.Examples.ShowVariableResistor
Modelica.Electrical.Analog.Examples.Utilities.NonlinearResistor
Modelica.Electrical.MultiPhase.Basic.Resistor
Modelica.Electrical.MultiPhase.Basic.VariableResistor

Figure 3-17: Finding resistor classes using the search feature of the **Class Browser**.

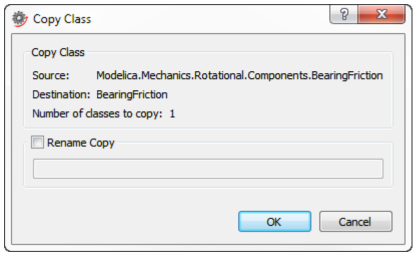
You can go directly to the location of a class in the **Class Browser** by right-clicking components in the **Components** window or the **Diagram View**, and choosing **Go to Class in Class Browser** in the popup menu.

3.2.4 Copying Classes

Classes can be copied using drag-and-drop within the **Class Browser** or by using menus. The destination of the copy operation can be any other class, as long as the class is not read-only. Drag the class you want and drop it on the class in which you want the copy to be created. To copy a class to the top level of the class hierarchy, drop the class on an empty area in the **Class Browser**, or on the item representing the root of the group.

Another way to copy a class is to right-click it and choose **Copy**. The class is copied to the clipboard and can be inserted to any number of classes by first selecting the class and then choosing **Paste** from the **Edit** menu.

Once the class is dropped or a class is pasted using the menu, a confirmation dialog box will be shown. The dialog box will also let you give the copy a name different from the original. Note that copying very complex classes or packages with many local classes is a time-consuming task and may take a considerable amount of time.


Copy Class

Copy Class

Source: Modelica.Mechanics.Rotational.Components.BearingFriction

Destination: BearingFriction

Number of classes to copy: 1

Rename Copy

Figure 3-18: Copying the `Modelica.Mechanics.Rotational.Components.BearingFriction` class.

3.2.5 Opening and Editing Classes

To open a class, right-click its name and choose **Open** from the popup menu. For classes that are not packages, double-clicking its name is a more convenient way. The class will appear in a new class window unless the class has already been opened, in which case its class window will become the active class window.

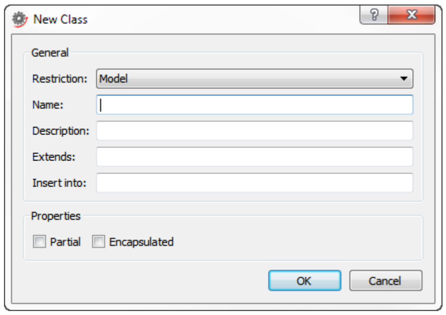
The default view of the class window when opening a class depends on the restriction of the class. For example, the default view of a package is the **Icon View**, while the default view of a model is the **Diagram View**. The default view can be changed for a specific class by specifying a preferred view as a Modelica annotation within the definition of the class. For more information on how to create such an annotation, see Section 3.3.1 on page 46.

In case the file in which the class is saved has the read-only attribute set, the class becomes read-only as well. A read-only class is still possible to open in a class window, but the class cannot be edited. All classes in the Modelica Standard Library and the `Introducto-ryExamples` library are read-only.

3.2.6 Creating Classes

New classes are created either by using the popup menus of the **Class Browser** or the **File** menu. Choose **New Class** from the popup menu that appears when right-clicking in an empty area of the **Class Browser**, or choose **New Class** from the **File** menu. This will open a dialog box in which you will be able to specify the attributes of the class.

If you want to insert the new class into an existing class, you can right-click the parent class in the **Class Browser** and choose **New Class** from the popup menu. The **Insert into** text box of the dialog box will then be filled in automatically for you.


New Class

General

Restriction: Model

Name:

Description:

Extends:

Insert into:

Properties

Partial
Encapsulated

Figure 3-19: Creating a new class.

The different sections of the dialog box are described in detail below.

- **General**
  Specifies general information about the class, such as the restriction, name, description, etc.
  - **Restriction**. Specifies the constraints that are applied to the class. For more information about class restrictions, see Section 3.2.1 on page 34.
  - **Name**. Identifies the class.
  - **Description** (optional). Describes the class. The class description appears along with the name of the class as balloon help for the class in the **Class Browser**.
  - **Extends** (optional). Specifies the base classes to inherit from. Multiple base classes can be specified by using a comma to separate them. In many situations it is convenient to declare a base class with a general interface that is extended when creating more specialized classes. You will see that this is a common practice in the Modelica Standard Library, where for example almost all two pin analog electrical components inherit the `Modelica.Electical.Analog.Interfaces.OneP-ort` class.
    If you do not know the full path to the class you want to extend, you can use the **Class Browser** to search for the class and then drag it from the **Class Browser** and drop it on the text box. See Section 3.2.3 on page 36 for more information on how to find classes using the search functionality of the **Class Browser**.
  - **Insert into** (optional). Specifies the location of the class. If you create a class using the **File** menu, the **Insert into** text box will be empty, which means that the class will be created at the top level of the class hierarchy. Creating the class using the

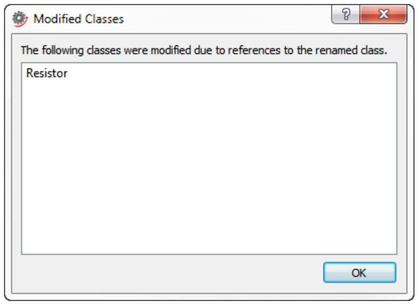
popup menu of the **Class Browser** will insert the class in the class currently selected in the **Class Browser** (if any).

- **Properties**
  Specifies the partial and encapsulated properties of the class.
  - **Partial**. The partial property is used to indicate that a class is incomplete such that it cannot be instantiated. A partial class can only be used as a base class. Packages cannot be declared partial.
  - **Encapsulated**. An encapsulated class represents an independent unit of code. All dependencies outside the class must be explicitly stated using import statements.

3.2.7 Renaming Classes

A class can be renamed at any time by right-clicking the name of the class in the **Class Browser**, choosing **Rename** from the popup menu, and editing the name in the **Class Properties** dialog box. It is also possible to rename a class by selecting its icon and pressing the **F2** key.

When renaming a class from the **Class Properties** dialog box, all references to the renamed class will be updated automatically in all currently loaded classes. For example, if you have a class `Resistor` that inherits the class `TwoPin` and you rename class `TwoPin` to `TwoPinInterface`, the `Resistor` class will automatically update its inheritance to class `TwoPinInterface`. When the renaming operation is completed, a list of all classes that were modified due to references to the renamed class is shown.


Modified Classes

The following classes were modified due to references to the renamed class.

Resistor

Figure 3-20: List of modified classes after renaming a class.

3.2.8 Deleting Classes

To delete a model or any other class, select it by clicking its name and choose **Edit ▸ Delete**, or right-click the name of the class in the **Class Browser** and choose **Delete** from the popup menu. No files are deleted when deleting a class. However, if you have more than one class associated with the same file, for instance a package named `Electrical` containing a class named `Resistor`, deleting the `Resistor` class and then saving the package will remove the deleted class `Resistor` from the file.

When deleting a top-level class, for instance `Modelica`, the class is unloaded; no files are deleted or affected.

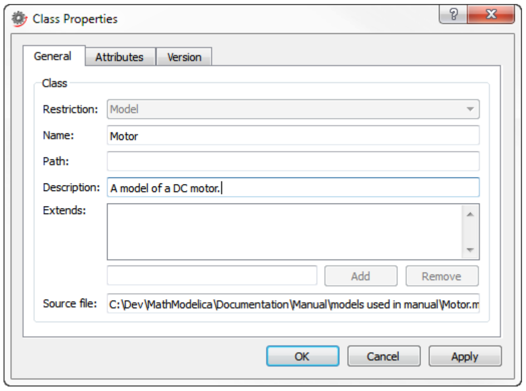
When deleting or unloading a class, all references to the class in other classes will become unresolved. As a consequence, the components of the class will no longer be graphically visible in *Model Center*. Classes with references to the deleted class will not be modified, however it will be impossible to simulate them until the unresolved references have been addressed.

3.2.9 Editing the Properties of Classes

The properties of a class, such as its name, description, and so on, can be viewed and edited from the **Class Properties** dialog box. The **Class Properties** dialog box is reached by right-clicking the name of a class in the **Class Browser** and choosing **Properties** from the popup menu.

The dialog box is divided into three views:

- **General**. Lets you view and edit general information about the class.
- **Attributes**. Lets you edit various attributes of the class.
- **Version**. Shows version information of the library of which the class is a part.



Restriction: Model
Name: Motor
Path:
Description: A model of a DC motor.
Extends:
Source file: C:\Dev\MathModelica\Documentation\Manual\models used in manual\Motor.m

Figure 3-21: Editing the class properties of a DC motor model.

The sections of the dialog box are described in detail below.

- **Class**, **General** view
  Specifies general information about the class, such as name and description.
  - **Restriction**. Specifies the constraints that are applied to the class. For more information about class restrictions, see Section 3.2.1 on page 34. The class restriction cannot be edited from the dialog box. If you need to change the restriction, it can be done in the **Modelica Text View** of the class.
  - **Name.** Identifies the class. The class can be renamed by editing its name. For information on how references are automatically updated when a class is renamed, see Section 3.2.7.
  - **Path.** The full path of the class. The path cannot be edited from the dialog box. See Section 3.2.4 on page 37 for information on how to copy a class from one location to another.
  - **Description**. A short description of the class. The class description appears along with the name of the class as balloon help for the class in the **Class Browser**.
  - **Extends**. If the class is extending any classes, these classes will be listed here. You may add or remove inheritance relationships by adding or removing classes to the list using the **Add** and **Remove** buttons.
  - **Source file**. If the class has an associated source file, the full path and name of the file is shown as the bottom item in the **Class** section of the dialog box.
- **Properties**, **Attributes** view
  Specifies various properties of the class. None of the properties listed below can be edited from the dialog box. If you need to change a property, it can be done in the **Modelica Text View** of the class.
  - **Final**. A class declared as final cannot be modified using redeclarations.
  - **Partial**. The partial property is used to indicate that a class is incomplete such that it cannot be instantiated. A partial class can only be used as a base class.
  - **Protected**. Specifies whether the class is a protected class or not. Protected classes are by default not shown in the **Class Browser**. Only local classes can be protected.
  - **Encapsulated**. An encapsulated class represents an independent unit of code. All dependencies outside the class must be explicitly stated using import statements.
- **Library Version**, **Version** view
  Shows version information of the library of which the class is a part. This information, if available, is extracted from the top-level ancestor class.
  - **Version**. The version number.

- **Version Date**. The date of the first version build.
- **Version Build**. The version build number, used for maintenance updates. The higher the number, the more recent the update.
- **Date Modified**. The date of the last version build of the library.
- **Revision Id**. Revision identifier of the version management system used to manage the library.

3.2.10 Saving Classes

To save the class of the active class window, choose **File ▸ Save**. When a class is saved for the first time, a dialog box is shown, letting you choose a file name and location for the class. If the class is located within an unsaved package, you will be asked to choose a file name and location for the package instead as all classes within a package are saved in the same file.

Classes can also be saved using the **Class Browser**. Right-click the name of the class and choose **Save** from the popup menu.

Saving a Copy of a Class

Sometimes you may want to save a class in a new file, for instance to create a new class by modifying an existing class in some library. This can be done by first copying the class and then saving it. A slightly more convenient way is to right-click the class in the **Class Browser** and choose **Save Copy As** from the popup menu.

Choose **Save Copy As** from the **File** menu to copy and save the class of the active class window.

Saving Complete Definitions

At times it can be useful to save a complete definition of a class, including all classes used by it, to a single file. This has the benefit that the saved class becomes independent of other files and libraries.

However, care has to be taken when loading such a file as already loaded libraries will be redefined if the file to load contains a subset of those libraries.

To save a complete definition of the class of the active class window, choose **File ▸ Save Total**. Complete definitions of classes can also be saved using the **Class Browser**. Right-click the name of the class and choose **Save Total** from the popup menu.

3.2.11 Reloading Libraries

It is possible, at any time, to reload the libraries that have been specified to automatically load at the startup of *SystemModeler* (see Section 3.1.11 on page 32) by a simple mouse click. This is useful if you have loaded a total model that has replaced one or more class definitions, for instance the Modelica Standard Library, and you wish to restore the original class definitions.

To reload all startup libraries, choose **Reload Libraries** from the **File** menu.

3.2.12 Switching Modelica Standard Library Versions

SystemModeler supports both Version 2.2.1 and Version 3.1 of the Modelica Standard Library (MSL). The **Class Browser** provides a way to quickly switch MSL versions. Simply right-click on the Modelica package and in the popup menu, choose the version to which you want to switch. *SystemModeler* will unload the currently loaded version of MSL and load the chosen version.

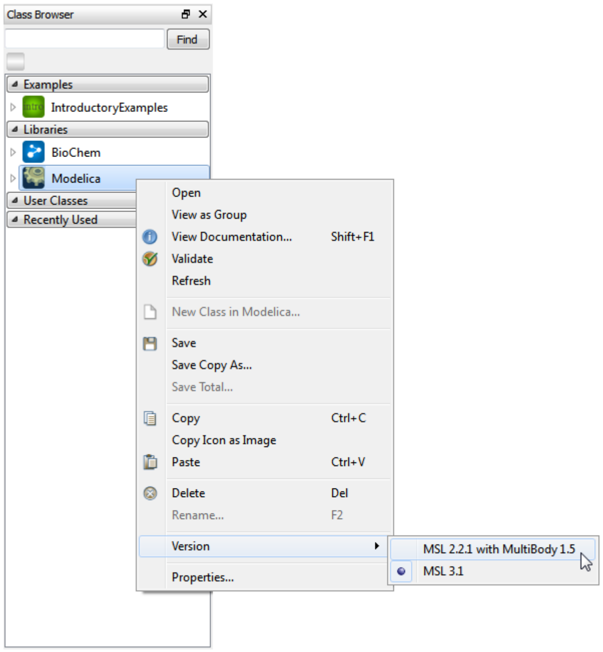
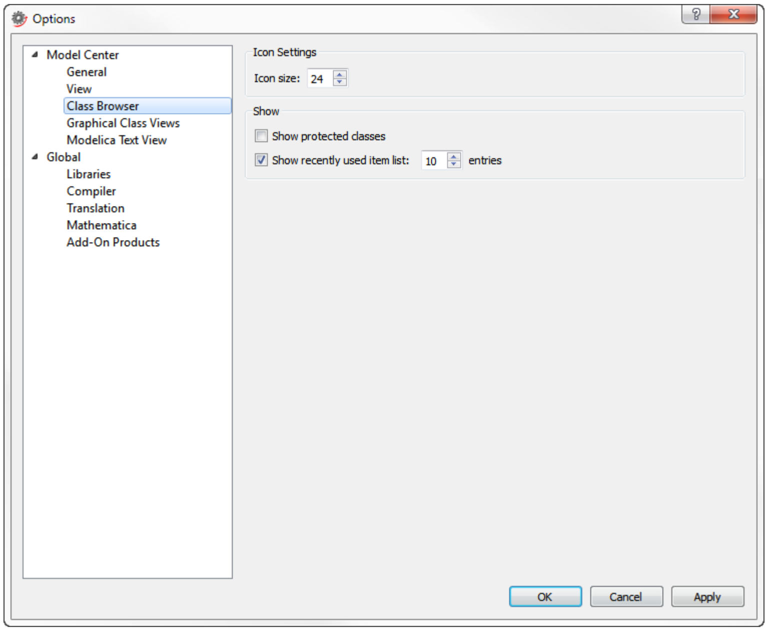


Figure 3-22: Switching MSL versions.

3.2.13 Customizing the Class Browser

The **Class Browser** can be customized by editing the settings found in the **Options** dialog box. The **Options** dialog box can be reached from the **Tools** menu. All settings available in the dialog box are automatically saved when closing the dialog box and restored the next time the model editor is started.



Icon size: 24

Show protected classes [ ]

Show recently used item list: [✓] 10 entries

Figure 3-23: The **Class Browser** view of the options dialog box.

The settings in the **Class Browser** view is described in detail below.

- **Icon size**. Specifies the size (width and height) in pixels of the icons in the **Class Browser**.
- **Show protected classes**. Specifies whether protected classes are shown in the **Class Browser**.
- **Show recently used item list**. Specifies whether recently used items, and the number of recently used items, are shown in the **Class Browser**.

3.3 Class Window

Class windows use three views to represent different aspects of a class:

- **Icon View**, a graphical view showing the icon layer of the Modelica class. The icon layer makes up the icon of the class when the class is used as a component of another model.
- **Diagram View**, a graphical view showing the diagram layer of the Modelica class. The diagram layer represents the composition of a class.
- **Modelica Text View**, which contains the Modelica definition (textual representation) of the Modelica class.

To change the active class window view, choose **View ▸ Class Window** and click the name of the view.

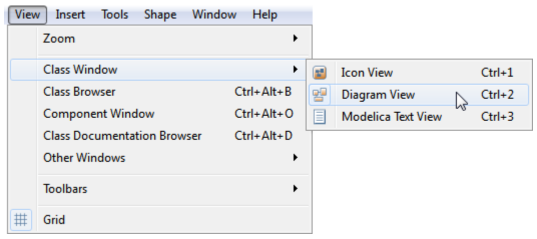


Figure 3-24: Choosing the **Diagram View** as the active class window view.

It is also possible to change the active class view by using the **Icon View**, **Diagram View**, and **Modelica Text View** buttons on the **Class View** toolbar.

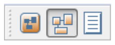


Figure 3-25: The toolbar buttons for changing the active class view.

The name of the class, along with its class restriction, class view, and file name, is displayed in the active class window below its title. An asterisk immediately to the right of the file name is an indication that changes to the class have been made since it was last saved.

3.3.1 Specifying a Preferred View

The default view of the class window, when a class window is opened, depends on the restriction of the class. For example, the default view of a package is the **Icon View**, while the default view of a model is the **Diagram View**. The default view can be overridden by specifying a preferred view as a Modelica annotation within the definition of the class.

An example of a model with an annotation specifying the **Icon View** as the preferred view is given below.

```
model Resistor
  annotation(preferredView="icon");
end Resistor;
```

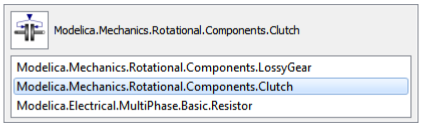
In order to make the **Diagram View** the preferred view, use `"diagram"` instead of `"icon"`; for the **Modelica Text View**, use `"text"`; and to show the documentation of the class when opening the class window, use `"info"`.

To specify a preferred view in a class of your own, switch to the **Modelica Text View** in the class window and type in the annotation on the line below the name of your class.

3.3.2 Switching between Windows

It is possible to have several class windows open at the same time. A list of open class windows is found in the **Window** menu. By clicking one of the window titles in the menu, the window will become the active class window.

A more convenient way of switching between open class windows is to use the **Ctrl** + **Tab** and **Ctrl** + **Shift** + **Tab** combination of keys. A list of all open class windows appears as soon as **Tab** is pressed with **Ctrl** down.


Modelica.Mechanics.Rotational.Components.Clutch

Modelica.Mechanics.Rotational.Components.LossyGear
Modelica.Mechanics.Rotational.Components.Clutch
Modelica.Electrical.MultiPhase.Basic.Resistor

Figure 3-26: Selecting an active class window using the class window switcher.

The list of windows is kept in an order with the most recently used window at the top. While **Ctrl** is down, **Tab** may be pressed and released repeatedly, combined with **Shift** if desired, to cycle through the list of windows. The window list remains open until **Ctrl** is released.

If *Model Center* is configured to use a tabbed document interface (TDI), the tabs below the toolbar can also be used to switch between open class windows. See Section 3.1.3 on page 24 for more information on document interfaces.

3.3.3 Arranging the Windows

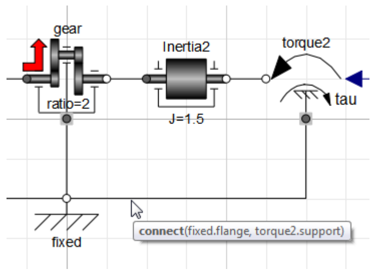
If you have more than one class window open you can view all windows at the same time. To display the windows side-by-side, choose **Tile** from the **Window** menu. To arrange the windows so that the title bar of every window is visible, choose **Cascade** from the **Window** menu.

Note that this is only possible if the model editor is set to use the multiple document interface (MDI). See Section 3.1.3 on page 24 for more information on document interfaces.

3.3.4 Graphical Views

The graphical views and the component window are synchronized to give a consistent view of the class. When a component is selected in the icon or **Diagram View** it also becomes selected in the component window and vice versa.

It is possible to view balloon help (on-screen information) for a component or connection in graphical views by moving the mouse cursor over the item and holding it still for a short period of time. For components the component type, name, and description (if available) are shown. For connections the name of the connected components and connection description (if available) are shown.



connect(fixed.flange, torque2.support)

Figure 3-27: Viewing balloon help for a connection in the **Diagram View**.

Selecting Objects

A single object in the **Icon View** or **Diagram View** is selected by clicking it once with the left mouse button. When an object is selected, a green selection outline appears, and unless the class is read-only, selection handles appear as well. If the selection handles appear in red, this indicates that the object is inherited from another class and it is not possible to modify the object in any other way than by changing the values of its variables and parameters. To remove or modify other properties of an inherited object, open the class window of the class from which the object is inherited. Base classes are easily accessible by right-clicking on an empty area of the graphical view. In the popup menu that appears when right-clicking, choose **Open Base Class** and the name of the class to edit. Base classes are also accessible from the **Components** window; see Section 3.5 on page 76.

There is a total of nine selection handles, except for components whose class has the the preserve aspect ratio property set, in which case they have five selection handles.

Eight of the selection handles have the shape of a square and one the has the shape of a circle with a dot in it. The square shaped selection handles are used to resize an object, change the shape of the object, or rotate an object. The circular selection handle defines the origin point of the object, which is used as the center of rotation when an object is rotated, as well as the center of flip operations. The circular selection handle moves along with the object when moved or transformed. To move the circular selection handle independently of the object by using the mouse, place the mouse cursor on top of it and hold down the **Shift** key and left mouse button while moving the mouse.

To select multiple objects, click the pointer tool and then drag a rectangle around all the objects that you want to select, or hold down the **Shift** key and then click to select other objects one at a time. It is also possible to select all objects by choosing **Select All** from the **Edit** menu.

An entire selection can be canceled by clicking on an empty area of the view, or by pressing the **Esc** key. To cancel the selection of one object when several are selected, hold down the **Shift** key and click the object.

Adding Components

To add a component to a model, open the class window of the model and drag the component from the **Class Browser** to the **Diagram View** of the class window.

If the component is a connector it will also appear in the **Icon View** once it is dropped in the **Diagram View**. Connectors are visible in both the **Icon View** and the **Diagram View** unless the connector is protected. A protected connector is only visible in the **Diagram View**.

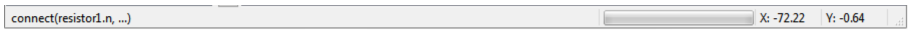
The initial size of the component is determined by the component icon settings of the class the component is representing. See "Changing the Initial Component Size for a Class" on page 69 for information on how to change this property.

Connecting Components

The **Connection Line Tool** on the **Graphic Tools** toolbar is used when connecting components.

Figure 3-28: The **Connection Line Tool** on the **Graphic Tools** toolbar.

To connect two components, select the **Connection Line Tool** and place the mouse cursor over a connector. When the cursor is close enough it will change in appearance and information about the connection will be displayed in the status bar.

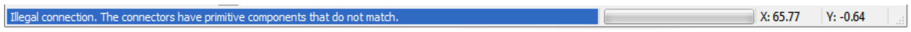


connect(resistor1.n, ...) X: -72.22 Y: -0.64

Figure 3-29: Information in the status bar when connecting components.

Click and hold down the left mouse button, move the cursor to the other connector, and release the mouse button to complete the connection. The connection line will be drawn as a right-angle connection line. Note that once the cursor gets close enough to the connector and the shape of the mouse cursor changes, the information in the status bar is also updated to show the name of the second connector in the connect equation.

Would the connection be illegal, the shape of the mouse cursor will change to indicate this and the connection will not be possible to complete. Information is also displayed in the status bar.



Illegal connection. The connectors have primitive components that do not match. X: 65.77 Y: -0.64

Figure 3-30: The status bar when attempting to create an illegal connection.

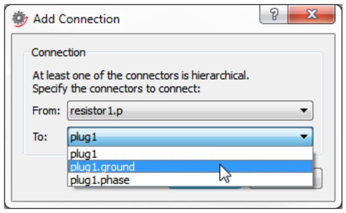
Connection lines, when created, are given a color matching the border color of the source connector. To change this behavior, see "Changing the Default Color of Connection Lines" on page 71. If you have models with existing connection lines that you wish to color according to their respective source connector, you can do this by selecting the connection lines and choosing **Color Connection Line** from the **Shape** menu.

Creating customized connections with multiple segments is just as simple, but instead of releasing the mouse button when the mouse cursor is over a connector, release it when the mouse cursor is anywhere else. Click the mouse button for each new line segment you want to create and complete the connection by clicking the mouse button while the mouse cursor is over a connector. To turn a customized connection line into a standard connection line, select the line and choose **Reset Connection Line** from the **Shape** menu, or right-click the connection line and choose **Reset Connection Line** from the popup menu. Another alternative is to double-click the connection line.

If you connect two components, *Model Center* makes sure the components remain connected when you rearrange the components.

A connection line can be turned into a curved connection line by right-clicking on it and choosing **Curved Connection Line** from the popup menu.

In the case that one of the connectors has hierarchical connectors, a dialog box for selecting what connectors to connect is displayed.


Add Connection

Connection

At least one of the connectors is hierarchical.
Specify the connectors to connect:

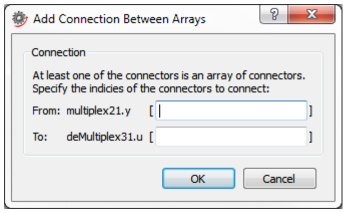
From: resistor1.p

To: plug1

Figure 3-31: The **Add Connection** dialog box.

In the figure above, a connection from the `resistor1.p` connector to the `plug1.ground` connector is created. The plug connector has several nested connectors, all of which are displayed in the drop-down list.

A connector visualized in the **Diagram View** can also be an array of connectors. When a connection is created between arrays of connectors and the *Model Center* is not able to deduce what connectors to connect or if there are several possibilities, a dialog box for selecting the indices of the connector arrays is displayed.


Add Connection Between Arrays

Connection

At least one of the connectors is an array of connectors.
Specify the indicies of the connectors to connect:

From: multiplex21.y [ ]

To: deMultiplex31.u [ ]

Figure 3-32: The **Add Connection Between Arrays** dialog box.

One example of when the dialog box is not shown even though one of connectors is an array of connectors is when the array is one-dimensional with only one element. In this case the model editor is able to deduce that the only possible connection is to the single element without any interaction with the user.

If both of the connectors are arrays of connectors and of the same dimension and size, a dialog box with a **Connect All** option will be shown. The **Connect All** option, if checked, will make a connection between the arrays' respective connectors. As can be seen when checking the option, a colon is used as the array index to describe such a connection in Modelica.

Adding Graphic Items

Graphic items can be added using the **Graphic Tools** toolbar. There are six types of graphic items: line, rectangle, ellipse, polygon, text, and bitmap. Ellipse items can also be visualized as arcs and pies by editing their properties after they have been added. For more information on how to edit the properties of graphic items, see "Editing the Properties of Graphic Items" on page 63.

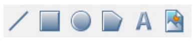


Figure 3-33: The **Graphic Tools** toolbar.

For instance, to draw a line, select the line tool and place the mouse cursor at the point where you want the line to begin. Click the left mouse button and hold it down while moving the mouse cursor to the point where you want the first segment of the line to end and release the mouse button. If you want a line consisting of a single segment only, click the left mouse button once without moving the mouse cursor. If you want to add more segments to the line, move the mouse cursor to the point where you want the segment to end and press the mouse button. When you no longer want to add more segments, click on the left mouse button once again without moving the mouse cursor.

At any time during the process of adding the line, you can press the right mouse button to cancel the operation.

Line items and polygon items have a smooth attribute to give them smooth edges. This attribute can be set already at drawing time by holding down the **Shift** key when creating the first point of the line or polygon. The **Shift** key can be released once the first point has been specified.

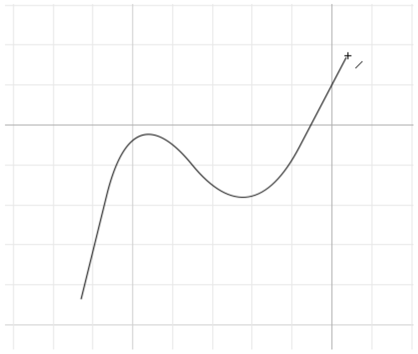


Figure 3-34: Drawing a line with smooth edges by holding down the **Shift** key.

Holding down the **Shift** key while creating a rectangle, ellipse, or text will restrict the new shape to equal length sides.

Removing Objects

Select the objects to remove and choose **Edit ▸ Delete**. This will remove the object from the model. If a component with connections to other components is removed, the connections in the class in which the component is declared will be removed as well.

Note that classes extending the class in which the removed component was declared are not modified in any way. Connections to or from an inherited component that is removed in its base class become unresolvable.

Transforming Components into a New Class

A model with a large amount of components is often hard to overview and understand. Such models benefit from being divided up into one or more hierarchical levels by encapsulating groups of components into new components. This is tedious work to do manually but quick and straightforward to do in *Model Center*.

Start by identifying groups of components that by themselves describe some entity that is easily understood. Continue by selecting all the components in one of these groups and choose **Transform into New Class** from the **Edit** menu.

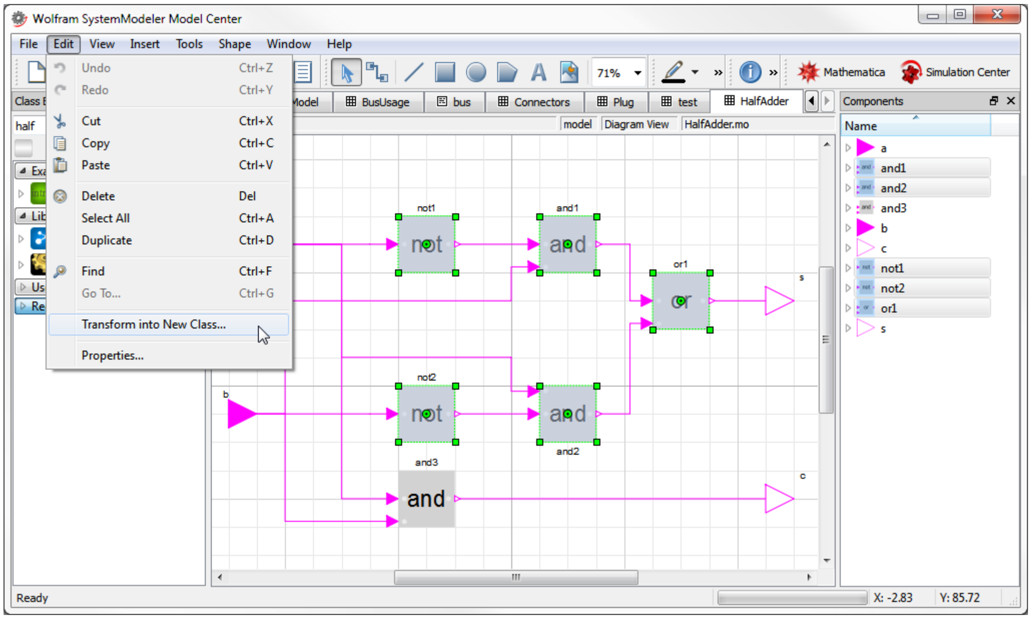


Figure 3-35: Transforming a group of logical components into a new XOR class.

You will be asked to give the new class a name and description (optional), as well as to specify a location for the new class. Click the **OK** button to close the dialog box and start the transformation process. Once completed, the selected components will have been replaced by a single component, an instance of the class you named in the dialog box. If you open the class you will see that the class contains all of the replaced components along with a set of automatically created connectors that define the interface of the class. You may wish to edit the class to rearrange its connectors and to create some graphics for its icon.

Adding Inheritance Relationships

One way to add an inheritance relationship to an existing class is to use the **Class Properties** dialog box and add the class to the list of extended classes. Another way, which is more convenient, is to simply drag the class to extend from the **Class Browser**, using the right mouse button, and drop it in one of the graphical views of the class. When the right mouse button is released, a popup menu will appear in which you may choose to extend the class.

Renaming Components

A component can be renamed by right-clicking it, choosing **Rename** from the popup menu, and editing the name in the **Component Properties** dialog box. It is also possible to rename a component by selecting it and pressing the **F2** key.

When renaming a component from the **Component Properties** dialog box, all references to the renamed component, for instance in connection equations, will be updated automatically in all currently loaded classes. When the renaming operation is completed, a list of all classes that were modified due to references to the renamed component is shown.

Duplicating Objects

At times you may wish to create several objects with identical or nearly identical properties. This can be a tedious task if the object is a complex shape or a component with many parameters. Instead of creating and customizing each object individually, it is only necessary to create one of the objects and then duplicate it as many times as desired. To duplicate an object, select it and choose **Duplicate** from the **Edit** menu or right-click an object and choose **Duplicate** from the popup menu.

Copying Graphic Items

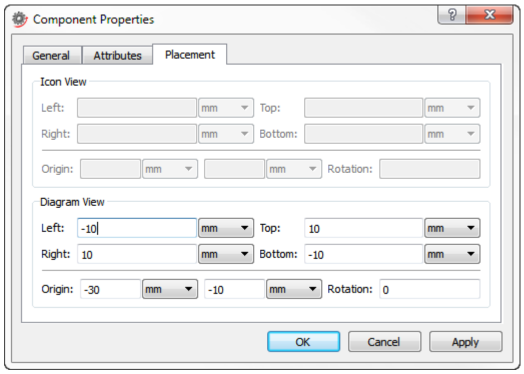
Graphic items can easily be copied from one view to another. Begin with selecting the graphic items you want to copy. Once the items are selected, choose **Copy** from the **Edit** menu, or right-click an item and choose **Copy** from the popup menu. Switch to the class and view where you want the items to be copied to and choose **Edit ▸ Paste**, or right-click anywhere on the view and choose **Paste** from the popup menu. You can paste the items any number of times and to any number of views.

Moving Objects within the View

Select the objects to be moved and point the mouse cursor toward one of them. Once in position, the mouse cursor will change to a four-headed arrow. Now drag the objects to where you want them. Be careful not to point to a selection handle when you move an object as this will resize the object instead.

An object can also be moved by editing its properties in a dialog box. Right-click an object and choose **Properties** from the menu.

If the object is a component or connection line, switch to the **Placement** view of the properties dialog box. To specify a new position for the object, edit the value of the **Origin** attribute. Note that for connector components you will be able to edit the position in both the **Icon View** and the **Diagram View** as connectors are visualized in both of these views.



Icon View

Left:		mm	Top:		mm
Right:		mm	Bottom:		mm
Origin:	mm	mm	Rotation:		

Diagram View

Left:	-10	mm	Top:	10	mm
Right:	10	mm	Bottom:	-10	mm
Origin:	-30 mm	-10 mm	Rotation:	0	

Figure 3-36: The **Placement** view of the **Component Properties** dialog box.

Resizing Objects

Objects in the graphical views are resized using their selection handles. Select the object to resize and drag one of its square selection handles until the object is the desired size. Rectangular graphic items (rectangles, ellipses, text, and bitmaps) and components can be resized proportionally by dragging one of their corner handles.

Depending on the preserve aspect ratio setting of the class a component represents, a component may only have five selection handles (four corner handles and one center handle) of the nine selection handles visible, restricting the resizing of the component to only proportional resizing. See "Changing the Initial Component Size for a Class" on page 69 for more information about the preserve aspect ratio property.

An object can also be resized by editing its properties in a dialog box. Right-click an object and choose **Properties** from the menu.

If the object is a component or connection line, click the **Placement** tab in the properties dialog box. To resize the object, edit the **Left**, **Top**, **Right**, and **Bottom** attributes, or the points that define the item if the object is a line or polygon item. For connector components you will be able to edit the size in both the **Icon View** and the **Diagram View** as connectors are visualized in both of these views.

If the object is a graphic item, edit the points that define the item in the properties dialog box to resize the item.

Flipping Objects

Most objects in the **Icon View** and **Diagram View** can be flipped horizontally or vertically by selecting the object and choosing **Shape ▸ Rotate or Flip ▸ Flip Horizontal**, or **Shape ▸ Rotate or Flip ▸ Flip Vertical**.

Objects can also be flipped using the **Flip Horizontal** and **Flip Vertical** tools found on the **Layout** toolbar.

Figure 3-37: The **Flip Horizontal** and **Flip Vertical** buttons on the **Layout** toolbar.

An object is always flipped around its origin point, which is the circular selection handle visible when the object is selected. To move the origin point independently of the object, place the mouse cursor on top of the selection handle and hold down the **Shift** key and the left mouse button while moving the mouse. The origin point can also be specified in the properties dialog box of the object.

Rotating Objects

To rotate an object 90 degrees counterclockwise, choose **Shape ▸ Rotate or Flip ▸ Rotate Left**, or use the **Rotate Left** tool on the **Layout** toolbar. To rotate an object 90 degrees clockwise, choose **Shape ▸ Rotate or Flip ▸ Rotate Right**, or use the **Rotate Right** tool on the **Layout** toolbar.

Figure 3-38: The **Rotate Left** and **Rotate Right** buttons.

All objects can also be rotated freely using their selection handles. Select the object and drag one of its square selection handles while holding down the **Shift** key.

Finally the rotation of an object can also be specified by editing the rotation attribute in the properties dialog box. For components and connection lines, the rotation attribute can be found in the **Placement** view of the properties dialog.

The rotation of an object is always performed around its origin point, which is the circular selection handle visible when the object is selected. To move the origin point independently of the object, place the mouse cursor on top of the selection handle and hold down the **Shift** key and the left mouse button while moving the mouse. The origin point can also be specified in the properties dialog box of the object.

Changing Stacking Order of Graphic Items

The stacking order of graphic items is the order in which graphic items overlap other graphic items in the graphical views. The first item added to a graphical view will be at the back of the stack and the item most recently added will be at the front.

To change a graphic item's position in the stacking order, select the item and choose one of the commands available from the **Shape ▸ Order** menu:

- Choose **Bring to Front** to bring the graphic item to the front.
- Choose **Send to Back** to send the graphic item to the back.
- Choose **Bring Forward** to bring the graphic item forward one level.
- Choose **Send Backward** to send the graphic item backward one level.

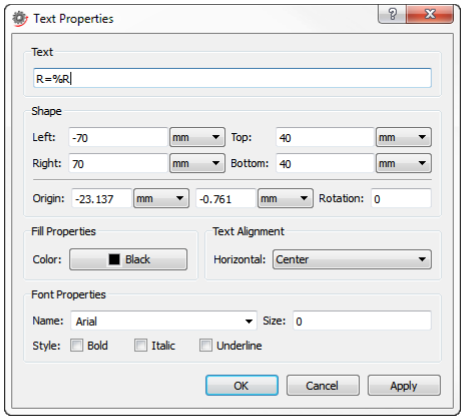
The **Bring to Front** and **Bring to Back** operations are also available on the toolbar.

Figure 3-39: The **Bring to Front** and **Bring to Back** buttons on the **Layout** toolbar.

Note that the stacking order applies to graphic items only and not to components or connections. Components are always drawn on top of all other graphic items and connections are drawn on top of both graphic items and components.

Showing Attributes of a Component in Its Icon

The name, description, and parameter values of a component can be shown in its icon by adding a **Text** item in the **Icon View** of the component's class window. To show the name of the component, set the text of the **Text** item to `"%name"`. The description of the component can be shown by setting text to "`%comment`", and any of the component's parameter values are shown by setting the text to "`%`" followed by the parameter name.



Text: R=%R

Shape
Left: -70 mm Top: 40 mm
Right: 70 mm Bottom: 40 mm
Origin: -23.137 mm -0.761 mm Rotation: 0

Fill Properties
Color: ■ Black

Text Alignment
Horizontal: Center

Font Properties
Name: Arial Size: 0
Style: ☐ Bold ☐ Italic ☐ Underline

Figure 3-40: Adding a **Text** item to show the value of the parameter `R` in a resistor model.

For information on how to add a **Text** item to a class, see "Adding Graphic Items" on page 52.

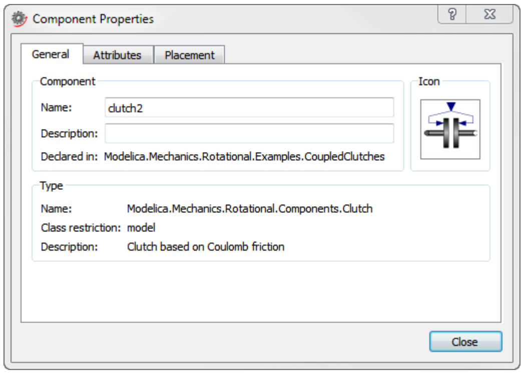
Editing the Properties of Components

To edit the properties of a component, open the properties dialog box for the component by right-clicking it and choosing **Properties** from the popup menu, or by selecting it and choosing **Properties** from the **Edit** menu.

The dialog box has three views:

- **General**. Provides the possibility of renaming the component and changing its description. This view also contains type information about the component.
- **Attributes**. This view lets you edit the attributes of the component.

- **Placement**. If the component is visible in one of the graphical views, it has a placement annotation describing its placement and transformations in that view. This annotation can be edited directly from the **Placement** view of the properties dialog box.



Component
Name: clutch2
Description:
Declared in: Modelica.Mechanics.Rotational.Examples.CoupledClutches
Icon

Type
Name: Modelica.Mechanics.Rotational.Components.Clutch
Class restriction: model
Description: Clutch based on Coulomb friction

Figure 3-41: The **General** view of the **Component Properties** dialog box.

The sections of each view in the dialog box are described in detail below.

- **Component**, **General** view
  Specifies the name and description of the component.
  - **Name**. The name of the component.
  - **Description** (optional). A short description of the component.
  - **Declared in**. The name of the class in which the component is declared.
  - **Redeclared in**. The name of the class in which the component is redeclared. Available only if the component is replaceable and has been redeclared.
- **Type**, **General** view
  Describes the type of the component.
  - **Name**. The name of the component's declared type.
  - **Class restriction**. The class restriction of the component's declared type. Available only if the type of the component is a class.
  - **Description**. The description of the component's declared type.
- **Properties**, **Attributes** view
  Specifies the properties of the component.
  - **Final**. Prevents modifications of the component.

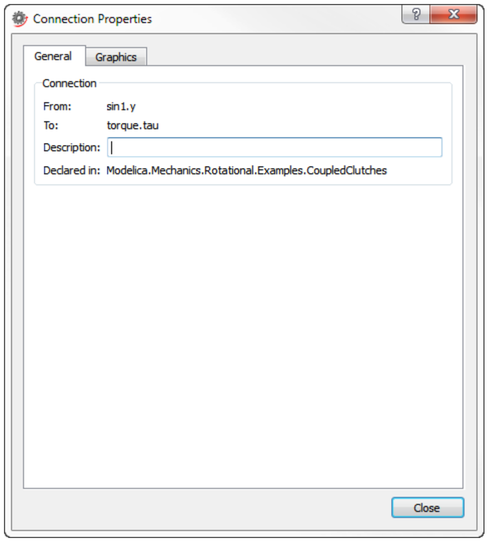
  - **Flow**. Applies only to components declared in connector classes. The flow property specifies whether the sum of the quantity should sum to zero when connected. Select this checkbox for components specifying a flow quantity such as current, fluid, force, etc. Clear this checkbox for components specifying potential (non-flow) quantities, such as voltage, pressure, position, etc.
  - **Protected**. Access to the component is restricted to the class in which it is declared, and to classes inheriting that class. Clear this checkbox to make the component accessible from all classes.
  - **Replaceable**. Allows modifications of the component in the form of redeclarations. See Section 3.5.7 on page 80 for information on how to change the type of a component by using redeclaration modifications.

- **Variability**, **Attributes** view
  Specifies the variability of the component, with respect to simulation time.
  - **Unspecified** (default). Continuous for variables of type `Real`. Discrete for components of type `Boolean`, `Integer`, `String`, and `enumeration`.
  - **Discrete**. Discrete components are piecewise constant with respect to simulation time and may only be changed by events during simulation.
  - **Constant**. Constants can never change once defined.
  - **Parameter**. Parameters may change before simulation, but remain constant during simulation.
- **Causality**, **Attributes** view
  Specifies the causality of the component. The causality of a component does not have to be explicitly specified unless the component is an input or output to a block or function.
  - **None** (default). No explicitly specified causality.
  - **Input**. The component is an input to a class.
  - **Output**. The component is an output to a class.
- **Dynamic Reference**, **Attributes** view
  Specifies the dynamic reference property of the component.
  - **Inner**. Enable access to the component in nested scopes.
  - **Outer**. Make the component reference an inner component with the same name in an enclosing scope.

- **Icon View**, **Placement** view
  Specifies the position, size, and rotation of connector components in the **Icon View**. This section is disabled for non-connector components as only connector components are visualized in **Icon View**.
- **Diagram View**, **Placement** view
  Specifies the position, size, and rotation of components in the **Diagram View**.

Editing the Properties of Connections

The properties of a connection can be edited by right-clicking the connection and choosing **Properties** from the popup menu, or by selecting the connection and choosing **Edit ▸ Properties**.



From: sin1.y
To: torque.tau
Description:
Declared in: Modelica.Mechanics.Rotational.Examples.CoupledClutches

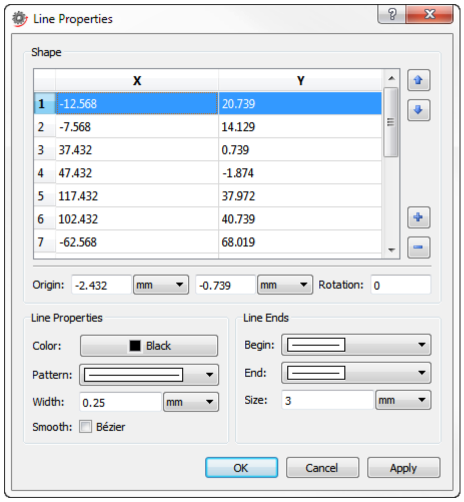
Figure 3-42: The **General** view of the **Connection Properties** dialog box.

The properties dialog box for connections has two views, a general view and a graphics view. The general view provides you with information about the connectors of the connection and the connection's declaration class. It also lets you edit the description of the connection. The graphics view lets you edit the appearance and placement of the connection in the **Diagram View**.

The color of a connection line can either be specified in the **Connection Properties** dialog box or be derived from the source connector of the connection. To color one or more connection lines using the border color of their respective source connectors, select the lines and choose **Color Connection Line** from the **Shape** menu, or right-click one of the lines and choose **Color Connection Line** from the popup menu.

Editing the Properties of Graphic Items

When a graphic item is created, it is automatically given default values for all its attributes. For instance, the border color of an ellipse is always set to black. All attributes of a graphic item can be edited by selecting the graphic item and choosing **Properties** from the **Edit** menu, or by right-clicking the item and choosing **Properties** from the popup menu. A third option to bring up the properties dialog box is to double-click the graphic item.



Shape

	X	Y
1	-12.568	20.739
2	-7.568	14.129
3	37.432	0.739
4	47.432	-1.874
5	117.432	37.972
6	102.432	40.739
7	-62.568	68.019

Origin: -2.432 mm -0.739 mm Rotation: 0

Line Properties

Color: Black

Pattern:

Width: 0.25 mm

Smooth: Bézier

Line Ends

Begin:

End:

Size: 3 mm

Figure 3-43: The **Line Properties** dialog box.

Most of the common properties of graphic items can also be edited by using the tools on the formatting toolbar.

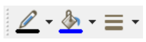


Figure 3-44: Tools to update common properties of graphic items.

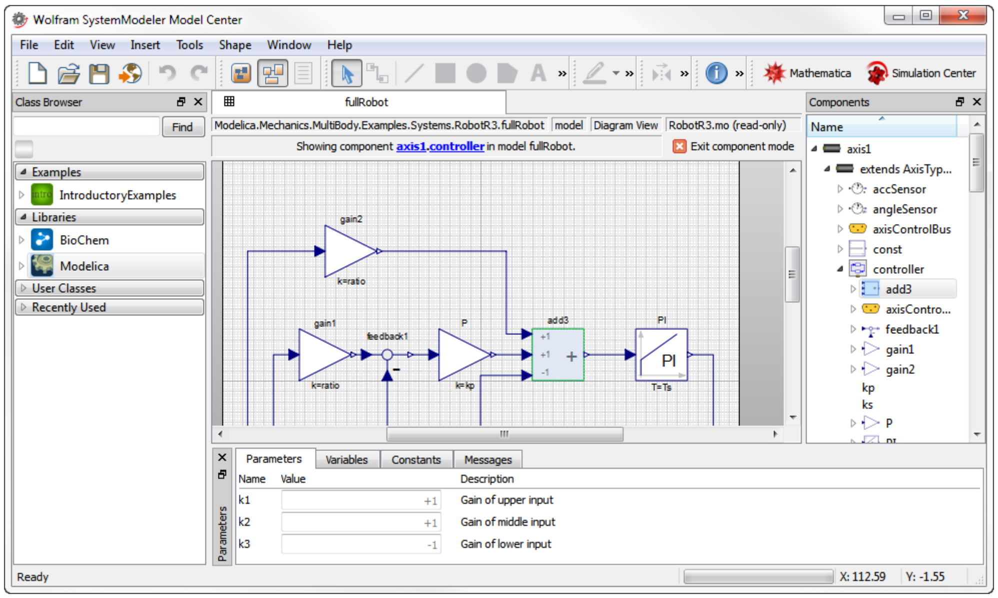
The tools operate on the selected graphic items. Configure the tools by clicking on their arrows. Any selected items are affected immediately by any changes made to the tools. Apply the current setting of a tool to the selected graphic items by clicking the tool's icon.

Showing Components

One step involved when configuring components in a hierarchical class is the process of identifying the components. You may use the component browser's tree structure to expand the internals of a component, but the presentation of the internals, a list of components names, makes the identification hard as you need to know the name of the component.

Identification is a lot easier by viewing the internals of a component graphically. This is possible by right-clicking a component and selecting **Open Component** from the popup menu. Double-clicking on a component gives the same result. The class window will enter component mode, where the component is graphically visualized in the context of the class. The component mode makes it possible to visualize the internals of components at any level in the component hierarchy of a class.

In component mode you can select components in the graphical views and configure them using the **Parameters**, **Variables**, and **Constants** views. All changes made will be stored as a component modification in the top-level class. Keep in mind that as you are visualizing a component, and not a class, no changes can be made in the graphical views.



Showing component axis1.controller in model fullRobot.

Name	Value	Description
k1	+1	Gain of upper input
k2	+1	Gain of middle input
k3	-1	Gain of lower input

Ready X: 112.59 Y: -1.55

Figure 3-45: Showing the internals of component `axis1.controller` in class `Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot`.

When in component mode, the name of the visualized component is shown at the top of the class window, along with a button that will let you exit component mode and return to the graphical view of the class. If a hierarchical component is visualized, such as in Figure 3-45, you can step up in the component hierarchy by clicking the hierarchical parts of the component name.

Editing Base Classes

If a class extends one or several other classes, you may want to edit one of them. Instead of finding the correct class in the **Class Browser**, you can right-click an empty location in the graphical view and choose **Open Base Class** followed by the name of the class to edit from the popup menu.

The **Components** window can also be used to open base classes; for more information on how to use the **Components** window, see Section 3.5 on page 76.

Zooming In and Out

When working with a detailed icon or diagram, you may want a closer view of certain objects to make it easier to work with them.

To quickly zoom in or out on any icon or diagram, do any of the following:

- Press **Ctrl** + **Shift** and click the location that you want to see more closely.
- Press **Ctrl** + **Shift** and right-click a location that you want to see from farther away.

If you have a mouse with a wheel, zoom in and out by holding down **Ctrl** as you rotate the wheel forward or backward.

To specify a certain zoom level, choose **View ▸ Zoom** and select one of the predefined levels, or click the zoom toolbar box to type your own zoom level. If you enter a zoom level of your own, press **Return** or **Enter** to confirm the input.

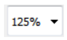


Figure 3-46: The zoom box on the **Graphic Tools** toolbar.

Panning and Scrolling

It is possible to pan any icon or model within a graphical view by using the mouse. Press and hold the **Ctrl** key. The mouse cursor will change into an open hand. Click and hold the left mouse button and drag the model to the desired location.

If you have a mouse with a wheel, you can scroll up and down within a graphical view by rotating the wheel forward or backward. To scroll left and right, hold the **Shift** key while rotating the wheel.

Grid

A grid helps you position objects visually in the **Icon View** and **Diagram View** of a class, and can also be used to snap objects; see "Snap" on page 66.

The grid spacing can be set individually for the **Icon View** and **Diagram View** for every class, and is saved along with the class definition when the class is saved.

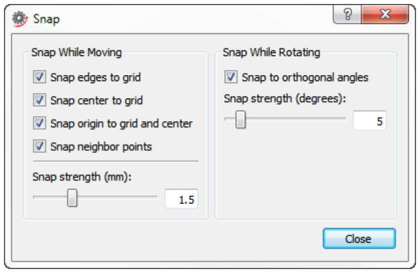
To edit the grid spacing for the **Icon View** or **Diagram View** of the active class window, choose **Page Setup** from the **File** menu, or right-click on an empty part of the graphical view and choose **Page Setup**. In the page setup dialog box, select either the **Icon Layer** view or the **Diagram Layer** view and edit the values for the horizontal and vertical spacing in the **Grid** section. To edit the grid spacing for the visible view of the active class window, choose **Grid** from the **Tools** menu. Here you can also choose to draw the grid on top of all objects, making it easier to align objects.

The grid can be toggled on and off by choosing **View ▸ Grid**.

See "Changing Default Page Properties for Graphical Layers" on page 70 for information on how to change default grid settings for the graphical layers.

Snap

The snap function helps you position and align objects. Snapping pulls objects to grid lines when objects are being moved, to orthogonal angles when objects are being rotated, and so on. You can control the type of snapping and the snap strength from the **Snap** dialog box, which can be reached from the **Tools** menu. Custom snap settings are applied to the visible view of the active class window and will remain active until you choose to exit the *Model Center*.



☑ Snap edges to grid
☑ Snap center to grid
☑ Snap origin to grid and center
☑ Snap neighbor points
Snap strength (mm): 1.5
☑ Snap to orthogonal angles
Snap strength (degrees): 5

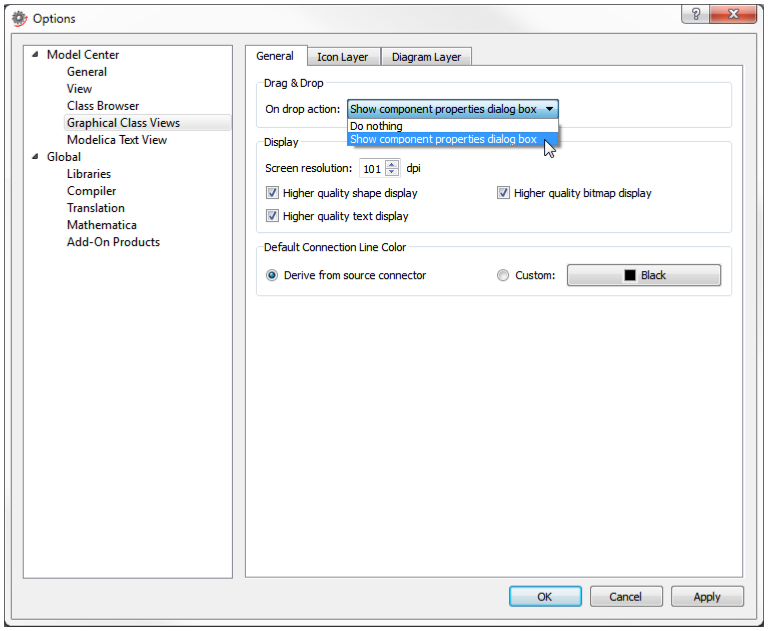
Figure 3-47: The **Snap** dialog box.

By default, all objects snap to grid lines when the objects are created or moved. The origin points of an object are by default also snapped to the grid as well as the center of the object. The points of a line or polygon object also snap to neighboring points and components are snapped to orthogonal angles while being rotated.

You may not always want to snap objects, for instance if you want an object to be rotated close to an orthogonal angle. To disable a certain type of snapping, uncheck it in the **Snap** dialog box.

On Drop Action

It is possible to specify an action to be performed when a class is dropped in any of the two graphical views of a class window while holding the **Ctrl** key. The default action is to show the **Component Properties** dialog box for the instance of classes dropped in a graphical view. To change this action, open the **Options** dialog box by choosing **Options** from the **Tools** menu, and go to the **Graphical Class Views** section. The on drop action applies to all class windows. The setting is automatically saved when closing the dialog and restored the next time *Model Center* is started.



On drop action: Show component properties dialog box
Do nothing
Show component properties dialog box

Screen resolution: 101 dpi
[✓] Higher quality shape display
[✓] Higher quality bitmap display
[✓] Higher quality text display

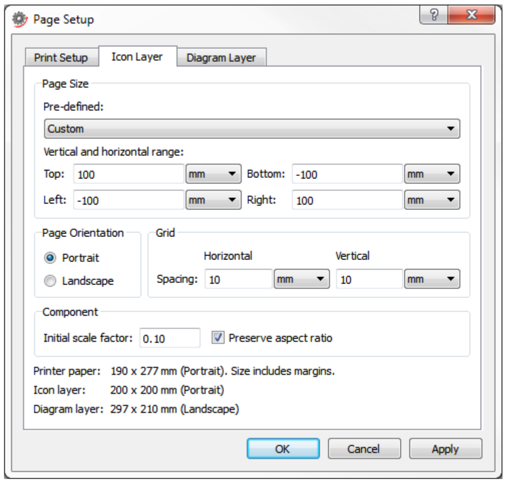
[● Derive from source connector] ○ Custom: ■ Black

Figure 3-48: Specifying an on drop action for the graphical views.

Resizing Graphical Layers

The size and coordinate system of the icon layer and diagram layer for a class can be set from the page setup dialog box.

To edit the coordinate systems of the class in the active class window, choose **Page Setup** from the **File** menu, or right-click on an empty part of the class window and choose **Page Setup** from the popup menu. Choose whether to edit the properties of the icon layer or diagram layer by clicking the **Icon Layer** or **Diagram Layer** tab. The settings are saved along with the class definition when the class is saved.


Page Setup

Print Setup | Icon Layer | Diagram Layer

Page Size

Pre-defined: Custom

Vertical and horizontal range:

Top: 100 mm Bottom: -100 mm

Left: -100 mm Right: 100 mm

Page Orientation: [✓ Portrait] Landscape

Grid: Horizontal Spacing: 10 mm; Vertical Spacing: 10 mm

Component

Initial scale factor: 0.10 [✓] Preserve aspect ratio

Printer paper: 190 x 277 mm (Portrait). Size includes margins.

Icon layer: 200 x 200 mm (Portrait)

Diagram layer: 297 x 210 mm (Landscape)

Figure 3-49: Editing the icon layer properties of a class in the **Page Setup** dialog box.

For all classes except connectors, the default the size of the diagram layer is 297 x 210 mm, which corresponds to the size of a landscape A4 sheet, and the size of the icon layer is 200 x 200 mm. For connector classes the default size of the diagram layer is 200 x 200 mm as a square is the most common shape of a connector. Also, by default the coordinate systems of both layers have their origin (0,0) at the center of the page.

See "Changing Default Page Properties for Graphical Layers" on page 70 for information on how to change the default page properties of the graphical layers.

Changing the Initial Component Size for a Class

The initial size of the component of a class when dropped in the **Icon View** or **Diagram View** can be specified in the **Page Setup** dialog box. For all classes except connectors, the icon layer of the class is used to represent the class when it appears as a component in the **Diagram View** of another model. For connector classes, that can appear in both **Icon Views** and **Diagram Views** of other models; the icon layer is used to represent it on **Icon Views**, while the diagram layer is used to represent it in **Diagram Views**.

To edit the property of the class in the active class window, choose **Page Setup** from the **File** menu, or right-click on an empty part of the class window and choose **Page Setup** from the popup menu. Choose whether to edit the property of the icon layer or diagram layer (available for connector classes only) by selecting the **Icon Layer** or **Diagram Layer** tab. The initial component icon size for the chosen graphical layer is specified by a scale factor in the **Component** section of the dialog box. The initial size of a component icon is the scale factor multiplied with the width and height of the graphical layer.

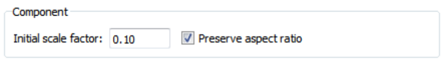

Component
Initial scale factor: 0.10
[✓ Preserve aspect ratio]

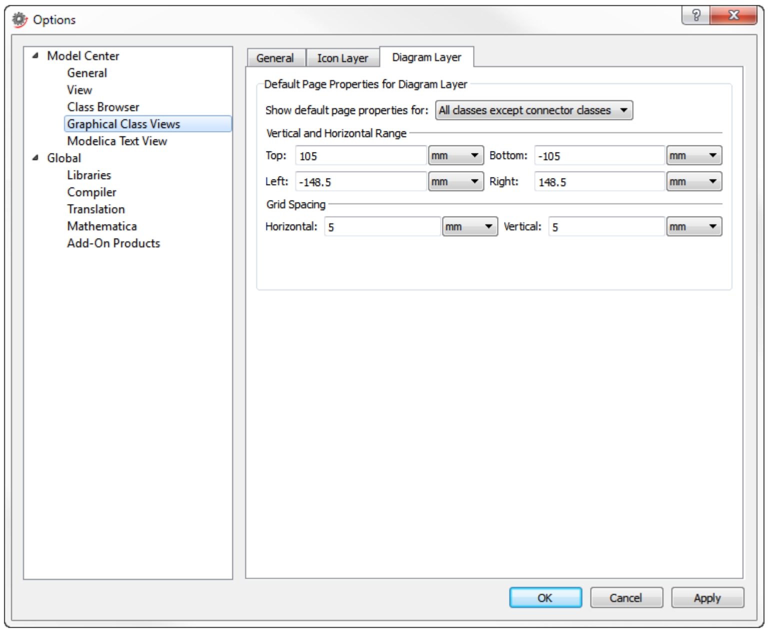
Figure 3-50: The **Component** section of the **Page Setup** dialog box.

It is also possible to preserve the aspect ratio of component icons representing the chosen layer of the class by checking the **Preserve aspect ratio** box. The settings are saved along with the class definition when the class is saved.

See "Changing Default Page Properties for Graphical Layers" on page 70 for information on how to change the default page properties of the graphical layers.

Changing Default Page Properties for Graphical Layers

The default page properties for the icon layer and diagram layer can be changed in the **Graphical Class Views** section of the **Options** dialog box. To open the dialog box, choose **Options** from the **Tools** menu.



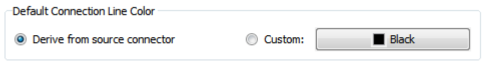
Default Page Properties for Diagram Layer
Show default page properties for: All classes except connector classes
Vertical and Horizontal Range
Top: 105 mm Bottom: -105 mm
Left: -148.5 mm Right: 148.5 mm
Grid Spacing
Horizontal: 5 mm Vertical: 5 mm

Figure 3-51: Editing the default diagram layer properties for connector classes.

The default page properties are used when new Modelica classes are created. Changing the default page properties does not affect existing classes. The default page properties are automatically saved when closing the dialog box and restored the next time *Model Center* is started.

Changing the Default Color of Connection Lines

When a connection line is created, its initial color is determined by a setting available in the **Graphical Class Views** section of the **Options** dialog box. To open the **Options** dialog box, choose **Options** from the **Tools** menu.



Default Connection Line Color

[✓ Derive from source connector] Custom: ■ Black

Figure 3-52: The **Default Connection Line Color** section of the **Options** dialog box.

By default, the initial color of a connection line is derived from the source connector of the connection.

Printing

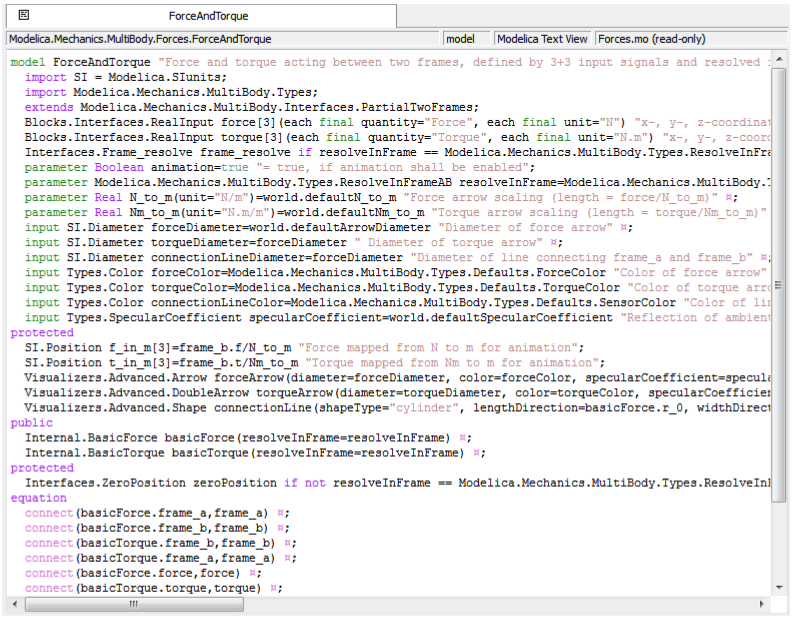
You can print the contents of the visible graphical view by choosing **File ▸ Print**. The contents of the view will automatically be scaled to fit the selected paper size if necessary.

Copy View as Image

The contents of the graphical view can be copied to the clipboard as an image by right-clicking anywhere in the view and choosing **Copy View as Image**. The image is a representation of the graphical view in its current state, with the current selection and zoom level.

3.4 Modelica Text View

The **Modelica Text View** is a text editor where the textual representation (Modelica definition) of the class can be viewed and edited. The **Modelica Text View** allows you to access and edit all attributes of a class. In most situations the graphical views are easier to work with and provide a better overview of the class. However, the graphical views do not let you edit all aspects of a class. For instance, to edit the equations of a class, the **Modelica Text View** is used.



```
model ForceAndTorque "Force and torque acting between two frames, defined by 3+3 input signals and resolved :
  import SI = Modelica.SIunits;
  import Modelica.Mechanics.MultiBody.Types;
  extends Modelica.Mechanics.MultiBody.Interfaces.PartialTwoFrames;
  Blocks.Interfaces.RealInput force[3](each final quantity="Force", each final unit="N") "x-, y-, z-coordinat
  Blocks.Interfaces.RealInput torque[3](each final quantity="Torque", each final unit="N.m") "x-, y-, z-coorc
  Interfaces.Frame_resolve frame_resolve if resolveInFrame == Modelica.Mechanics.MultiBody.Types.ResolveInFra
  parameter Boolean animation=true "= true, if animation shall be enabled";
  parameter Modelica.Mechanics.MultiBody.Types.ResolveInFrameAB resolveInFrame=Modelica.Mechanics.MultiBody.T
  parameter Real N_to_m(unit="N/m")=world.defaultN_to_m "Force arrow scaling (length = force/N_to_m)" ¤;
  parameter Real Nm_to_m(unit="N.m/m")=world.defaultNm_to_m "Torque arrow scaling (length = torque/Nm_to_m)"
  input SI.Diameter forceDiameter=world.defaultArrowDiameter "Diameter of force arrow" ¤;
  input SI.Diameter torqueDiameter=forceDiameter " Diameter of torque arrow" ¤;
  input SI.Diameter connectionLineDiameter=forceDiameter "Diameter of line connecting frame_a and frame_b" ¤;
  input Types.Color forceColor=Modelica.Mechanics.MultiBody.Types.Defaults.ForceColor "Color of force arrow"
  input Types.Color torqueColor=Modelica.Mechanics.MultiBody.Types.Defaults.TorqueColor "Color of torque arrc
  input Types.Color connectionLineColor=Modelica.Mechanics.MultiBody.Types.Defaults.SensorColor "Color of lir
  input Types.SpecularCoefficient specularCoefficient=world.defaultSpecularCoefficient "Reflection of ambient
protected
  SI.Position f_in_m[3]=frame_b.f/N_to_m "Force mapped from N to m for animation";
  SI.Position t_in_m[3]=frame_b.t/Nm_to_m "Torque mapped from Nm to m for animation";
  Visualizers.Advanced.Arrow forceArrow(diameter=forceDiameter, color=forceColor, specularCoefficient=specula
  Visualizers.Advanced.DoubleArrow torqueArrow(diameter=torqueDiameter, color=torqueColor, specularCoefficier
  Visualizers.Advanced.Shape connectionLine(shapeType="cylinder", lengthDirection=basicForce.r_0, widthDirect
public
  Internal.BasicForce basicForce(resolveInFrame=resolveInFrame) ¤;
  Internal.BasicTorque basicTorque(resolveInFrame=resolveInFrame) ¤;
protected
  Interfaces.ZeroPosition zeroPosition if not resolveInFrame == Modelica.Mechanics.MultiBody.Types.ResolveInI
equation
  connect(basicForce.frame_a,frame_a) ¤;
  connect(basicForce.frame_b,frame_b) ¤;
  connect(basicTorque.frame_b,frame_b) ¤;
  connect(basicTorque.frame_a,frame_a) ¤;
  connect(basicForce.force,force) ¤;
  connect(basicTorque.torque,torque) ¤;
```

Figure 3-53: The textual representation of the `ForceAndTorque` model in `Modelica.Mechanics.MultiBody.Forces`.

Any changes made to the class in the **Modelica Text View** are applied when saving the class or when switching to a view that accesses the Modelica definition of the class. The changes can also be applied manually by pressing **Shift** + **Enter**.

The Modelica class definition is checked for syntax errors when applying changes in the **Modelica Text View**. Syntax errors will appear as clickable links in the **Messages** view. Clicking a link will move the text cursor to the origin of the error. It is important that the syntax errors are corrected before editing the class (or any other class) in any other view or you risk losing the parts of the class definition with syntax errors.

The position (line and column) of the text cursor is visible in the status bar when the **Modelica Text View** is active.

Syntax Highlighting

To make Modelica code easier to read and write, the text editor highlights elements of the Modelica language in different colors. The highlighting is performed in real time while typing or modifying text in the text editor. The colors used by the highlighter can be customized in the **Options** dialog box; see "Customizing the Modelica Text View" on page 74.

To further assist you in reading and writing Modelica code, the editor will highlight matching parentheses, braces, or brackets. Whenever you type one of these parenthesis, or place the text cursor next to an existing parenthesis, the editor will attempt to match it. If a matching parenthesis is found, the parenthesis, as well as the text in between, will be highlighted.

Toggling Annotation Visibility

To increase readability, each annotation is by default collapsed to a single character, ¤, in the text editor. Annotations can be expanded or collapsed by right-clicking anywhere within the **Modelica Text View** and choosing **Expand Annotations** or **Collapse Annotations** from the popup menu. The default behavior can be specified in the **Options** dialog box; see "Customizing the Modelica Text View" on page 74.

Finding and Replacing Text

The find and replace toolbar allows you to search in the **Modelica Text View** of a class window and to make text substitutions if you wish. To open the find and replace toolbar, choose **Edit ▸ Find**.

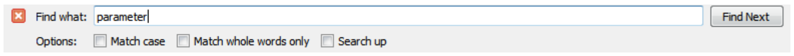

Find what: parameter

Figure 3-54: Searching for parameters in the **Modelica Text View**.

Enter the text you intend to search for in the **Find what** text field. The search is performed while you are typing, and if the text is found it will be selected in the document. Click the **Find Next** button or press **Enter** to find the next occurrence of the text. The search automatically continues past the end or start of the document into the unsearched portion. If the text is not found at all, the background color of the text field will turn red.

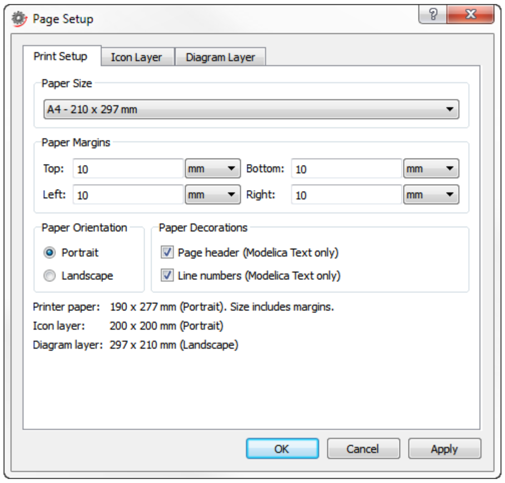
Clicking the **Replace** button will substitute the first occurrence of the text you search for with the text entered in the **Replace with** text field. To replace all occurrences of the text, click the **Replace All** button.

By default the search is performed from the insertion point and forward in the document. To search backward from the insertion point, select the **Search up** checkbox. Check the **Match case** checkbox to restrict the search to look only for occurrences that match the uppercase and lowercase characters you enter. You can also choose to search only for whole words, rather than matching the text as it occurs within words, by checking the **Match whole words only** box.

Close the find and replace toolbar by clicking the close button or the **Esc** key.

Printing

You can print the contents of the currently active **Modelica Text View** by choosing **File ▸ Print**. By default a page header including the name of the printed class and the page number is inserted on every page. Also, to the left of each line of code a line number is inserted. To turn off the header or line numbers, choose **File ▸ Page Setup** in order to open the **Page Setup** dialog box. In the **Print Setup** tab, clear the checked boxes in the **Paper Decorations** section.



Paper Size: A4 - 210 x 297 mm

Paper Margins
Top: 10 mm Bottom: 10 mm
Left: 10 mm Right: 10 mm

Paper Orientation: [✓ Portrait] Landscape

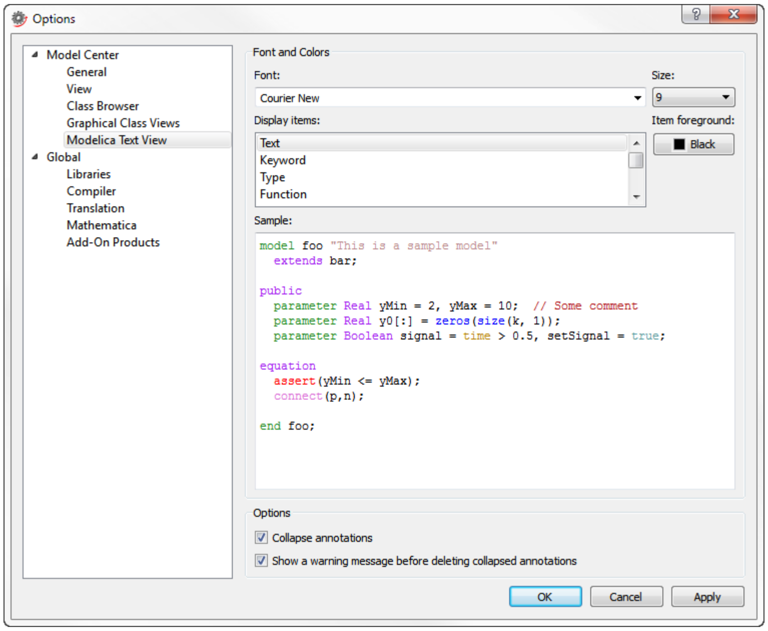
Paper Decorations
☑ Page header (Modelica Text only)
☑ Line numbers (Modelica Text only)

Printer paper: 190 x 277 mm (Portrait). Size includes margins.
Icon layer: 200 x 200 mm (Portrait)
Diagram layer: 297 x 210 mm (Landscape)

Figure 3-55: The **Print Setup** view of the **Page Setup** dialog box.

Customizing the Modelica Text View

The text editor of the **Modelica Text View** may be customized in the **Options** dialog box. To open the dialog box, choose **Options** from the **Tools** menu.


Options

- Model Center
  - General
  - View
  - Class Browser
  - Graphical Class Views
  - Modelica Text View
- Global
  - Libraries
  - Compiler
  - Translation
  - Mathematica
  - Add-On Products

Font and Colors

Font: Courier New

Size: 9

Display items: Text, Keyword, Type, Function

Item foreground: ■ Black

Sample:

```
model foo "This is a sample model"
  extends bar;

public
  parameter Real yMin = 2, yMax = 10;  // Some comment
  parameter Real y0[:] = zeros(size(k, 1));
  parameter Boolean signal = time > 0.5, setSignal = true;

equation
  assert(yMin <= yMax);
  connect(p,n);

end foo;
```

Options

☑ Collapse annotations

☑ Show a warning message before deleting collapsed annotations

Figure 3-56: Editing the options of the **Modelica Text View**.

All settings in the dialog box are automatically saved when closing the dialog box and restored the next time the model editor is started.

The sections of the **Modelica Text View** options are described in detail below.

- **Font and Colors**
  The **Font and Colors** settings allow you to establish a custom font and color scheme for various display items in the **Modelica Text View**.
  - **Font**. The font style to use in the **Modelica Text View**. Changing the font affects all **Display** items.
  - **Size**. Specifies the point size of the font to use in the **Modelica Text View**. A point is a traditional measure used by typesetters and is equal to 1/72 of an inch. Changing the font size affects all **Display** items.
  - **Display items**. Lists the items for which you can modify the foreground color. Text is the default display item. As such, properties assigned to **Text** will be overridden by properties assigned to other display items. For example, if you assign the color blue to **Text** and the color green to **Identifier**, all identifiers will appear in green.

In this example, **Identifier** properties override **Text** properties.

  - **Item foreground**. Specifies the foreground color of the item selected in **Display** items.
  - **Sample**. Displays a sample of the font and color scheme for the **Modelica Text View**. You can use this box to preview the results as you experiment with different formatting options.

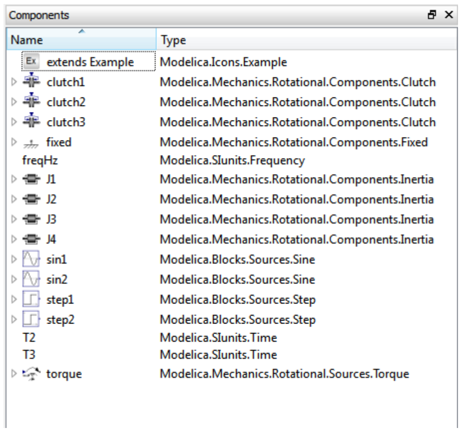
- **Options**
  These options allow you to customize some aspects of the **Modelica Text View**.
  - **Collapse annotations**. If checked, annotations in class definitions are by default collapsed into a ¤ character in the **Modelica Text View**.
  - **Show a warning message when deleting collapsed annotations**. Specifies whether or not a warning message is shown when a collapsed annotation is deleted in the **Modelica Text View**.

3.5 Component Window

The component window on the right-hand side of the model editor lists all components of the currently open class in a tree view. The components are grouped with respect to the class in which they are declared. Base classes in the component window are recognized by the *extends* prefix before their name.

Components and classes with no declared components appear as leaves in the tree view, while components and classes containing components are represented as branches. To expand a component or class, click the symbol to the left of its icon (or name if no icon is available). Clicking the symbol of an expanded component or class will collapse it again.

The tree view has two columns where the first column shows the icon (if available) and the name of the class or component, while the second column shows the component type for components and the full hierarchical name for classes. The second column showing type information is not visible by default. See Section 3.5.1 on page 77 on how to toggle the visibility of the columns.



Name	Type
extends Example	Modelica.Icons.Example
clutch1	Modelica.Mechanics.Rotational.Components.Clutch
clutch2	Modelica.Mechanics.Rotational.Components.Clutch
clutch3	Modelica.Mechanics.Rotational.Components.Clutch
fixed	Modelica.Mechanics.Rotational.Components.Fixed
freqHz	Modelica.SIunits.Frequency
J1	Modelica.Mechanics.Rotational.Components.Inertia
J2	Modelica.Mechanics.Rotational.Components.Inertia
J3	Modelica.Mechanics.Rotational.Components.Inertia
J4	Modelica.Mechanics.Rotational.Components.Inertia
sin1	Modelica.Blocks.Sources.Sine
sin2	Modelica.Blocks.Sources.Sine
step1	Modelica.Blocks.Sources.Step
step2	Modelica.Blocks.Sources.Step
T2	Modelica.SIunits.Time
T3	Modelica.SIunits.Time
torque	Modelica.Mechanics.Rotational.Sources.Torque

Figure 3-57: The components of the `Modelica.Mechanics.Rotational.Examples.CoupledClutches` model.

3.5.1 Toggling Column Visibility

The columns of the component window can be toggled on and off by right-clicking the column header. This will bring up a popup menu from which you can choose the name of the column to show or hide. By default, only the **Name** column is shown.

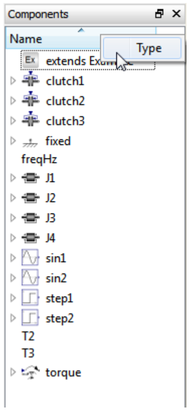


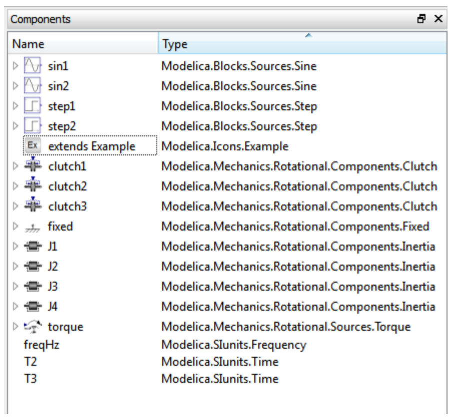
Figure 3-58: Toggling the visibility of component window columns.

3.5.2 Sorting Components

The items in the component window can be sorted by name or type. By default the components are sorted by name in ascending order, which is indicated by the arrow pointing upward in the right corner of the **Name** column title.

To sort the items by type in ascending order, click once on the **Type** column title (if the **Type** column is not visible, see Section 3.5.1 on page 77 on how to toggle the visibility of the columns). Click the **Type** column title a second time to sort the components by type in descending order.

Note how the pointing direction of the arrow switches when toggling between the ascending and descending sorting orders.



Name	Type
sin1	Modelica.Blocks.Sources.Sine
sin2	Modelica.Blocks.Sources.Sine
step1	Modelica.Blocks.Sources.Step
step2	Modelica.Blocks.Sources.Step
extends Example	Modelica.Icons.Example
clutch1	Modelica.Mechanics.Rotational.Components.Clutch
clutch2	Modelica.Mechanics.Rotational.Components.Clutch
clutch3	Modelica.Mechanics.Rotational.Components.Clutch
fixed	Modelica.Mechanics.Rotational.Components.Fixed
J1	Modelica.Mechanics.Rotational.Components.Inertia
J2	Modelica.Mechanics.Rotational.Components.Inertia
J3	Modelica.Mechanics.Rotational.Components.Inertia
J4	Modelica.Mechanics.Rotational.Components.Inertia
torque	Modelica.Mechanics.Rotational.Sources.Torque
freqHz	Modelica.SIunits.Frequency
T2	Modelica.SIunits.Time
T3	Modelica.SIunits.Time

Figure 3-59: Sorting the components in the component window by their type.

3.5.3 Editing Classes

The base classes and the classes of the components listed in the component window can be edited by right-clicking on a class or component and choosing **Open Class** from the popup menu. The class will be opened in a new class window unless already open.

3.5.4 Removing Components

The component window can also be used to remove components from a class. First, make sure that the active view of the class window is either the **Icon View** or the **Diagram View**. Select the component(s) to remove, and choose **Edit ▸ Delete** or right-click the component's name and choose **Delete** from the popup menu. Note that it is not possible to delete individual inherited components, nor is it possible to delete components while editing a class in the **Modelica Text View**.

To remove all components inherited from a specific extended class you can remove the inheritance relationship by selecting the extended class in the component window and deleting it. For more information, see Section 3.5.5 on page 79.

3.5.5 Removing Inheritance Relationships

The component window groups components with regards to their declaration class, making it easy to see from what class components are inherited. By selecting an extended class in the component window and choosing **Edit ▸ Delete**, the inheritance relationship can be removed. The class will be removed from the list of extended classes, causing all components inherited from that class to be removed.

The list of extended classes can also be edited from the **Class Properties** dialog box, where you may also add classes; see Section 3.2.9 on page 41.

Note that it is not possible to remove an inheritance relationship using the component window while editing a class in the **Modelica Text View**.

3.5.6 Renaming Components

A component (or extended class) can be renamed from the component window by right-clicking it, choosing **Rename** from the popup menu, and editing the name in the properties dialog box. It is also possible to rename a component by selecting it and pressing the **F2** key.

When renaming a class or component using the properties dialog box, all references to the renamed class or component, for instance in connection equations, will be updated automatically in all currently loaded classes. When the renaming operation is completed, a list of all classes that were modified due to references to the renamed class or component is shown.

Note that it is not possible to rename a component or class using the component window while editing a class in the **Modelica Text View**, nor is it possible to rename an inherited component.

3.5.7 Changing the Type of Components

The type of a component can be changed if the component is declared in the class of the active class window, or if the component is declared as replaceable. Although the result is the same, there is a slight difference.

In the former case, the type specifier of the declaration is simply substituted in the class in which the component is declared, while in the latter case, the type of the component is changed by introducing a redeclare modification in the class of the active class window. This makes it possible to replace a component somewhere in the component hierarchy without actually changing the definition of the class in which it is declared.

For instance, if we have a system board model with an integrated circuit (IC) component whose internal transistor components are replaceable, we can change the type of a transistor component without affecting other models using the IC component. The reason for this is that the redeclare modification that actually changes the type of the transistor component is introduced in the system board model.

By default components are not replaceable. For information on how to make a component replaceable, see Section 3.5.8.

To change the type of a component, right-click the component in the component window and choose **Redeclare** from the popup menu.

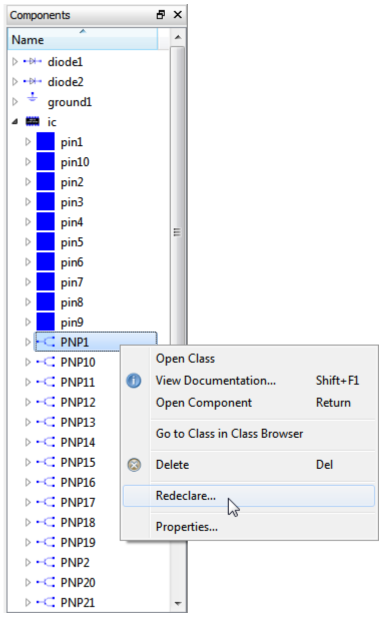
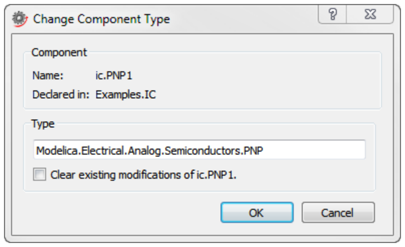


Figure 3-60: Selecting a component to change its type.

In the dialog box that is shown, enter the new type of the component and click **OK**. It is also possible to drag a class from the **Class Browser** and drop it in the text box.



Component
Name: ic.PNP1
Declared in: Examples.IC
Type
Modelica.Electrical.Analog.Semiconductors.PNP
☐ Clear existing modifications of ic.PNP1.

Figure 3-61: Changing the type of a component.

Note that the Modelica language restricts the possibilities when changing the type of a component using a redeclare modification. One of the more basic requirements is that the new type must have an interface compatible with the interface of the original declaration.

Providing a Set of Candidate Components for Redeclarations

If you develop a model with replaceable components you may want to provide a set of candidate components for the user to choose from when a component is redeclared. This can be done using the choices annotation, which is a standard Modelica annotation.

If we for instance have an electrical model of a main board with a CPU, we can make the CPU replaceable and introduce a choices annotation with configurations of pin compatible processors, making it possible for a user of the main board to easily test the main board with different processors. The choices annotation will be an annotation for the CPU component in the main board model.

```
replaceable Electrical.MC68000 cpu annotation(
  choices(choice(redeclare Computer.MC68000 cpu(freq=8)
                "Motorola 68000 at 8MHz" ),
          choice(redeclare Computer.MC68000 cpu(freq=16)
                "Motorola 68000 at 16MHz" ),
          choice(redeclare Computer.MC68010 cpu(freq=16)
                "Motorola 68010 at 16MHz" )),
  Placement(...)));
```

In the example above, we provide three candidates for the redeclaration. A MC68000 CPU at 8MHz, a MC68000 CPU at 16MHz, and a MC68010 CPU at 16MHz. When the user adds the main board as a component in his computer model, changing the CPU becomes a matter of right-clicking the CPU, choosing **Redeclare** from the menu, and then selecting the new processor from a list of candidates. As can be seen in Figure 3-62, the strings in the choices annotation represent the candidates when redeclaring the CPU component.

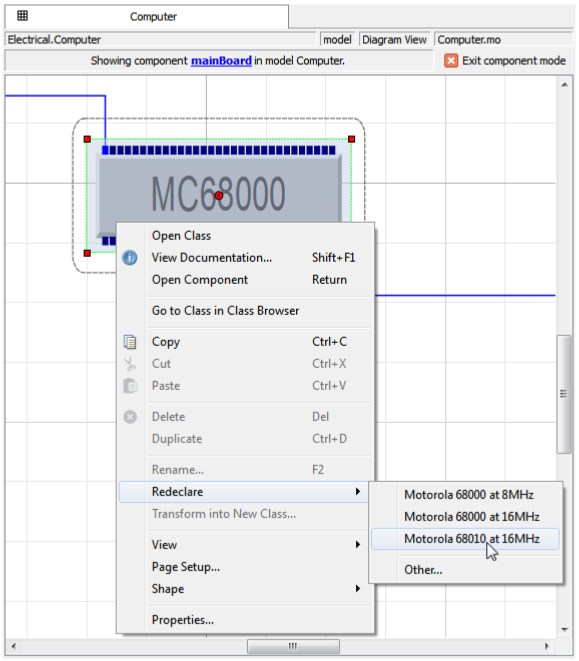


Figure 3-62: Redeclaring a component with a choices annotation.

3.5.8 Editing the Properties of Components

The properties of a component (or base class) can be edited by right-clicking the name of the item and choosing **Properties** from the popup menu, or by selecting the item and choosing **Properties** from the **Edit** menu.

For more information on the **Component Properties** dialog box, see "Editing the Properties of Components" on page 59. For more information on the **Class Properties** dialog box, see "Editing the Properties of Classes" on page 41. Note that it is not possible to edit the properties of a component or class while editing a class in the **Modelica Text View**.

3.5.9 Adding Graphical Representations for Components

When adding a component to a class in the **Modelica Text View** you may notice that the component does not appear in the **Diagram View** unless you specify a graphical annotation for it. The graphical annotation describes the placement of the component in the **Diagram View**, and for connector components, also their placement in the **Icon View**.

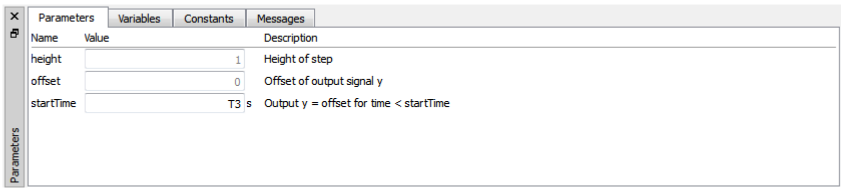
The model editor can generate a graphical annotation for a component if you drag the component from the component window and drop it in the **Diagram View**. If the component is a connector component, it will also appear in the **Icon View**.

Graphical annotations for several components can be generated at once by dragging a selection of components from the component window and dropping them in the **Diagram View**. The components will then be laid out in a grid.

Graphical annotations for existing connections will also be generated, if missing, when graphical annotations for connected components are generated.

3.6 Parameters View

The **Parameters** view, found in the bottom area of the model editor, is used to view and edit parameters of the class in the active class window, as well as parameters of its components. If no component is selected, the parameters of the class will be shown. To show the parameters of a specific component, select the component in either the **Icon View**, **Diagram View**, or the **Components** window.



Name	Value	Description
height	1	Height of step
offset	0	Offset of output signal y
startTime	T3 s	Output y = offset for time < startTime

Figure 3-63: Viewing the parameters of the `step2` component in the `Modelica.Mechanics.Rotational.Examples.CoupledClutches` model.

When viewing the parameters of a specific component, the color of the parameter values is either gray or black. A gray color indicates that the parameter value is the default value of the component. If the parameter value has been modified in the class of the active class window, the color of the value will be black. To remove the modification of a parameter and restore the default value, simply delete the contents in the text box while editing the value. See Section 3.6.2 on page 85 for more information on how to edit parameters.

The **Parameters** view is visible by default but can be hidden by clicking the X to the left of the window list. To show the view if hidden, choose **View ▸ Other Windows ▸ Parameters**.

3.6.1 Sorting Parameters

The parameters in the **Parameters** view can be sorted by name or declaration order. By default the parameters are sorted by name in ascending order.

To sort the parameters by declaration order in ascending order, right-click an empty area of the **Parameters** view, and choose **Arrange by ▸ Declaration Order** from the popup menu.

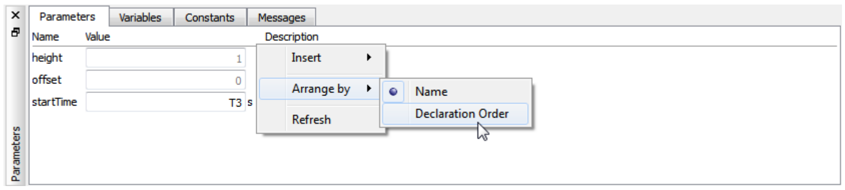


Figure 3-64: Sorting the parameters in the **Parameters** view by declaration order.

3.6.2 Editing Parameters

Before attempting to edit a parameter in the **Parameters** view, make sure that either the **Icon View** or **Diagram View** is the active view of the class window. Editing in the **Parameters** view is disabled while editing the class in the **Modelica Text View**.

To edit the value of a parameter, click the **Value** box to the right of its name, edit the existing value or type in a new value, and press **Return**, **Enter**, or **Tab**. Note that parameters declared using the final or outer attribute cannot be edited. For more information on how to declare a parameter final or outer, see "Editing the Properties of Components" on page 59.

On some occasions you may find it useful to set the value of a parameter to the same value for several components. This is possible as long as the components are of the same type. Select the components whose parameters you wish to edit. For those parameter values that are the same for all components, the values will be shown in the **Value** boxes. If the values are different, the **Value** boxes will show "No Common Value". When typing in a new parameter value or editing an existing value, all of the selected components will be affected. Components that inherit from the same base class can be selected and the parameters from the base class edited in the same way.

3.6.3 Adding Parameters

To add a parameter, choose **Parameter** from the **Insert** menu, or right-click an empty area of the **Parameters** view, **Variables** view, **Constants** view, or component window and choose **Insert ▸ Parameter** from the popup menu. The dialog box that appears is the common dialog box for adding variables of any kind of variability. The variability will automatically be set to **Parameter** as you chose to add a parameter.

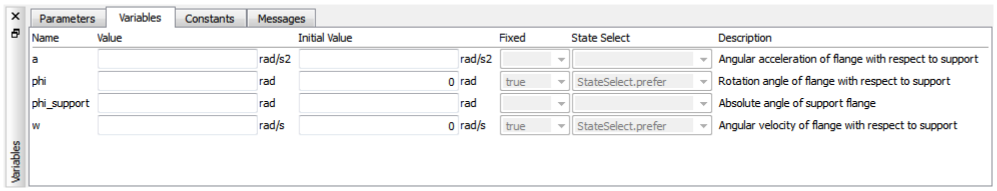
The dialog box lets you specify a type, name, default value, and description, as well as various attributes for the parameter. To add the parameter, click the **Add** button. The parameter will appear in the **Components** window. If the **Parameters** view is visible, the parameter will appear there as well. The dialog box will remain open to let you add more variables. To close it, click the **Close** button. For a detailed description of the dialog box, see Section 3.7.3 on page 88.

3.6.4 Removing Parameters

To remove a parameter of a class, locate the component that represents the parameter in the **Components** window and remove it. For information on how to remove components, see "Removing Components" on page 78.

3.7 Variables View

The **Variables** view is found in the bottom area of the model editor and is used to view and edit the variables of the class in the active class window, as well as the variables of its components. If no component is selected, the variables of the class will be shown. To show the variables of a specific component, select the component in either the **Icon View**, **Diagram View**, or the component window.



Name	Value		Initial Value		Fixed	State Select	Description
a		rad/s2		rad/s2			Angular acceleration of flange with respect to support
phi		rad	0	rad	true	StateSelect.prefer	Rotation angle of flange with respect to support
phi_support		rad		rad			Absolute angle of support flange
w		rad/s	0	rad/s	true	StateSelect.prefer	Angular velocity of flange with respect to support

Figure 3-65: The variables of the `Modelica.Mechanics.Rotational.Accelerate` model.

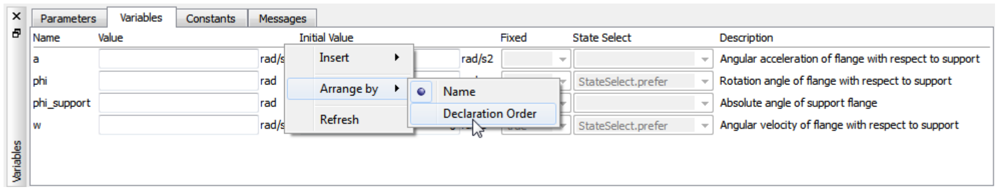
When viewing the variables of a specific component, the color of the variable attributes is either gray or black. A gray color indicates that the value is the default value of the component. If a variable attribute has been modified in the class of the active class window, it will be black. To remove the modification of a variable and restore the default value, simply delete the contents of the text box while editing the attribute. See Section 3.7.2 on page 87 for more information on how to edit variables.

The **Variables** view is visible by default but can be hidden by clicking the X in the left corner of the window list. To show the **Variables** view if hidden, choose **View ▸ Other Windows ▸ Variables**.

3.7.1 Sorting Variables

The variables in the **Variables** view can be sorted by name or declaration order. By default the variables are sorted by name in ascending order.

To sort the variables by declaration order in ascending order, right-click an empty area of the **Variables** view, and choose **Arrange by ▸ Declaration Order** from the popup menu.



Name	Value		Initial Value		Fixed	State Select	Description
a		rad/s		rad/s2			Angular acceleration of flange with respect to support
phi		rad				StateSelect.prefer	Rotation angle of flange with respect to support
phi_support		rad					Absolute angle of support flange
w		rad/s				StateSelect.prefer	Angular velocity of flange with respect to support

Figure 3-66: Sorting the variables in the **Variables** view by declaration order.

3.7.2 Editing Variables

Before attempting to edit a variable in the **Variables** view, make sure that either the **Icon View** or **Diagram View** is the active view of the class window. Editing in the **Variables** view is disabled while editing the class in the **Modelica Text View**.

Variables have several editable attributes. To give a variable an initial value, edit the value of the corresponding **Initial Value** box. All editable attributes of variables are explained in detail below. Note that the number of attributes available will depend on the type of the variable. For instance, a variable of type `String` has only the attribute value.

- **Initial Value**. The initial value of the variable, by default zero or an array of which all elements are zero.
- **Value**. Specify a value to give the variable a constant value throughout the simulation.

- **Fixed.** If set to false (default setting for variables), the simulation tool (solver) is free to use another initial value than the one specified if necessary in order to solve the systems of equations. If set to true, the initial value of the variable is fixed to the value specified in the **Initial Value** box.
  Note: State variables have the fixed attribute set to true as the default setting. This does not conform to standard Modelica, but since *SystemModeler* allows overdetermined system of equations as initial conditions, it is more intuitive with the fixed attribute set to true for state variables.
- **State Select**. This attribute provides manual control over the selection of state variables for numerical solution of equation systems. The alternatives are as follows:
  - **never**. The variable will never be selected as a state variable.
  - **avoid**. Use the variable as a state variable only if it appears differentiated.
  - **default**. Select the variable as a state variable if appropriate, meaning that the variable is likely to be selected as a state variable if its derivative appears in the model. This is the default setting.
  - **prefer**. The variable is preferably selected as a state variable over those that have the default attribute.
  - **always**. The variable will always be selected as a state variable.

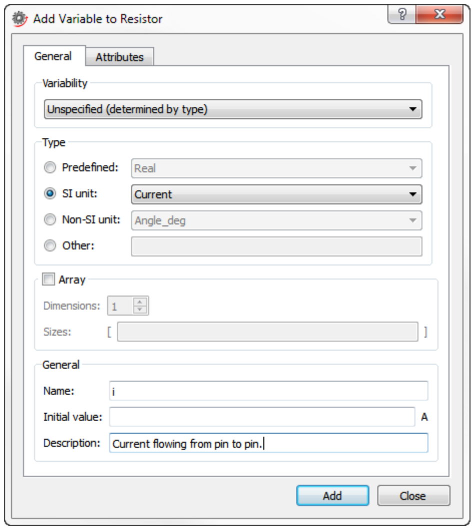
Note that attributes of a variable declared with the final or outer attribute are not possible to edit. For more information on how to declare a variable final or outer, see "Editing the Properties of Components" on page 59.

On some occasions you may find it useful to set an attribute of a variable for several components at once. This is possible as long as the components are of the same type. Select the components whose variables you wish to edit. For those variable attributes that are the same for all components, the values will be shown in the value text boxes. If the values differ, the text boxes will show "No Common Value". When typing in a new value for a parameter or editing an existing value, all of the selected components will be affected. Components that inherit from the same base class can be selected and the variables from the base class edited in the same way.

3.7.3 Adding Variables

To add a variable, choose **Variable** from the **Insert** menu, or right-click an empty area of the **Parameters** view, **Variables** view, **Constants** view, or component window and choose

Insert ▸ Variable from the popup menu. The dialog box that appears is the common dialog box for adding variables of any kind of variability.


Add Variable to Resistor

General | Attributes

Variability: Unspecified (determined by type)

Type:
- Predefined: Real
- (●) SI unit: Current
- Non-SI unit: Angle_deg
- Other:

☐ Array
- Dimensions: 1
- Sizes: [ ]

General
- Name: i
- Initial value: A
- Description: Current flowing from pin to pin.

Add | Close

Figure 3-67: Adding a continuous variable of the SI unit type **Current** to a resistor model.

The dialog box is divided into two views:

- **General**. In this view you specify the variability, type, dimensions, name, value, and description of the variable.
- **Attributes**. This view lets you specify various attributes of the variable.

To add the variable, click the **Add** button. The variable will appear in the **Components** window. If the **Variables** view is visible the variable will appear there as well. The dialog box will remain open to let you add more variables. To close it, click the **Close** button.

The sections of the dialog box are described in more detail below.

- **Variability**, **General** view
  Specifies the variability of the variable, with respect to simulation time.
  - **Unspecified** (default). Continuous for variables of type `Real`. Discrete for variables of type `Boolean`, `Integer`, `String`, and `enumeration`.
  - **Discrete**. Discrete variables are piecewise constant with respect to simulation time and may only be changed by events during simulation.
  - **Constant**. Constants can never change once defined.

  - **Parameter**. Parameters may change before simulation, but remain constant during simulation.

- **Type**, **General** view
  Specifies the type of the variable. Note that the SI unit and non-SI unit types are only available if the Modelica Standard Library is loaded.
  - **Built-in**. Modelica built-in types.
  - **SI unit**. SI unit types defined in the Modelica Standard Library.
  - **Non-SI unit**. Non-SI unit types defined in the Modelica Standard Library.
  - **Other**. Custom type. If you do not know the full path of the type, you can search for it using the **Class Browser** and then drag the type from the **Class Browser** and drop it in the text box. See Section 3.2.3 on page 36 for information on how to use the **Class Browser** to find types and other classes.

- **Array**, **General** view
  Specifies the dimensions of the variable. Leave this section unchecked for scalars.
  - **Dimensions**. The number of dimensions.
  - **Sizes**. The dimension sizes. If the number of dimensions exceeds 4, a single text box is shown where the dimension sizes are separated using a comma.

- **General**, **General** view
  Specifies the name, value, and description of the variable.
  - **Name**. The name of the variable.
  - **Value** (optional). The value for variables with the variability set to **Parameter** or **Constant**, and the initial value for continuous and discrete variables.
  - **Description** (optional). A short description of the variable.

- **Properties**, **Attributes** view
  Specifies the properties of the variable.
  - **Final**. Prevents modifications of the variable.
  - **Protected**. Access to the variable is restricted to the class in which it is declared, and to classes inheriting that class. Clear this checkbox to make the variable accessible from all classes.

  - **Flow**. Applies only to variables declared in connector classes. The flow property specifies whether the sum of the quantity should sum to zero when connected. Select this checkbox for variables specifying a flow quantity such as current, fluid, force, etc. Clear this checkbox for variables specifying potential (non-flow) quantities, such as voltage, pressure, position, etc.
  - **Replaceable**. Allows modifications of the variable in form of redeclarations.

- **Causality**, **Attributes** view
  Specifies the causality of the variable. The causality of a variable does not have to be explicitly specified unless the variable is an input or output to a block or function.
  - **None** (default). No explicitly specified causality.
  - **Input**. The variable is an input to a class.
  - **Output**. The variable is an output to a class.

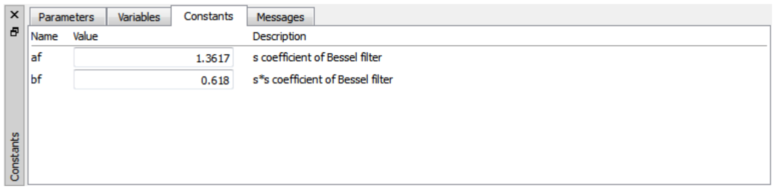
- **Dynamic Reference**, **Attributes** view
  Specifies the dynamic reference property of the variable.
  - **Inner**. Enable access to the variable in nested scopes.
  - **Outer**. Make the variable reference an inner variable with the same name in an enclosing scope.

3.7.4 Removing Variables

To remove variables of a class, locate the component that represents the variable in the component window and remove it. For information on how to remove components, see "Removing Components" on page 78.

3.8 Constants View

The **Constants** view, found in the bottom area of the model editor, is used to view and edit the constants of the class in the active class window, as well as the constants of its components. If no component is selected, the constants of the class will be shown. To show the constants of a specific component, select the component in either the **Icon View**, **Diagram View**, or the **Components** window.



Name	Value	Description
af	1.3617	s coefficient of Bessel filter
bf	0.618	s*s coefficient of Bessel filter

Figure 3-68: The constants of the `Modelica.Mechanics.Rotational.Position` model.

When viewing the constants of a specific component, the color of the constant values is either gray or black. A gray color indicates that the constant value is the default value of the component. If the constant value has been modified in the class of the active class window, the color of the value will be black. To remove the modification of a constant and restore the default value, if any, simply delete the contents of the text box while editing the value. See Section 3.8.2 on page 93 for more information on how to edit constants.

The **Constants** view is visible by default but can be hidden by clicking the X to the left of the window list. To show the view if hidden, choose **View ▸ Other Windows ▸ Constants**.

3.8.1 Sorting Constants

The constants in the **Constants** view can be sorted by name or declaration order. By default the constants are sorted by name in ascending order.

To sort the constants by declaration order in ascending order, right-click an empty area of the **Constants** view, and choose **Arrange by ▸ Declaration Order** from the popup menu.

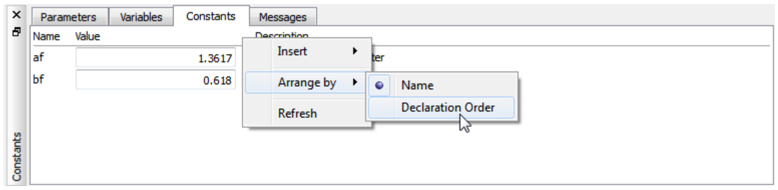


Figure 3-69: Sorting the constants in the **Constants** view by declaration order.

3.8.2 Editing Constants

Before attempting to edit a constant in the **Constants** view, make sure that either the **Icon View** or **Diagram View** is the active view of the class window. Editing in the **Constants** view is disabled while editing the class in the **Modelica Text View**.

To edit the value of a constant, click the **Value** box to the right of its name, edit the existing value or type in a new value, and press **Return**, **Enter**, or **Tab**. Note that constants declared using the final or outer attribute cannot be edited. For more information on how to declare a constant final or outer, see "Editing the Properties of Components" on page 59.

On some occasions you may find it useful to set the value of a constant to the same value for several components. This is possible as long as the components are of the same type. Select the components whose constants you wish to edit. For those constant values that are the same for all components, the values will be shown in the **Value** boxes. If the values are different, the **Value** boxes will show "No Common Value". When typing in a new constant value or editing an existing value, all of the selected components will be affected. Components that inherit from the same base class can be selected and the constants from the base class edited in the same way.

3.8.3 Adding Constants

To add a constant, choose **Constant** from the **Insert** menu, or right-click an empty area of the **Parameters** view, **Variables** view, **Constants** view, or component window and choose **Insert ▸ Constant** from the popup menu. The dialog box that appears is the common dialog box for adding variables of any kind of variability. The variability will automatically be set to **Constant** as you chose to add a constant.

The dialog box lets you specify a type, name, value, and description, as well as various attributes for the constant. To add the constant, click the **Add** button. The constant will appear in the **Components** window. If the **Constants** view is visible the constant will appear there as well. The dialog box will remain open to let you add more variables. To close it, click the **Close** button. For a detailed description of the dialog box, see Section 3.7.3 on page 88.

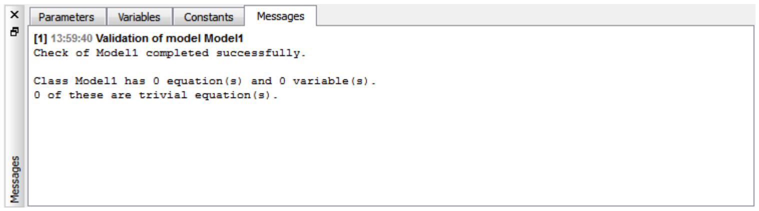
3.8.4 Removing Constants

To remove a constant of a class, locate the component that represents the constant in the **Components** window and remove it. For information on how to remove components, see "Removing Components" on page 78.

3.9 Messages View

The **Messages** view is a non-interactive view used by the model editor to display results of operations requested by the user. The **Messages** view is also used to notify the user about certain types of errors in a class, such as syntax errors and unresolved references.

The **Messages** view is located in the bottom area of the model editor and is visible by default but can be hidden by clicking the X in the left corner of the window list. To show the **Messages** view if hidden, choose **View ▸ Other Windows ▸ Messages**.



[1] 13:59:40 Validation of model Model1

```
Check of Model1 completed successfully.

Class Model1 has 0 equation(s) and 0 variable(s).
0 of these are trivial equation(s).
```

Figure 3-70: Validation results in the **Messages** view.

3.9.1 Indication of New Unread Messages

Would the **Messages** view not be visible at the time a message is output, a number within brackets will follow directly after its title in the tab. This number indicates the total number of new unread messages. This number will be cleared as soon as the **Messages** view becomes visible again.

For more critical errors the **Messages** view will automatically become visible when the error message is output.

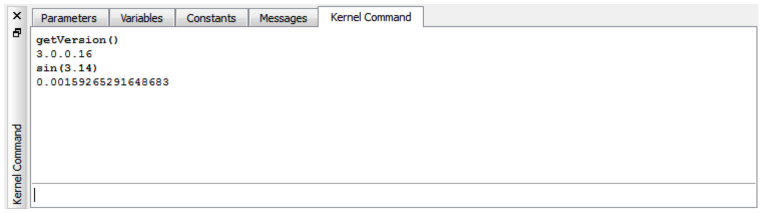
3.9.2 Clearing Messages

The **Messages** view can be cleared by right-clicking anywhere within the view and choosing **Clear All**.

3.10 Kernel Command View

The **Kernel Command** view is used for executing commands and evaluating Modelica expressions using the *SystemModeler* kernel. Open the **Kernel Command** view by choosing **View ▸ Other Windows ▸ Kernel Command**. It will appear in the bottom area of the model editor.

For information on available commands, see Appendix B.



```
getVersion()
3.0.0.16
sin(3.14)
0.0015926529164868
```

Figure 3-71: Executing commands in the **Kernel Command** view.

3.10.1 Clearing the View

The **Kernel Command** view can be cleared by right-clicking anywhere within the view and choosing **Clear All**.

3.11 Dynamic Help

While working with the model editor it is possible to get help and information about items in the model editor as well as loaded Modelica classes.

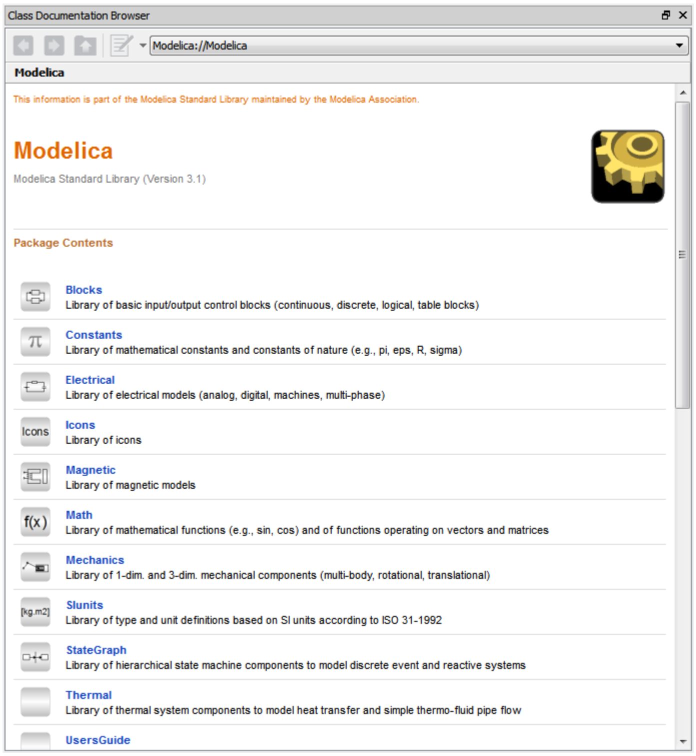
3.11.1 Tool Tips

Tool tips are the small yellow rectangles with help messages that pop up while keeping the mouse cursor still over an item, for example a button on the toolbar or a connection in the **Diagram View**.

In most dialog boxes you can get help with any item in the dialog box by clicking the question mark icon in the title bar and then clicking the item.

3.12 Class Documentation Browser

The **Class Documentation Browser** is used to browse and edit the embedded documentation of classes. To get an overview of the Modelica Standard Library, choose **Modelica Standard Library User Guide** from the **Help** menu.


Class Documentation Browser

Modelica://Modelica

Modelica

This information is part of the Modelica Standard Library maintained by the Modelica Association.

Modelica

Modelica Standard Library (Version 3.1)

Package Contents

Blocks
Library of basic input/output control blocks (continuous, discrete, logical, table blocks)

Constants
Library of mathematical constants and constants of nature (e.g., pi, eps, R, sigma)

Electrical
Library of electrical models (analog, digital, machines, multi-phase)

Icons
Library of icons

Magnetic
Library of magnetic models

Math
Library of mathematical functions (e.g., sin, cos) and of functions operating on vectors and matrices

Mechanics
Library of 1-dim. and 3-dim. mechanical components (multi-body, rotational, translational)

SIunits
Library of type and unit definitions based on SI units according to ISO 31-1992

StateGraph
Library of hierarchical state machine components to model discrete event and reactive systems

Thermal
Library of thermal system components to model heat transfer and simple thermo-fluid pipe flow

UsersGuide

Figure 3-72: Viewing the documentation of the `Modelica` package.

Viewing Documentation of Classes

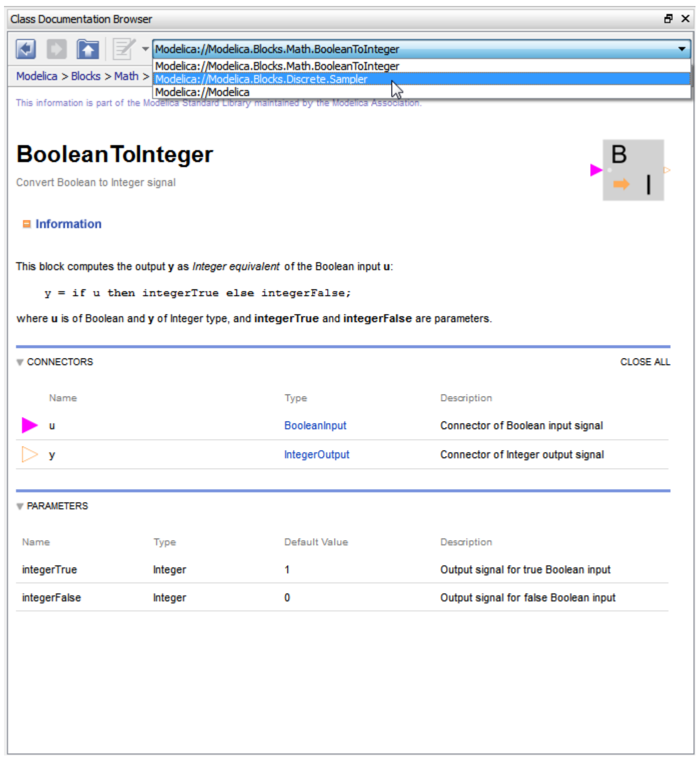
To get information about classes in the Modelica Standard Library or any other documented class, right-click on the icon of a class in the **Class Browser** and choose **View Documentation**. Clicking the **View Class Documentation** button on the toolbar will show the documentation of the class in the active class window. It is also possible to get information about classes by right-clicking components in the graphical views or in the component window and choosing **View Documentation** from the popup menu.

Navigating

The **Class Documentation Browser** keeps a record of all viewed pages. It is possible to go back and forth in this history of visited pages by clicking the **Back** and **Forward** buttons on the toolbar, or by choosing a specific page in the drop-down list of the browser.

Click the **Up** button on the toolbar to take a step up in the hierarchy of classes. Clicking the **Up** button while viewing the documentation of the class `Modelica.Blocks.Math` will take you to the documentation of the class `Modelica.Blocks`.

The top of the **Class Documentation Browser** also contains a hierarchy where you can click on any level of the hierarchy to get to the documentation for that level.



Class Documentation Browser

Modelica://Modelica.Blocks.Math.BooleanToInteger

Modelica://Modelica.Blocks.Math.BooleanToInteger
Modelica://Modelica.Blocks.Discrete.Sampler
Modelica://Modelica

Modelica > Blocks > Math >

This information is part of the Modelica Standard Library maintained by the Modelica Association.

BooleanToInteger

Convert Boolean to Integer signal

Information

This block computes the output **y** as *Integer equivalent* of the Boolean input **u**:

```
    y = if u then integerTrue else integerFalse;
```

where **u** is of Boolean and **y** of Integer type, and **integerTrue** and **integerFalse** are parameters.

CONNECTORS CLOSE ALL

	Name	Type	Description
▶	u	BooleanInput	Connector of Boolean input signal
▷	y	IntegerOutput	Connector of Integer output signal

PARAMETERS

Name	Type	Default Value	Description
integerTrue	Integer	1	Output signal for true Boolean input
integerFalse	Integer	0	Output signal for false Boolean input

Figure 3-73: Choosing a page to view in the history of visited pages in the **Class Documentation Browser**.

Printing

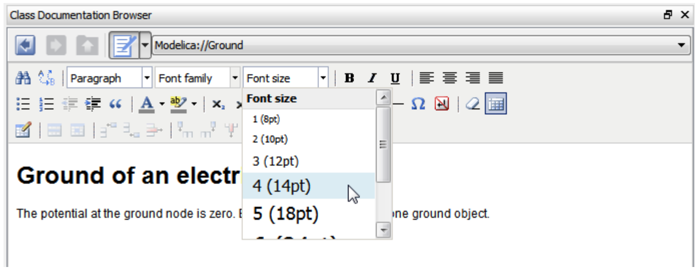
The page currently viewed in the **Class Documentation Browser** can be sent to the printer by choosing **Print** from the **File** menu.

Documenting Classes

The documentation for a class consists of information embedded in the class as an annotation (the information and revisions pages) as well as information that is autogenerated by *SystemModeler*, including information about parameters, variables, constants, and components. The documentation provided by annotations can be edited using the edit mode of the **Class Documentation Browser**. While viewing the page you want to edit in the browser, click the **Edit** button on the toolbar to switch to edit mode; clicking it again will take you back to view mode where you can see the results of your changes. Note that the edit mode is disabled for read-only classes as these cannot be edited.

Figure 3-74: The **Edit** button on the toolbar of the **Class Documentation Browser**.

The default documentation editor is a what you see is what you get (WYSIWYG) editor that lets you edit the documentation in an environment similar to that of a word processor.



Ground of an electr

The potential at the ground node is zero. E

ne ground object.

Figure 3-75: Editing the embedded documentation of a class in the WYSIWYG editor.

The format of the embedded documentation is standard HTML. If you are familiar with HTML, or if you require more control than what is provided by the WYSIWYG editor, you may want to switch to the HTML editor. This is done by clicking the arrow of the **Edit** button and choosing **HTML** from the drop-down menu.

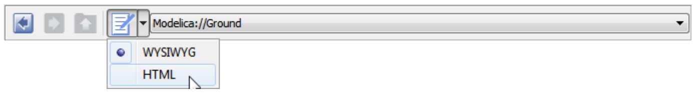
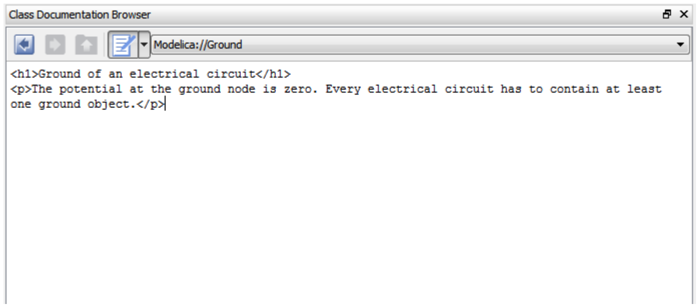


Figure 3-76: Switching to the HTML editor.

The HTML editor lets you edit the documentation in plain HTML. You may switch back and forth between the two editors at any time by using the drop-down menu of the **Edit** button.



```
<h1>Ground of an electrical circuit</h1>
<p>The potential at the ground node is zero. Every electrical circuit has to contain at least
one ground object.</p>
```

Figure 3-77: Editing the embedded documentation of a class in the HTML editor.

If you include images in your documentation, supported image formats include BMP, GIF, JPEG, and PNG. The path to image files may be given relative to the *.mo file in which they are referenced, or as an absolute path. Note that images specified using a relative path will not show up in the WYSIWYG editor. Therefore, when adding an image, we recommend switching back to the view mode to verify that the image was found.

Modelica links (links that start with Modelica://) are handled by the **Class Documentation Browser**, while external links are delegated and opened in your system's default web browser.

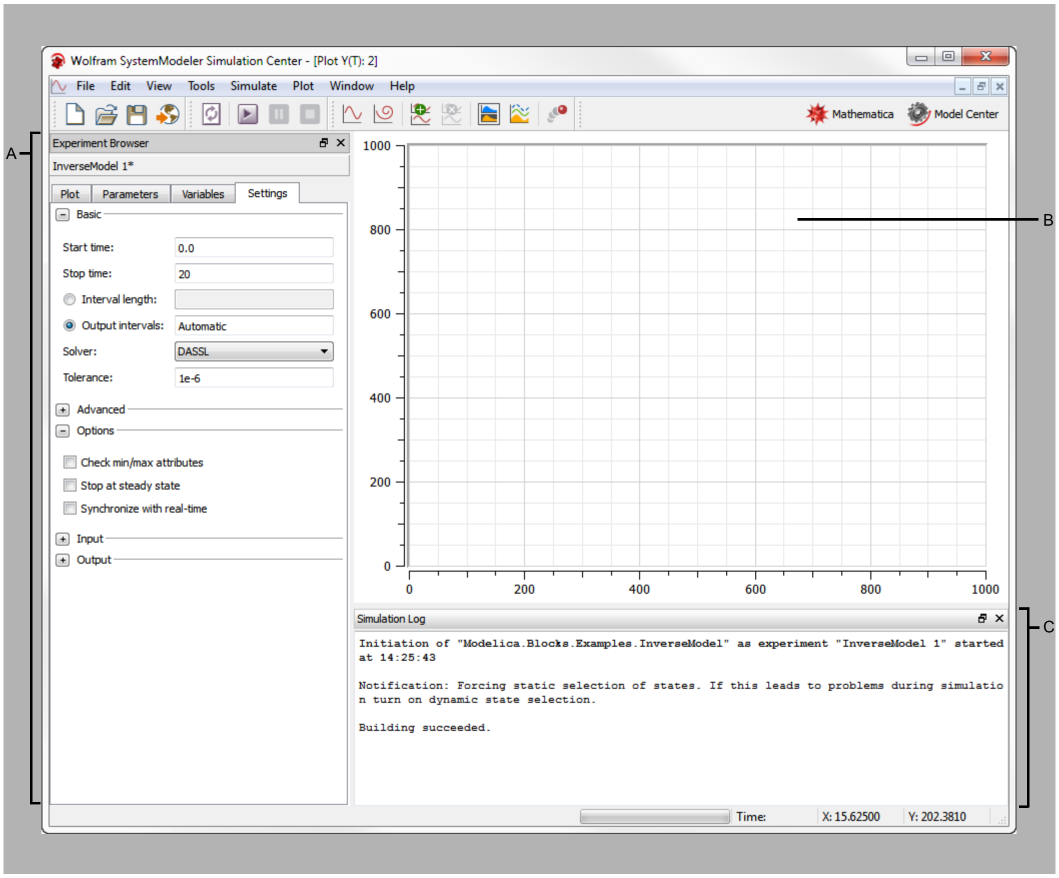
To create links to the documentation of another Modelica class, format the URL as Modelica:// followed by the name of the Modelica class. You can also create links that will open a class window for the specified class. This is done by adding a hash sign (#) and the name of the view to the link. For instance, the link Modelica://Modelica.Math.asin#diagram will show the **Diagram View** of the `Modelica.Math.asin` class when clicked.

Chapter 4: ***Simulation Center***

This chapter describes the *SystemModeler* environment for simulating models.

4.1 Introduction

By default *Simulation Center* starts with three windows: the **Experiment Browser** (A), a plot window (B), and the **Simulation Log** window (C); see Figure 4-1. The **Experiment Browser** and **Simulation Log** window can be dragged and dropped, wherever preferred within the main window of *Simulation Center*, or outside to make windows floating. Plot windows can be dragged and dropped within the main window, but not outside.



Start time: 0.0
Stop time: 20
Interval length:
Output intervals: Automatic
Solver: DASSL
Tolerance: 1e-6

Check min/max attributes
Stop at steady state
Synchronize with real-time

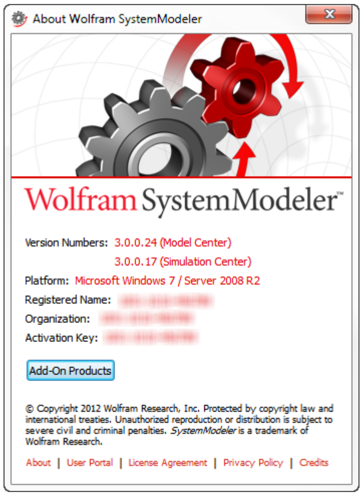
```
Initiation of "Modelica.Blocks.Examples.InverseModel" as experiment "InverseModel 1" started
at 14:25:43

Notification: Forcing static selection of states. If this leads to problems during simulatio
n turn on dynamic state selection.

Building succeeded.
```

Time: X: 15.62500 Y: 202.3810

Figure 4-1: An overview of *Simulation Center*.

4.1.1 Version and License Details

The version of *Simulation Center* can be found by choosing **About SystemModeler** from the **Help** menu.


About Wolfram SystemModeler

Wolfram SystemModeler™

Version Numbers: 3.0.0.24 (Model Center)
3.0.0.17 (Simulation Center)
Platform: Microsoft Windows 7 / Server 2008 R2
Registered Name: [redacted]
Organization: [redacted]
Activation Key: [redacted]

Add-On Products

© Copyright 2012 Wolfram Research, Inc. Protected by copyright law and international treaties. Unauthorized reproduction or distribution is subject to severe civil and criminal penalties. *SystemModeler* is a trademark of Wolfram Research.

About | User Portal | License Agreement | Privacy Policy | Credits

Figure 4-2: The **About SystemModeler** dialog box.

The dialog will also show the name of the licensee and the activation key. Click the **Add-On Products** button to get information about available add-ons.

4.1.2 Creating New Experiments

A new experiment is created by choosing **New** from the **File** menu. This will open a dialog where you specify which model you want to create an experiment for. Select the desired model from the model drop-down list. The model drop-down list contains all models that are currently open in *Model Center*. It is also possible to create an experiment for any model that is loaded into the *SystemModeler* environment by typing the name of the model and pressing the **Enter** key.

When you select the desired model and click **OK** or press **Enter**, a new experiment with the most recent updates of the model will be created. The new experiment will be initialized from the experiment annotation in the model. See Section 4.12 on page 132 for more information about the experiment annotation. All values that are not defined by the experiment annotation will be initialized from the default experiment settings; see Section 4.2.2 on page 107.

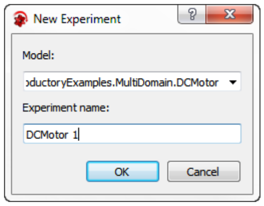


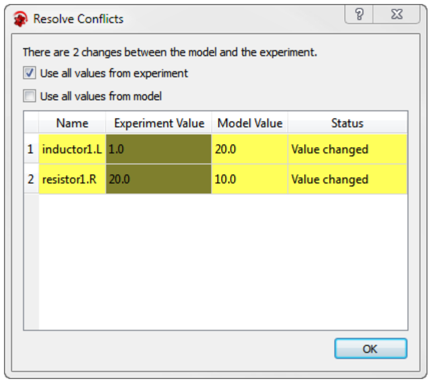
Figure 4-3: The **New Experiment** dialog box.

Note that if the model is later edited in *Model Center*, the experiment will not be automatically updated. To use the updated model, use the rebuild functionality, see Section 4.1.3 or create a new experiment.

If the translation process fails, for instance because the model is erroneous, an error message will be shown in the **Simulation Log** window. For more information on the **Simulation Log** window, see Section 4.8 on page 130.

4.1.3 Rebuild Experiments

If the model is edited in *Model Center*, the experiment needs to be rebuilt in order for it to use the most current model code. To rebuild the currently active experiment, choose **Simulation ▸ Rebuild**. If any parameter value or variable initial value in the experiment is not equal to what is specified in the model, the **Resolve Conflicts** dialog will appear. In this dialog you can select if the value from the model or the value from the experiment should be used.



There are 2 changes between the model and the experiment.

[✓] Use all values from experiment

[ ] Use all values from model

	Name	Experiment Value	Model Value	Status
1	inductor1.L	1.0	20.0	Value changed
2	resistor1.R	20.0	10.0	Value changed

Figure 4-4: The **Resolve Conflicts** dialog.

4.1.4 Open Experiments

You can open a *SystemModeler* experiment file (*.sme) in *Simulation Center* by choosing **File ▸ Open**. Alternatively, you can drop one or several *SystemModeler* experiment files anywhere in the *Simulation Center* window. Recently used files are available in the **File ▸ Recent Files** menu. Click on one of the menu items to open the corresponding file.

4.1.5 Saving Experiments

To save the currently active experiment, choose **File ▸ Save**. When an experiment is saved for the first time, a dialog box letting you choose a file name and location for the experiment will appear. Experiments are saved as *SystemModeler* experiment files.

To save the currently active experiment to a new file, choose **File ▸ Save As**.

4.1.6 Closing Experiments

To close the currently active experiment, choose **File ▸ Close**.

4.1.7 Simulating

Starting a Simulation

To simulate the active experiment, choose **Simulate** from the **Simulation** menu or click the **Simulate** button on the **Simulation** toolbar. During the simulation the current simulation time (model time) is shown in the status bar. When the simulation is finished, the results can be plotted and analyzed.

If a plot window is already open for the experiment that is being simulated, the window will be updated on a regular basis during the simulation with the values of the selected plot variables in the **Experiment Browser**. This makes it possible to study partial results of simulations before the simulation is completed.

Figure 4-5: The **Simulate** button on the **Simulation** toolbar.

Interrupting a Simulation

An ongoing simulation can be interrupted at any time by choosing **Simulation ▸ Stop** or by clicking the **Stop** button on the **Simulation** toolbar.

Figure 4-6: The **Stop** button on the **Simulation** toolbar.

Resuming an Interrupted Simulation

It is possible to resume an interrupted simulation by choosing **Simulation ▸ Resume**. This will initialize all state variables and discrete variables to the value they had at the time that the simulation was interrupted and continue the simulation from that point in time.

Pausing a Simulation

An ongoing simulation can be paused at any time by choosing **Simulation ▸ Pause** or by clicking the **Pause** button on the **Simulation** toolbar. To continue the simulation click the **Pause** button again.

Figure 4-7: The **Pause** button on the **Simulation** toolbar.

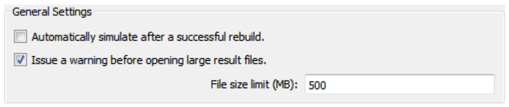
4.2 Options

The **Options** dialog box can be reached from the **Tools** menu. All settings available in the **Options** dialog box are automatically saved when closing the dialog box and restored the next time *Simulation Center* is started.

4.2.1 General

- **Automatically resimulate after successful rebuild**
  If enabled, the experiment is automatically simulated after a successful rebuild.
- **Issue a warning before opening large result files**
  Working with large result files can be very slow. Large result files are often an indication that it might be advisable to switch to an explicit number of output intervals or interval length and thereby avoiding the automatic setting. If enabled, a warning will

be issued before loading a result file that is larger than the specified limit.


General Settings

☐ Automatically simulate after a successful rebuild.

☑ Issue a warning before opening large result files.

File size limit (MB): 500

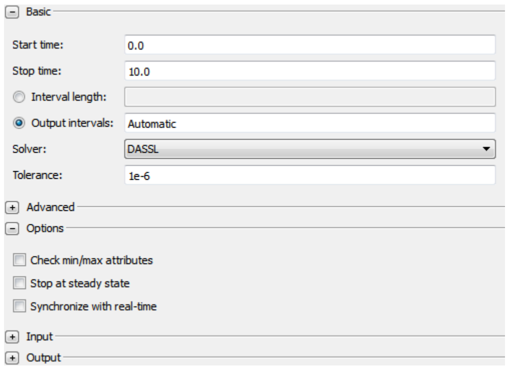
Figure 4-8: The **General** view of the **Options** dialog box.

4.2.2 Default Experiment Settings

If the model does not contain an experiment annotation (see Section 4.12 on page 132 for an explanation of the experiment annotation), all new experiments, when created, will be initialized with the default experiment settings.

If the default experiment settings are changed in the **Options** dialog box, all new experiments created after a change will receive the new settings. Please note that changing the default experiment settings in the **Options** dialog box has no effect on the active experiment. To change the settings of a specific experiment, see Section 4.3.3.

The **Output intervals** setting is by default set to Automatic. When set to Automatic, the output interval length is variable and decided by the solver during the simulation. The output intervals can be set to a fixed number by typing in a number in the text box. To revert to Automatic mode, delete the contents of the text box and press the **a** key.


Basic

Start time: 0.0

Stop time: 10.0

○ Interval length:

● Output intervals: Automatic

Solver: DASSL

Tolerance: 1e-6

Advanced

Options

☐ Check min/max attributes

☐ Stop at steady state

☐ Synchronize with real-time

Input

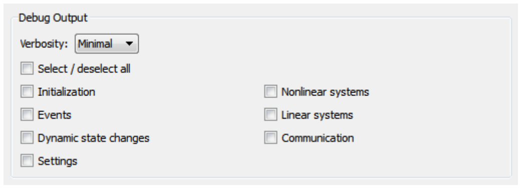
Output

Figure 4-9: The default experiment **Settings** view of the **Options** dialog box.

4.2.3 Debug Settings

To enable debug output for a simulation, choose **Tools ▸ Options** to bring up the **Options** dialog box and select the **Debug Output** view. The amount of debug output can be controlled by setting a verbosity level in combination with enabling output for different subsystems of the simulation. The verbosity level can take one of the following values: **Minimal**, **Normal**, or **Verbose**. Debug output can be enabled for the following subsystems:

- Initialization solver
- Event handling
- Settings
- Nonlinear systems solver
- Linear systems solver
- Communication


Debug Output

Verbosity: Minimal

- [ ] Select / deselect all
- [ ] Initialization
- [ ] Events
- [ ] Dynamic state changes
- [ ] Settings
- [ ] Nonlinear systems
- [ ] Linear systems
- [ ] Communication

Figure 4-10: The **Debug Output** view of the **Options** dialog box.

4.2.4 Plot

Legend

- **Use experiment name as prefix**
  If enabled, all legends are prefixed with the experiment.
- **Use shortest unique name**
  If enabled, as many path components as possible will be stripped from the variable name. For example, if we have the variables *a.b.r* and *a.c.r*, their legend text will be *b.r* and *c.r*.

Real-Time Plot

When running a real-time simulation and **Save results to file** is disabled, the simulation result is streamed to the plot using a network protocol and the plot is updated frequently. Here it is possible to set how long of a history to keep for the plotted variables and how often the plot should be updated.

- **Buffer length**
  Specifies how long of a history, in seconds, to keep for the plotted variables.
- **Plot update frequency**
  Specifies how many times per second the plot should be updated.

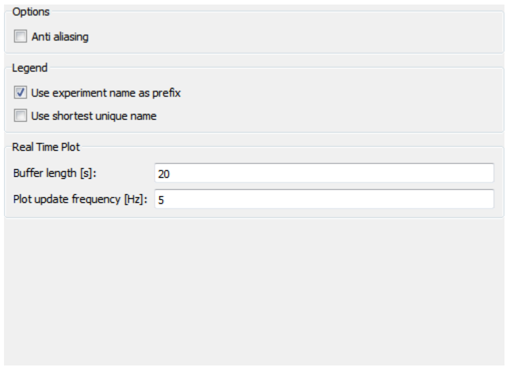

Options
[ ] Anti aliasing
Legend
[✓] Use experiment name as prefix
[ ] Use shortest unique name
Real Time Plot
Buffer length [s]: 20
Plot update frequency [Hz]: 5

Figure 4-11: The **Plot** view of the **Options** dialog box.

4.2.5 Translation

Options

- **Tearing**
  A technique used to reduce large sparse systems of equations to smaller denser systems. It can significantly improve the runtime performance of a model.

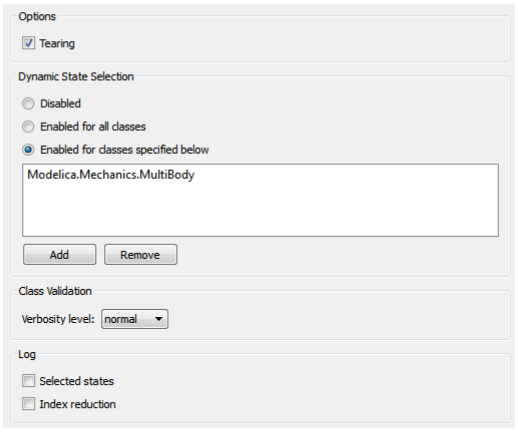
Dynamic State Selection

If enabled, a state selection algorithm that may introduce dynamic states is used. It is possible to enable it for all classes or only for classes that use the specified libraries.

Log

- **Selected states**
  If enabled, information about which states are selected during the translation phase is written to the build log.
- **Index reduction**
  If enabled, information about which equations are differentiated due to index reduction

during the translation phase is written to the build log.


Options

[✓] Tearing

Dynamic State Selection

○ Disabled
○ Enabled for all classes
● Enabled for classes specified below

Modelica.Mechanics.MultiBody

Class Validation

Verbosity level: normal

Log

☐ Selected states
☐ Index reduction

Figure 4-12: The **Translation** view of the **Options** dialog box.

4.3 Experiment Browser

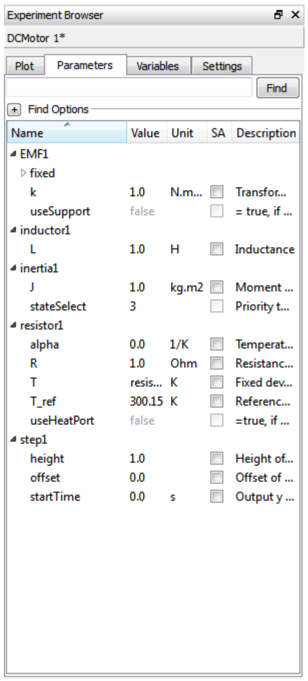
The **Experiment Browser** is used to browse experiments and to perform operations on the experiments. Each experiment has four views for different types of operations:

- **Plot** view, a view from which you can select what variables and parameters to plot.
- **Parameters** view, a view from which you can edit the parameter values of the experiment.
- **Variables** view, a view from which you can set the initial values for state variables.
- **Settings** view, which contains all settings for the experiment.

To change the active view of the **Experiment Browser**, click the tabs at the top of the browser.

4.3.1 Setting Parameter Values

The **Parameters** view of the **Experiment Browser** contains all parameters of the experiment. The parameters are presented in a tree view that is built up hierarchically with components as branches. Double-click a component name to expand or collapse its branch. By expanding a branch you will be able to access and edit the parameters of the corresponding component. All parameters that are changed compared to the default value in the model are highlighted in red.


Experiment Browser

DCMotor 1*

Plot | Parameters | Variables | Settings

Find

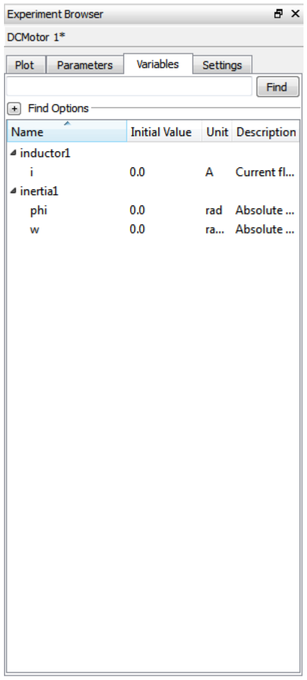
Find Options

Name	Value	Unit	SA	Description
EMF1				
fixed				
k	1.0	N.m...	☐	Transfor...
useSupport	false		☐	= true, if ...
inductor1				
L	1.0	H	☐	Inductance
inertia1				
J	1.0	kg.m2	☐	Moment ...
stateSelect	3		☐	Priority t...
resistor1				
alpha	0.0	1/K	☐	Temperat...
R	1.0	Ohm	☐	Resistanc...
T	resis...	K	☐	Fixed dev...
T_ref	300.15	K	☐	Referenc...
useHeatPort	false		☐	=true, if ...
step1				
height	1.0		☐	Height of...
offset	0.0		☐	Offset of ...
startTime	0.0	s	☐	Output y ...

Figure 4-13: The **Parameters** view of the **Experiment Browser**.

4.3.2 Setting Initial Values of State Variables

The **Variables** view of the **Experiment Browser** contains all state variables of the experiment. The state variables are presented in a tree view that is built up hierarchically with components as branches. Double-click a component name to expand or collapse its branch. By expanding a branch, you will be able to access and edit the state variables of the corresponding component. All variables that are changed compared to the default value in the model are highlighted in red.

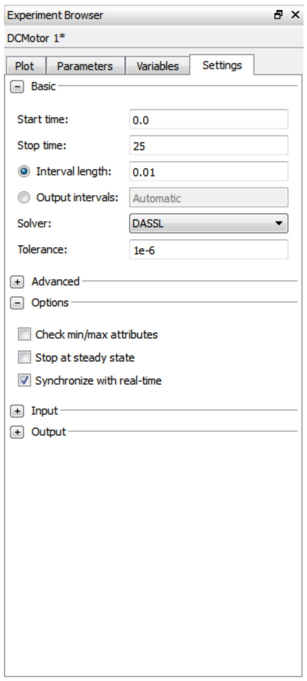


Name	Initial Value	Unit	Description
inductor1			
i	0.0	A	Current fl...
inertia1			
phi	0.0	rad	Absolute ...
w	0.0	ra...	Absolute ...

Figure 4-14: The **Variables** view of the **Experiment Browser**.

4.3.3 Changing Experiment Settings

The settings for a specific experiment can be changed from the **Settings** view of the **Experiment Browser**. Click the **Settings** tab to activate the **Settings** view. The settings are divided into **Basic**, **Advanced**, **Options**, and **Output**.



Experiment Browser

DCMotor 1*

Plot | Parameters | Variables | Settings

Basic

Start time:	0.0
Stop time:	25
● Interval length:	0.01
○ Output intervals:	Automatic
Solver:	DASSL
Tolerance:	1e-6

Advanced

Options

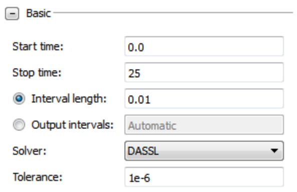
- ☐ Check min/max attributes
- ☐ Stop at steady state
- ☑ Synchronize with real-time

Input

Output

Figure 4-15: The **Settings** view of the **Experiment Browser**.

Basic



Basic

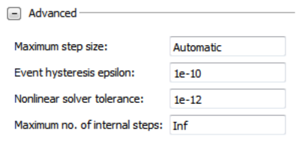
Start time:	0.0
Stop time:	25
● Interval length:	0.01
○ Output intervals:	Automatic
Solver:	DASSL
Tolerance:	1e-6

Figure 4-16: Basic experiment settings.

- **Start time**
  Specifies the start time of the simulation.

- **Stop time**
  Specifies the stop time of the simulation.
- **Interval length**
  Specifies the length of the interval between output points from the solver. Note that the solver will output at least two data points for every event that is generated in the model in addition to what is specified by the interval length.
- **Output intervals**
  Specifies the number of output intervals that the solver generates. The **Output Intervals** setting is by default set to Automatic. When set to Automatic, the output interval length is variable and decided by the solver during the simulation. This can potentially lead to huge datasets. Therefore, it might be better to use a fixed number of output intervals, or interval length, for more complex models. The output intervals can be set to a fixed number by typing in a number in the text box. To revert to Automatic mode, delete the contents of the text box and press the **a** key. Note that when a fixed number of output intervals is specified, the solver will output at least two data points for every event that is generated in the model in addition to what is specified by the number of output intervals.
- **Solver**
  Specifies the solver used to solve the dynamic system. There are five different solver choices: DASSL, CVODES, Explicit Euler, Heun's method, and Runge Kutta (RK4). DASSL is a variable step size and variable order solver that uses a backward differentiation formula method. CVODES is also a variable step size and variable order solver that uses a backward differentiation formula method. The added benefit of CVODES is that it supports forward sensitivity analysis. For more information on CVODES, see http://computation.llnl.gov/casc/sundials. Explicit Euler, Heun's method, and Runge Kutta (RK4) are fixed step size solvers suitable for real-time simulation. The step size is controlled by the **Interval length** setting, which also controls the number of output values in the simulation result.
- **Tolerance**
  Specifies the local tolerance that is used in each solver step. The final (global) error is in some way an accumulation of those per step errors. As a rule of thumb, reduce the desired global tolerance with a factor of 0.01 and use that as tolerance.

Advanced

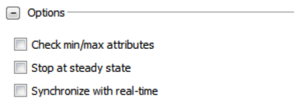

Advanced

Maximum step size: Automatic
Event hysteresis epsilon: 1e-10
Nonlinear solver tolerance: 1e-12
Maximum no. of internal steps: Inf

Figure 4-17: Advanced experiment settings.

- **Maximum step size**
  Specifies the maximum step size that the solver is allowed to use.
- **Event hysteresis epsilon**
  Specifies the hysteresis threshold used for detecting events.
- **Nonlinear solver tolerance**
  Specifies the tolerance for the nonlinear solver.
- **Maximum no. of internal steps**
  Specifies the maximum number of internal steps that the solver is allowed to take before reaching the next output point.

Options


Options

☐ Check min/max attributes
☐ Stop at steady state
☐ Synchronize with real-time

Figure 4-18: Experiment settings options.

- **Check min/max attributes**
  If enabled, the simulation will terminate if any variable with min/max attributes goes outside its valid region.
- **Stop at steady state**
  If enabled, the simulation will terminate if all state derivatives become equal to zero.
- **Synchronize with real time**
  If enabled, the simulation will be synchronized with real time. That is, the simulation will be slowed down so that the simulation progresses with the same pace as real time.

Input

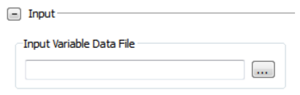
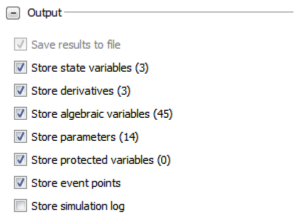


Figure 4-19: Experiment input settings.

- **Input Variable Data File**
  If an input variable data file is specified, variables declared as input variables and which are found (by matching names) in the specified file will receive their values from the data available in the file during the simulation. The file can either be a result file from a previous simulation (making it possible to create models that will generate input data to a different model), or an ASCII text file formatted according to Section C.3 on page 161 (making it possible to use external tools to generate input data to models).
  The file can either be specified as relative to the simulation executable file or with an absolute path.

Output


Output

☑ Save results to file
☑ Store state variables (3)
☑ Store derivatives (3)
☑ Store algebraic variables (45)
☑ Store parameters (14)
☑ Store protected variables (0)
☑ Store event points
☐ Store simulation log

Figure 4-20: Experiment output settings.

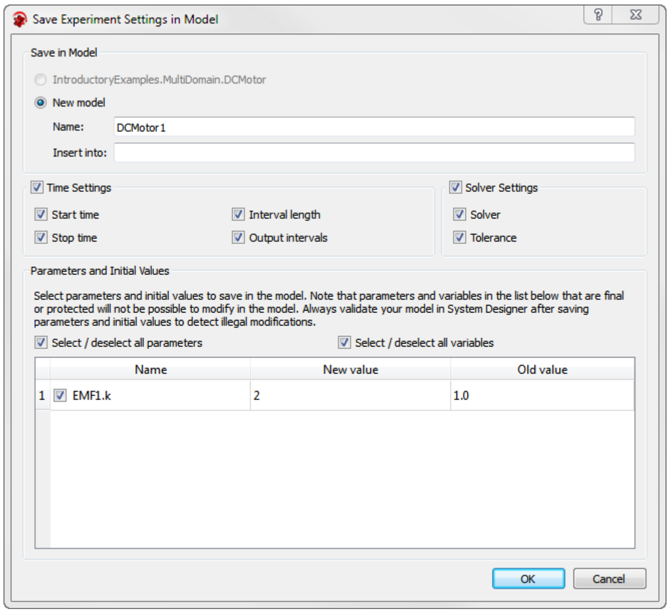
- **Save results to file**
  If enabled, the simulation result is written to file. This is the default behavior and it only makes sense to uncheck this when running long real-time simulations.

- **Store state variables**
  If enabled, all states, except protected if store protected is disabled, are stored in the result file.

- **Store derivatives**
  If enabled, all derivatives, except protected if store protected is disabled, are stored in the result file.

- **Store algebraic variables**
  If enabled, all algebraics, except protected if store protected is disabled, are stored in the result file.
- **Store parameters**
  If enabled, all parameters, except protected if store protected is disabled, are stored in the result file.
- **Store protected variables**
  If enabled, all protected variables are stored in the result file.
- **Store event points**
  If enabled, the solver outputs extra output points at event points with the value before and after the event.
- **Store simulation log**
  If enabled, variables for the number of events, number of RHS evaluations, and how real time progresses during the simulation are stored in the result file.

4.3.4 Saving Experiment Settings in Model



Save in Model

( ) IntroductoryExamples.MultiDomain.DCMotor
(●) New model
Name: DCMotor1
Insert into:

[✓] Time Settings
[✓] Start time [✓] Interval length
[✓] Stop time [✓] Output intervals

[✓] Solver Settings
[✓] Solver
[✓] Tolerance

Parameters and Initial Values

Select parameters and initial values to save in the model. Note that parameters and variables in the list below that are final or protected will not be possible to modify in the model. Always validate your model in System Designer after saving parameters and initial values to detect illegal modifications.

[✓] Select / deselect all parameters [✓] Select / deselect all variables

	Name	New value	Old value
1	[✓] EMF1.k	2	1.0

Figure 4-21: The **Save Experiment Settings in Model** dialog.

To save the experiment settings in the model choose **File ▸ Save Experiment Settings in Model**. There are two ways of saving the experiment settings in the model:

- In the original model. This option is only available if the original model is not read-only.
- In a new model. The new model is created by extending the original model.

Time settings and solver settings are saved using the experiment annotation (see Section 4.12 on page 132). Parameters and initial values are set directly or by using modifiers depending on where in the hierarchy the variable is located.

After saving the settings the model still needs to be saved in *Model Center* for them to be permanent.

4.3.5 Closing Experiments

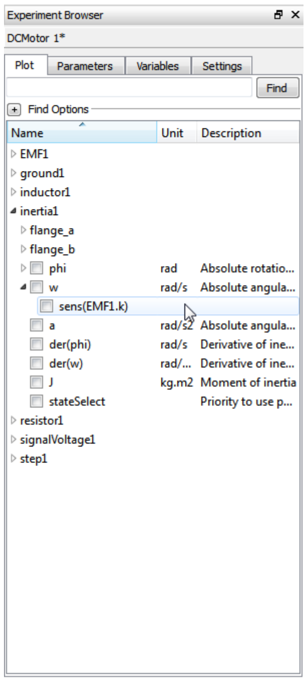
An experiment can be closed by right-clicking its title bar and choosing **Close** from the popup menu. Choose **Close All** to close all experiments. Choose **Close All But This** to close all experiments except the selected one. When closing an experiment, any plotted variables or parameters from that experiment are removed from all plot windows.

4.3.6 Duplicating Experiments

An experiment can be duplicated by right-clicking its title bar and choosing **Duplicate** from the menu. This will create a new experiment with the same parameter values, initial values, and experiment settings as the original experiment. Note that the model is not rebuilt when duplicating an experiment.

4.3.7 Sensitivity Analysis

When the CVODES solver is used it is possible to perform forward sensitivity analysis on parameters. To enable sensitivity analysis for a parameter, check its **SA** checkbox in the **Parameters** view of the **Experiment Browser**. After a simulation the result of the sensitivity analysis is available in the **Plot** view of the **Experiment Browser**. Now, each state variable is expandable and the sensitivities for that state will be listed as `sens(parameter name)`.



Name	Unit	Description
EMF1		
ground1		
inductor1		
inertia1		
flange_a		
flange_b		
phi	rad	Absolute rotatio...
w	rad/s	Absolute angula...
sens(EMF1.k)		
a	rad/s2	Absolute angula...
der(phi)	rad/s	Derivative of ine...
der(w)	rad/...	Derivative of ine...
J	kg.m2	Moment of inertia
stateSelect		Priority to use p...
resistor1		
signalVoltage1		
step1		

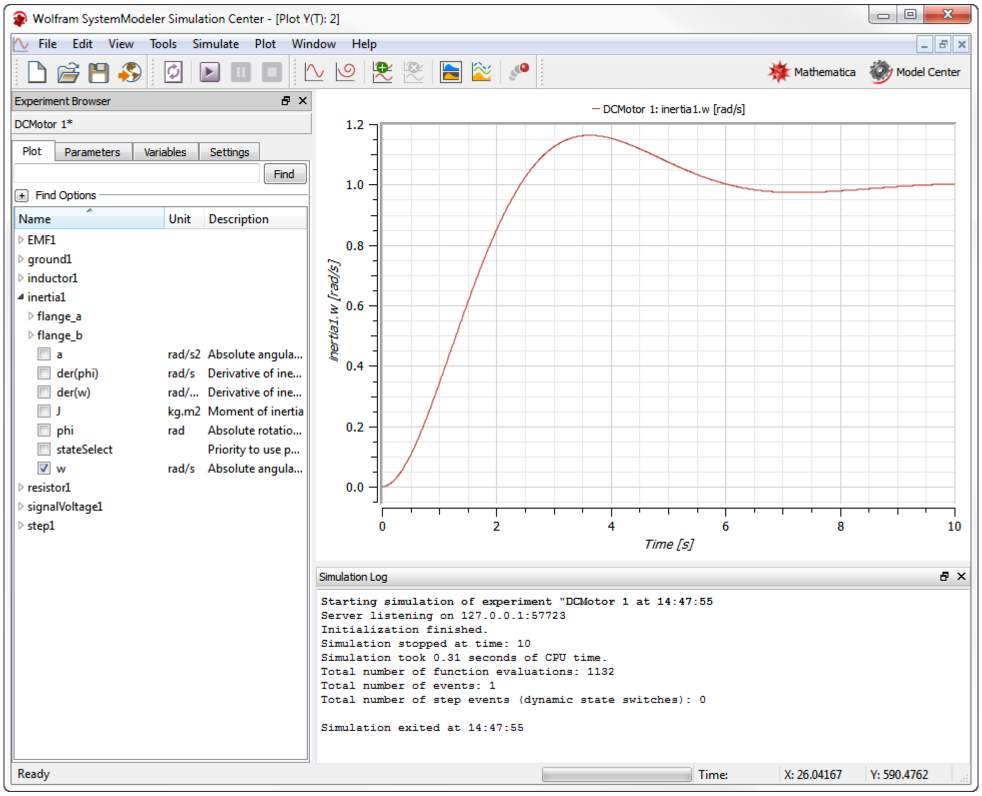
Figure 4-22: Available sensitivity result for the state `inertia1.w`.

4.3.8 Plotting Variables and Parameters

After an experiment has been simulated, the **Plot** view of the **Experiment Browser** contains all available variables and parameters of the experiment. The variables and parameters are presented in a tree view that is built up hierarchically with components as branches. Double-click a component name to expand or collapse its branch. By expanding a branch, you will be able to access the variables and parameters of the corresponding component. Each variable and parameter has a checkbox for selecting that item for plotting. The behavior when selecting a variable for plotting differs depending on the type of the active plot window. There are two different plot windows: the Y(T) plot window that is used for plotting variables and parameters versus time and the Y(X) plot window that is used for plotting a variable or parameter versus another variable or parameter.

Y(T) Plot: Time Plot

If the active window is a Y(T) plot window, see Section 4.4.2 on page 122 for information on how to create a new Y(T) plot window. The selected variable is plotted immediately when clicking the corresponding checkbox. A variable or parameter can be removed from the plot window by clicking its checkbox a second time.



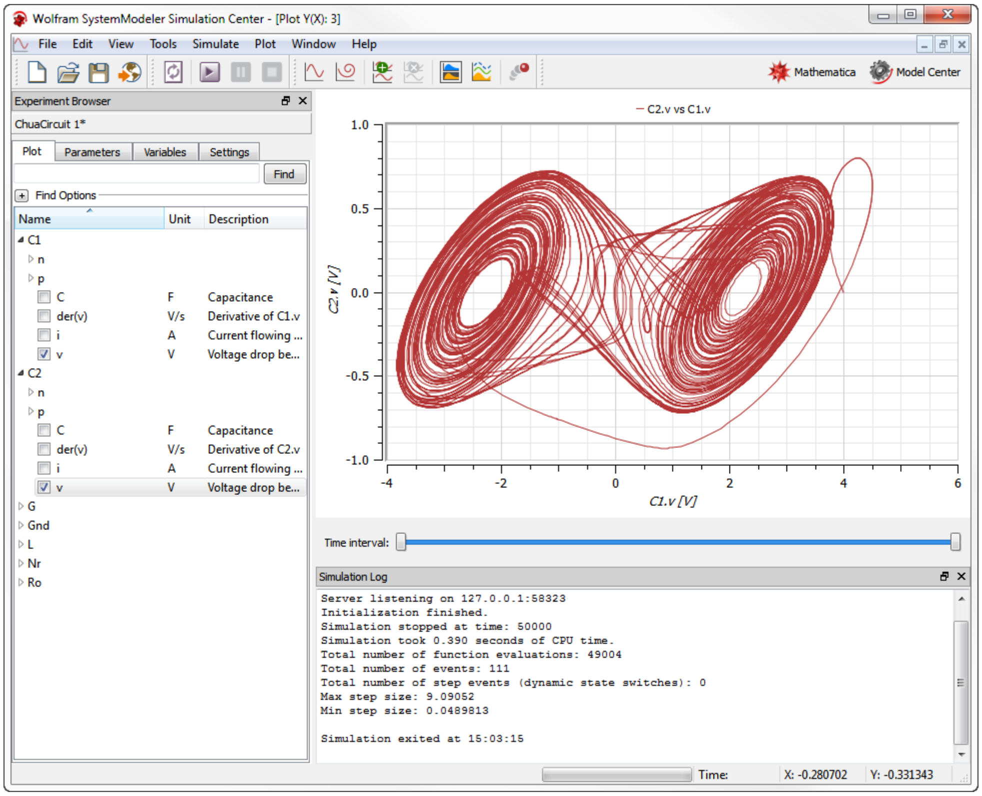
Name	Unit	Description
EMF1		
ground1		
inductor1		
inertia1		
flange_a		
flange_b		
☐ a	rad/s2	Absolute angula...
☐ der(phi)	rad/s	Derivative of ine...
☐ der(w)	rad/...	Derivative of ine...
☐ J	kg.m2	Moment of inertia
☐ phi	rad	Absolute rotatio...
☐ stateSelect		Priority to use p...
✓ w	rad/s	Absolute angula...
resistor1		
signalVoltage1		
step1		

```
Starting simulation of experiment "DCMotor 1 at 14:47:55
Server listening on 127.0.0.1:57723
Initialization finished.
Simulation stopped at time: 10
Simulation took 0.31 seconds of CPU time.
Total number of function evaluations: 1132
Total number of events: 1
Total number of step events (dynamic state switches): 0

Simulation exited at 14:47:55
```

Figure 4-23: Plotting the `inertia1.w` variable versus time from the `IntroductoryExamples.MultiDomain.DCMotor` model using the **Plot** view of the **Experiment Browser**.

Y(X) Plot: Parametric Plot

If the active window is a Y(X) plot window, see Section 4.4.2 on page 122 for information on how to create a new Y(X) plot window. The first selected parameter or variable is queued as the *x* variable for the plot. When the *y* variable is selected by clicking on a second variable or parameter, the Y(X) plot is shown in the active plot window. A variable or parameter can be removed from the plot window by clicking its checkbox a second time. When you remove a variable or parameter from a Y(X) plot window, the other variable or parameter in that Y(X) plot will also be removed.



Name	Unit	Description
C1		
n		
p		
☐ C	F	Capacitance
☐ der(v)	V/s	Derivative of C1.v
☐ i	A	Current flowing ...
✓ v	V	Voltage drop be...
C2		
n		
p		
☐ C	F	Capacitance
☐ der(v)	V/s	Derivative of C2.v
☐ i	A	Current flowing ...
✓ v	V	Voltage drop be...
G		
Gnd		
L		
Nr		
Ro		

```
Server listening on 127.0.0.1:58323
Initialization finished.
Simulation stopped at time: 50000
Simulation took 0.390 seconds of CPU time.
Total number of function evaluations: 49004
Total number of events: 111
Total number of step events (dynamic state switches): 0
Max step size: 9.09052
Min step size: 0.0489813

Simulation exited at 15:03:15
```

Figure 4-24: The chaotic behavior of the Chua circuit (`Modelica.Electrical.Analog.Examples.ChuaCircuit`) is illustrated by the parametric plot of `C2.v` versus `C1.v`.

4.4 Plot Windows

The main purpose of *Simulation Center* is to plot and study the results of a simulation. There are two kinds of plot windows: the Y(T) plot window and the Y(X) plot window. The Y(T) plot window shows how the values of one or more variables change over time, whereas the Y(X) plot window shows how the point $(x(t), y(t))$ changes over time. The plotted variables may all be from the same experiment or from different experiments. When you move the mouse cursor over a plot window the current *xy* coordinate of the cursor is shown in the status bar.

4.4.1 Arranging the Windows

If you have more than one plot window open, you can view all windows at the same time. To display the windows side by side, choose **Tile** from the **Window** menu. To arrange the windows so that the title bar of every window is visible, choose **Cascade** from the **Window** menu.

4.4.2 Creating a New Plot Window

When creating a new experiment, an empty Y(T) plot window is automatically created. If more plot windows are needed, choose **Plot ▸ New Y(T) Plot Window** or **Plot ▸ New Y(X) Plot Window** on the toolbar to create new plot windows. You can also use the buttons available on the toolbar.

Figure 4-25: The **New Y(T) Plot Window** and **New Y(X) Plot Window** buttons.

4.4.3 Creating a New Subplot

An alternative to creating new plot windows is to use a subplot, i.e. create several plots in the same window. To create a new subplot choose **Plot ▸ New Subplot** or click the **New Subplot** button on the **Plot** toolbar.

Figure 4-26: The **New Subplot** button on the **Plot** toolbar.

4.4.4 Removing a Subplot

Select the subplot by clicking somewhere in it and choose **Plot ▸ Remove Subplot** or click the **Remove Subplot** button on the **Plot** toolbar.

Figure 4-27: The **Remove Subplot** button on the **Plot** toolbar.

4.4.5 Zooming In and Out

It is possible to zoom by pressing the mouse button and dragging the mouse inside the window to create a rectangle. When releasing the mouse button, the window will zoom to the specified rectangle. To go back to the previous zoom level press the **Backspace** key. Double-clicking within the plot window resets the diagram coordinates to default.

4.4.6 Restricting the Plotted Time Interval

It is possible to restrict the plotted time interval by right-clicking in the plot window and choosing **Restrict time interval**. The plotted time interval is then controlled by adjusting the slider at the bottom of the plot window. This slider is by default visible in all Y(X) plot windows.

4.4.7 Saving Plots

To save the contents of the active plot window to a file, choose **File ▸ Export ▸ Plot as PNG** or **File ▸ Export ▸ Plot as CSV**. In the file selector window, choose a location and specify a file name. This will save the plot as either an image in PNG format or as a text file using CSV (comma-separated values).

4.4.8 Printing Plots

To print the active plot window choose **File ▸ Print**.

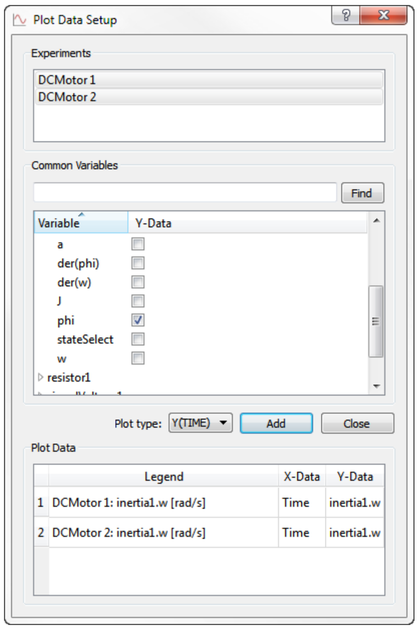
4.4.9 Copying Plots

To copy the contents of the active plot window to the clipboard as an image, choose **Edit ▸ Copy** or right-click in the plot window and choose **Copy as Image**. To copy the contents of a plot window to the clipboard as TAB delimited data, suitable for pasting into Excel, for example, right-click in the plot window and choose **Copy as Data**.

4.4.10 Plot Data Setup

To open the **Plot Data Setup** dialog, choose **Plot ▸ Plot Data Setup**. From this dialog it is possible to plot the same variable from several experiments. First select the experiments of interest in the list of experiments. The **Common Variables** view will then show the common set of variables in the selected experiments. The experiments do not need to be from the same model; they only have to contain some common variable. Select the variables to plot by checking their **Y-Data** checkbox and pressing the **Add** button. This will add one plot for each variable from all of the selected experiments. It is also possible to create Y(X) plots from this dialog by selecting Y(X) from the plot type combo box.

The plot data view in the dialog shows all datasets from the plot. It is possible to remove datasets from the plot by selecting them and pressing the **Delete** key.



Experiments: DCMotor1, DCMotor2

Variable	Y-Data
a	☐
der(phi)	☐
der(w)	☐
J	☐
phi	✓
stateSelect	☐
w	☐
▷ resistor1	

Plot type: Y(TIME)

	Legend	X-Data	Y-Data
1	DCMotor1: inertia1.w [rad/s]	Time	inertia1.w
2	DCMotor2: inertia1.w [rad/s]	Time	inertia1.w

Figure 4-28: The **Plot Data Setup** dialog.

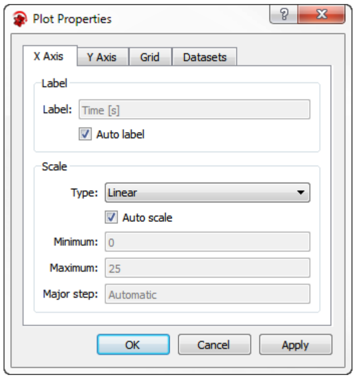
4.4.11 Plot Properties

To change the properties of a plot, choose **Plot ▸ Properties** or right-click in the plot window and choose **Properties**. This will open the **Plot Properties** dialog.

Axis Properties

To change the axis properties for a plot window, open the **Plot Properties** dialog and select the appropriate axis tab. For quick access to the axis properties it is possible to double-click on the axis in the plot window. Here you can specify an axis label and set up the axis scale. An automatic label will be generated if the **Auto label** checkbox is checked. To specify an axis label, uncheck the **Auto label** checkbox and write the label text in the **Label** text box.

If the plot is in a plot window that contains subplots, it is possible to enable **Synchronize with subplots**. When that is enabled, all scale changes to the selected axis will be propagated to the other subplots.



Label

Label: Time [s]

☑ Auto label

Scale

Type: Linear

☑ Auto scale

Minimum: 0

Maximum: 25

Major step: Automatic

Figure 4-29: The **Plot Properties** dialog box with the **X Axis** tab opened.

Grid Properties

To change the grid properties for a plot window, open the **Plot Properties** dialog and select the **Grid** tab. Here you can turn on and off the different grid lines as well as change their line style and color.

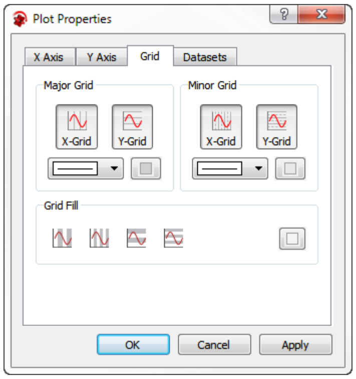
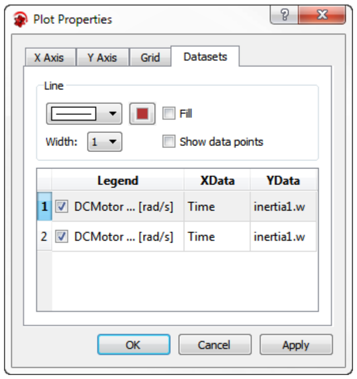


Figure 4-30: The **Plot Properties** dialog box with **Grid** tab opened.

Dataset Properties

To change the dataset properties for a plot window, open the **Plot Properties** dialog and select the **Datasets** tab. For quick access to the dataset properties, click on the legend item for the dataset that you want to edit. Here you can specify line properties for each dataset. To change the same property for multiple datasets, press and hold down the **Ctrl** key and mark multiple datasets by clicking on the desired datasets. To edit the legend text for a dataset, double-click in the **Legend** column for that dataset. To turn off the legend for a dataset, uncheck the checkbox in the **Legend** column for that dataset.



	Legend	XData	YData
1	✓ DCMotor ... [rad/s]	Time	inertia1.w
2	✓ DCMotor ... [rad/s]	Time	inertia1.w

Figure 4-31: The **Plot Properties** dialog box with the **Datasets** tab opened.

4.4.12 Plot Themes

To apply predefined themes to a plot, click the **Plot Themes** button on the **Plot** toolbar, or click **Plot Themes** in the **Plot** menu. In the **Select Plot Theme** dialog that opens, select one of the themes to apply it to the plot.

Figure 4-32: The **Plot Themes** button on the **Plot** toolbar.

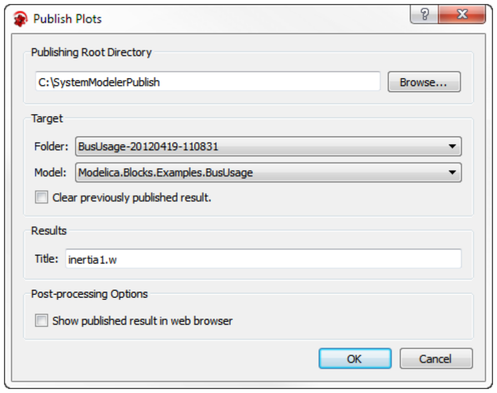
4.4.13 Removing All Plotted Variables

To remove all plotted variables from a plot window, right-click in the plot window and choose **Clear Plot** from the popup menu that appears.

4.4.14 Publishing Plots

Using the publisher tool from within *Model Center*, it is possible to generate documentation for Modelica classes that can be viewed in any web browser; see Section 3.1.9 on page 27 for more information on how to publish classes. Using the publisher tool from within

Simulation Center, it is possible to publish plots to accompany the published documentation. To publish the active plot window choose **File ▸ Publish Plots**. This will open the publisher configuration dialog box.



Folder: BusUsage-20120419-110831

Model: Modelica.Blocks.Examples.BusUsage

☐ Clear previously published result.

Title: inertia1.w

☐ Show published result in web browser

C:\SystemModelerPublish

Figure 4-33: The publisher configuration dialog box.

- **Publishing Root Directory**
  The directory in which all files and folders generated by the publisher are saved; it should be set to the directory that was used when publishing from *Model Center*.
- **Target Folder**
  Selects the folder within the publishing root directory where the plot should be published.
- **Target Model**
  Selects which model the plot should be associated with.
- **Title**
  A custom title for the plot; by default it will be set to a comma-separated list of all the variables in the plot.
- **Show published result in web browser**
  Shows the published model in the default web browser.

4.5 3D Animation

If the current experiment uses any components from the `MultiBody` library that have a visual representation, it is possible to view a 3D animation of the model as shown in Figure 4-34. To view the animation choose **View ▸ Animation**.

To start the animation press the play button in the toolbar within the **Animation** view. When the animation is running it is possible to pause it by pressing the **Pause** button or stop it by pressing the **Stop** button. It is also possible to change the time scale of the animation by selecting the desired time scale from the time scale combo box or by entering a custom time scale and pressing the **Enter** key.

If the animation window is open while a real-time simulation is running, the toolbar will be disabled and the animation will run synchronized with the real-time simulation.

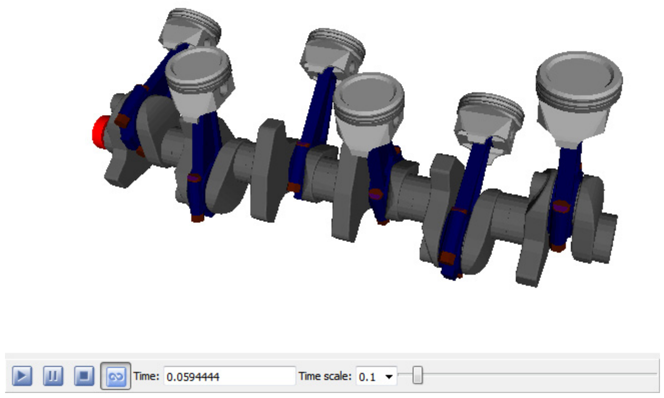


Figure 4-34: Animation view of `Modelica.Mechanics.MultiBody.Examples.Loops.EngineV6_analytic`.

4.5.1 Navigation

It is possible to rotate, pan, and zoom in/out the 3D model by using the mouse. To rotate the 3D model, click and hold the left mouse button and drag the mouse in the desired rotation direction. To pan the 3D model, press and hold down the **Ctrl** key. Then click and hold the left mouse button and drag the mouse in the desired pan direction. To zoom in or out, rotate the mouse scroll wheel backward or forward. To reset the camera position and fit the 3D model to the window, right-click and select **Fit to Window**.

It is possible to attach the camera to a specific object by right-clicking and selecting **Attach Camera to**. The camera will then maintain its relative position to that object when the object moves.

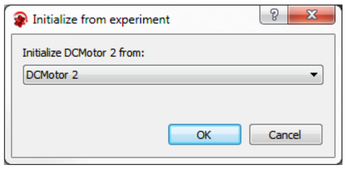
4.6 Initializing Experiments

4.6.1 To Steady State

To initialize the current experiment to steady state choose **Tools ▸ Initialize ▸ To Steady State**. This will run a steady state solver that tries to solve for all state derivatives equal to zero. The current initial values will be used as start values for the steady state solver and can affect the solution. If the solver succeeds, the experiment is initialized to start in that state. Note this does not work for models that have any state where the fixed attribute is set to true.

4.6.2 From Experiment

To initialize the current experiment to the end state of another experiment, or itself, choose **Tools ▸ Initialize ▸ From Experiment**. This will bring up the **Initialize from experiment** dialog; select the desired experiment from which to initialize. This will set all initial values of the current experiment to the value that the corresponding variable had at the last time instance in the experiment from which we are initializing.



Initialize DCMotor 2 from:
DCMotor 2

Figure 4-35: The **Initialize from experiment** dialog.

4.7 FFT Analysis

To do a FFT analysis choose **Tools ▸ FFT Analysis**; this will bring up the **FFT Analysis** dialog. To select a variable for FFT analysis switch to the **Plot** view of the **Experiment Browser**. Then click and hold the right mouse button on the desired variable and drag it into the drop area in the **FFT Analysis** dialog. It is also possible to right-click on a variable in a plot window and choose **FFT Analysis**.

4.7.1 Options

If **Remove DC component** is checked, the mean value of the signal is removed before the FFT is done.

By default the FFT is performed over the time interval where the variable is defined. However, it is possible to define a custom time window by changing **Time start** and **Time end**.

Since the solver generally does not give equidistant output, the signal needs to be re-sampled. When **Sampling Interval** is unchecked, it will be based on the length of the time window divided by the number of output steps in that time window. The result is rounded down so that it is coherent with the next larger fast FFT size. This approach only works if the output is almost equidistant; if the model has a large amount of events that are clustered it is advisable to select a custom sampling interval.

If a custom sampling interval is entered, the number of FFT points is calculated from the chosen sampling interval and vice versa. By using a custom sampling interval and a custom number of FFT points, it is possible to do a zero-padded FFT. The resulting plot will be cut off at the selected cut-off frequency.

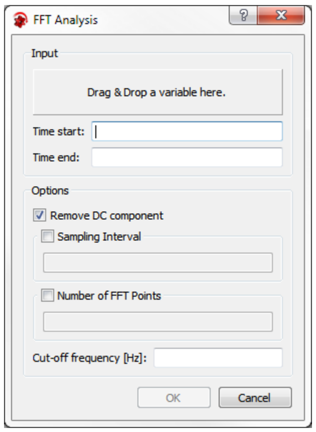


Figure 4-36: FFT Analysis dialog.

4.8 Simulation Log Window

The simulation log is shown in the simulation log window at the bottom of *Simulation Center*. Any errors in the translation or simulation of a model will appear in this window.

4.9 Source View

The generated C code can be viewed by choosing **View ▸ Generated Source Code**; this will bring up the source view for the active experiment. To search for a string in the source view choose **Edit ▸ Find**; this will bring up a find dialog. Type in the search string and press **Enter** to go to the next match.

4.10 Exporting

4.10.1 CSV File

To export the simulation result from the active experiment as a CSV file choose **File ▸ Export ▸ Experiment as CSV**. In the file selector window, choose a location and specify a file name. This will save the simulation result as a CSV file.

4.10.2 *SystemModeler* Result File

To export the simulation result from the active experiment as a *SystemModeler* result file choose **File ▸ Export ▸ Experiment as MAT**. In the file selector window, choose a location and specify a file name. This will save the simulation result as a *SystemModeler* result file. The file format is described in detail in the appendix "Simulation Result Files" on page 155.

4.10.3 Simulation Executable

To export the simulation executable and the corresponding settings file from the active experiment choose **File ▸ Export ▸ Simulation executable** and select the desired output directory. Two files will be created in that directory: ExperimentName.exe, which is the simulation executable, and ExperimentName.sim, which is the settings file. The settings file contains all parameter values and initial values for the model. The simulation executable takes three switches: `-f` to specify which settings file to use, `-r` to specify the result file name, and `-server` to specify an IP address and a port if the simulation should be run via external application. The communication between the simulation and external applications is described in detail in Appendix D.The last two arguments are optional. For example,

```
MyExperiment.exe -f MyExperiment.sim -r ResultFile.mat
```

will run MyExperiment.exe using the MyExperiment.sim settings file and the result will be stored in ResultFile.mat.

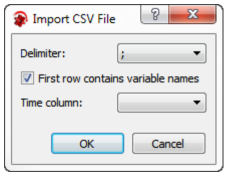
```
MyExperiment.exe -f MyExperiment.sim -server "127.0.0.1:7000"
```

will run MyExperiment.exe using the MyExperiment.sim settings file and listening to IP address 127.0.0.1 and port 7000.

4.11 Importing

4.11.1 CSV Files

Simulation Center can load externally produced data in the form of CSV files. To import a CSV file choose **File ▸ Open** and select the CSV file. A CSV import dialog will appear that lets you specify the separator used in the file (comma, semicolon, tab, or space) and which column should be interpreted as time. The imported data will appear as a new experiment with the **Plot** view as the only available view in the **Experiment Browser**.


Import CSV File

Delimiter: ;

[✓] First row contains variable names

Time column:

Figure 4-37: The **Import CSV File** dialog box.

4.11.2 *SystemModeler* Result Files

Simulation Center can open *SystemModeler* result files exported from *Simulation Center*. To import a *SystemModeler* result file choose **File ▸ Open** and select the MAT file. The imported results will appear as a new experiment with the **Plot** view as the only available view in the **Experiment Browser**.

4.12 Experiment Annotation

To define default experiment settings, a model can have the experiment annotation. The experiment annotation is added to the model in the **Modelica Text View** of *Model Center*.

```
annotation(experiment(StartTime=VALUE,
                      StopTime=VALUE,
                      Interval=VALUE,
                      NumberOfIntervals=VALUE,
                      Tolerance=VALUE,
                      Algorithm="NAME"));
```

- `StartTime` specifies the experiment start time.
- `StopTime` specifies the experiment stop time.
- `Interval` specifies the interval length.
- `NumberOfIntervals` specifies the number of output intervals the solver should generate.
- `Tolerance` specifies the tolerance used by the solver.
- `Algorithm` specifies the solver that should be used.

Note that `Interval` and `NumberOfIntervals` cannot be used at the same time.

For example, if we want the default experiment for our model to start at *t*=0, end at *t*=20, and use a tolerance of 0.00001, we add the following experiment annotation:

```
annotation(experiment(StartTime=0,
                      StopTime=20,
                      Tolerance=0.00001));
```

Appendix A: **Keyboard Shortcuts**

This appendix lists all the keyboard key combinations and keyboard shortcuts available in *SystemModeler*.

A.1 *Model Center*

The keyboard key combinations and keyboard shortcuts available in *Model Center* are listed in this section.

A.1.1 Common Keyboard Shortcuts

- **CTRL + O**
  Open a Modelica file (.mo).
- **CTRL + S**
  Save the class in the active class window, or if the **Class Documentation Browser** is open and is active, the class of the **Class Documentation Browser**.
- **CTRL + Q**
  Quit *Model Center*.
- **CTRL + F**
  Activate the find text box in the **Class Browser** window, or if the **Modelica Text View** is active, open the find panel.
- **CTRL + ALT + B**
  Show the **Class Browser** window.
- **CTRL + ALT + D**
  Show the **Class Documentation Browser** window.
- **CTRL + ALT + O**
  Show the **Components** window.

- **CTRL + ALT + P**
  Show the **Parameters** view.
- **CTRL + ALT + V**
  Show the **Variables** view.
- **CTRL + ALT + S**
  Show the **Constants** view.
- **CTRL + ALT + M**
  Show the **Messages** view.
- **CTRL + ALT + K**
  Show the **Kernel Command** view.
- **CTRL + TAB**
  Change the active class window.

A.1.2 Class Browser Window

- **DELETE**
  Delete selected class, or if the class is a top-level class, unload the class.
- **CTRL + C**
  Copy selected class to the clipboard.
- **CTRL + V**
  Paste class on the clipboard, if any, into the selected class.
- **ENTER**
  Open selected class in a class window.
- **F2**
  Rename selected class.
- **SHIFT + F1**
  Show documentation of the selected class.
- **LEFT ARROW**
  If selected class is a package, collapse the branch of the tree and hide the contents of the package. If the branch is already collapsed, select the parent package, if any.
- **RIGHT ARROW**
  If selected class is a package, expand the branch of the tree and show the contents of the package. If the branch is already expanded, select the first class in the package.
- **UP ARROW**
  Select class above the currently selected class.

- **DOWN ARROW**
  Select class below the currently selected class.

A.1.3 Class Window

All Views

- **CTRL** + **P**
  Print contents of the view.
- **CTRL** + **Z**
  Undo last operation.
- **CTRL** + **Y**
  Redo last operation.
- **CTRL** + **W**
  Close active class window.
- **CTRL** + **1**
  Switch to the **Icon View**.
- **CTRL** + **2**
  Switch to the **Diagram View**.
- **CTRL** + **3**
  Switch to the **Modelica Text View**.
- **CTRL** + **SHIFT** + **V**
  Validate visualized class.

Icon and Diagram Views

- **DELETE**
  Delete selected objects.
- **CTRL** + **C**
  Copy selected graphic items to the clipboard.
- **CTRL** + **X**
  Cut selected graphic items to the clipboard.
- **CTRL** + **V**
  Paste clipboard contents into the view.
- **CTRL** + **A**
  Select all objects.

- **CTRL** + **D**
  Duplicate selected objects.
- **CTRL** + **SHIFT** + **W**
  Zoom the view to fit the size of the window.
- **SPACE**
  Pointer Tool.
- **C**
  Connection Line Tool.
- **L**
  Line Tool.
- **R**
  Rectangle Tool.
- **E**
  Ellipse Tool.
- **P**
  Polygon Tool.
- **T**
  Text Tool.
- **B**
  Bitmap Tool.
- **CTRL** + **L**
  Rotate selected objects 90º to the left (counterclockwise).
- **CTRL** + **R**
  Rotate selected objects 90º to the right (clockwise).
- **CTRL** + **SHIFT** + **H**
  Flip selected objects horizontally.
- **CTRL** + **SHIFT** + **J**
  Flip selected objects vertically.
- **ENTER**
  If a single component is selected, open the component in component mode. If a single graphic item is selected, edit the properties of the object.
- **F2**
  If a single component is selected, rename the component.

- **SHIFT** + **F1**
  If a single component is selected, show the documentation of its class. If no components are selected, show the documentation of the visualized class.
- **ESC**
  Deselect all objects, or if in component mode, exit component mode.
- **LEFT ARROW**
  Move selected objects left. If no selected objects exist, move left in the graphical view.
- **RIGHT ARROW**
  Move selected objects right. If no selected objects exist, move right in the graphical view.
- **UP ARROW**
  Move selected objects up. If no selected objects exist, move up in the graphical view.
- **DOWN ARROW**
  Move selected objects down. If no selected objects exist, move down in the graphical view.
- **CTRL** + **LEFT ARROW**
  Move to the left end of the graphical view.
- **CTRL** + **RIGHT ARROW**
  Move to the right end of the graphical view
- **CTRL** + **UP ARROW**
  Move to the top of the graphical view.
- **CTRL** + **DOWN ARROW**
  Move to the bottom of the graphical view.
- **PAGE UP**
  Move up in the graphical view.
- **PAGE DOWN**
  Move down in the graphical view.
- **SHIFT** + **PAGE UP**
  Move left in the graphical view.
- **SHIFT** + **PAGE DOWN**
  Move right in the graphical view.

Modelica Text View

- **CTRL** + **Z**
  Undo last text edit.

- **CTRL** + **Y**
  Redo last text edit.
- **CTRL** + **X**
  Cut selected text to the clipboard.
- **CTRL** + **C**
  Copy the selected text to the clipboard.
- **CTRL** + **V**
  Paste clipboard contents into the **Modelica Text View**.
- **DELETE**
  Delete selected text, or if no text is selected, delete the character to the right.
- **CTRL** + **F**
  Find text.
- **CTRL** + **G**
  Go to line.
- **CTRL** + **A**
  Select all text.
- **SHIFT** + **Enter**
  Apply any changes made in the **Modelica Text View**.

A.1.4 View Window

Parameters View

Most of the shortcuts and keys described in this section are only valid when one of the text fields of the **Parameters** view is activated, i.e. when the text cursor is visible in one of the text fields.

- **CTRL** + **Z**
  Undo last text edit.
- **CTRL** + **Y**
  Redo last text edit.
- **CTRL** + **X**
  Cut selected text to the clipboard.
- **CTRL** + **C**
  Copy selected text to the clipboard.

- **CTRL + V**
  Paste clipboard contents into the text field.
- **DELETE**
  Delete selected text, or if no text is selected, delete the character to the right.
- **CTRL + A**
  Select all text.
- **ENTER**
  Confirm the parameter value in the active text field and move to the next field, if any.
- **TAB**
  Confirm the parameter value in the active text field and move to the next field, if any.

Variables View

Most of the shortcuts and keys described in this section are only valid when one of the text fields of the **Variables** view is activated, i.e. when the text cursor is visible in one of the text fields.

- **CTRL + Z**
  Undo last operation.
- **CTRL + Y**
  Redo last operation.
- **CTRL + X**
  Cut selected text to the clipboard.
- **CTRL + C**
  Copy selected text to the clipboard.
- **CTRL + V**
  Paste clipboard contents into the text field.
- **DELETE**
  Delete selected text, or if no text is selected, delete the character to the right.
- **CTRL + A**
  Select all text.
- **ENTER**
  Confirm the variable value in the active text field and move to the next field, if any.
- **TAB**
  Confirm the variable value in the active text field and move to the next field, if any.

Constants View

Most of the shortcuts and keys described in this section are only valid when one of the text fields of the **Constants** view is activated, i.e. when the text cursor is visible in one of the text fields.

- **CTRL** + **Z**
  Undo last operation.
- **CTRL** + **Y**
  Redo last operation.
- **CTRL** + **X**
  Cut selected text to the clipboard.
- **CTRL** + **C**
  Copy selected text to the clipboard.
- **CTRL** + **V**
  Paste clipboard contents into the text field.
- **DELETE**
  Delete selected text, or if no text is selected, delete the character to the right.
- **CTRL** + **A**
  Select all text.
- **ENTER**
  Confirm the constant value in the active text field and move to the next field, if any.
- **TAB**
  Confirm the constant value in the active text field and move to the next field, if any.

Kernel Command View: Output View

- **CTRL** + **C**
  Copy selected text to the clipboard.
- **CTRL** + **A**
  Select all text.

Kernel Command View: Input Field

- **CTRL** + **Z**
  Undo last text edit.
- **CTRL** + **Y**
  Redo last text edit.

- **CTRL + X**
  Cut selected text to the clipboard.
- **CTRL + C**
  Copy selected text to the clipboard.
- **CTRL + V**
  Paste clipboard contents into the text field.
- **DELETE**
  Delete selected text, or if no text is selected, delete the character to the right.
- **CTRL + A**
  Select all text.
- **ENTER**
  Evaluate given command.
- **UP ARROW**
  Step backward in the history of commands.
- **DOWN ARROW**
  Step forward in the history of commands.

A.1.5 Components Window

- **DELETE**
  Delete selected components.
- **CTRL + A**
  Select all components.
- **ENTER**
  If a single component is selected, open the component in component mode.
- **F2**
  If a single component is selected, rename the component.
- **SHIFT + F1**
  If a single component is selected, show the documentation of its class. If a single class is selected, show the documentation of the class.
- **UP ARROW**
  Select the component above the currently selected component. If a graphical view is active in the class window, the component is also selected in that view.

- **DOWN ARROW**
  Select the component below the currently selected component. If a graphical view is active in the class window, the component is also selected in that view.

A.1.6 Class Documentation Browser Window

- **CTRL + Z**
  Undo last text edit (edit mode only).
- **CTRL + Y**
  Redo last text edit (edit mode only).
- **CTRL + P**
  Print current page.
- **CTRL + X**
  Cut selected text to the clipboard (edit mode only).
- **CTRL + C**
  Copy selected text to the clipboard.
- **CTRL + V**
  Paste clipboard contents into editor (edit mode only).
- **CTRL + E**
  Switch between edit and view mode.
- **CTRL + PLUS**
  Zoom in (view mode only).
- **CTRL + MINUS**
  Zoom out (view mode only).
- **BACKSPACE**
  View next page in the list of visited pages (view mode only).
- **SHIFT + BACKSPACE**
  View previous page in the list of visited pages (view mode only).
- **ESC**
  Close window.

A.2 Simulation Center

This section lists the keyboard key combinations and keyboard shortcuts available in the *Simulation Center*.

A.2.1 Common Keyboard Shortcuts

- **CTRL + N**
  Create a new experiment.
- **CTRL + O**
  Open an experiment file (.sme), a result file (.mat), or import a CSV file (.csv).
- **CTRL + S**
  Save the currently active experiment.
- **CTRL + Q**
  Quit *Simulation Center*.
- **CTRL + P**
  Print the active plot window.
- **CTRL + C**
  Copy the active plot window as an image to the clipboard.
- **CTRL + F**
  Find text in the source view.
- **CTRL + ALT + E**
  Toggle visibility of the **Experiment Browser**.
- **CTRL + ALT + L**
  Toggle visibility of the log window.
- **CTRL + R**
  Simulate current experiment.
- **CTRL + SHIFT + R**
  Rebuild current experiment.
- **CTRL + W**
  Close current window.

A.2.2 Animation Window

- **CTRL + SHIFT + W**
  Fit to window.
- **SPACE**
  Toggle play/pause.
- **LEFT ARROW**
  Skip backward.

- **RIGHT ARROW**
  Skip forward.

Appendix B: **Kernel Commands**

This appendix lists all the kernel commands that can be used in the **Kernel Command** view of *Model Center*.

- `cd()`
  Returns the current directory.
- `cd(`*`dir`*`)`
  Changes directory to the directory given as a string.
- `checkModel(`*`classname`*`)`
  Validates the given class and returns a string with the result.
- `clear()`
  Clears all class definitions currently loaded.
- `clearVariables()`
  Clears all variables defined in the kernel command session.
- `getVersion()`
  Returns the version number of the *SystemModeler* kernel.
- `instantiateModel(`*`classname`*`)`
  Performs code instantiation of a class and returns a string containing the flat class definition.
- `list()`
  Returns a string containing all loaded class definitions.
- `list(`*`classname`*`)`
  Returns a string containing the class definition of the named class.
- `listVariables()`
  Returns a vector of the names of the variables currently defined in the kernel session.
- `loadModel(`*`classname`*`)`
  Loads the class with the given name from the *SystemModeler* library path. Use with care as this function does not update the internal structure of *Model Center*. Loading a

class that is already loaded may lead to critical errors.

- `loadFile(`*`filename`*`)`
  Loads the Modelica file with the name given as a string. Use with care as this function does not update the internal structure of *Model Center*. Loading a class that is already loaded may lead to critical errors.
- `readFile(`*`filename`*`)`
  Loads the file given as a string and returns a string containing the file content. Use with care as this function does not update the internal structure of *Model Center*. Loading a class that is already loaded may lead to critical errors.
- `timing(`*`expr`*`)`
  Evaluates the specified expression and returns the elapsed time in seconds.
- `saveTotalModel(`*`filename`*`, `*`classname`*`)`
  Saves a total class definition of the named class to the file given as a string. The file will contain all class definitions used by the named class, making it a self-contained file.
- `typeOf(`*`var`*`)`
  Returns the type of a kernel session variable as a string.

Appendix C: **File Formats**

This appendix contains information on the various file formats used by *SystemModeler*.

C.1 Simulation Settings Files

The simulation settings file (.sim) is used for passing experiment settings, initial values, and parameter values to the simulation. The file is an XML file and all valid tags are described in this chapter. Attributes that are marked with [read-only] are not meant to be changed by the user; they only convey information about the model. An example file, a simulation settings file for the `IntroductoryExamples.HelloWorld` model, is shown below.

```
<?xml version="1.0"?>
<!DOCTYPE simulation SYSTEM "simulation_settings.dtd">

<simulation version="1.0" name="mmsd_1249473736_41" start="0.0" end="10.0" stepSize="0"
outputSteps="-1" tolerance="1e-6" method="dassl" filePath="mmsd_1249473736_41.exe"
resultFilePath="mmsd_1249473736_41.mat">

<model name="IntroductoryExamples.HelloWorld" description="" numOfVariables="4">
  <variable name="$dummy" value="0.0" defaultValue="0.0" type="Real" kind="STATE"
  direction="BIDIR" unit="" index="1" description="" equationBound="" protected="false" final="false"/
  >
  <variable name="der($dummy)" value="0.0" defaultValue="0.0" type="Real" kind="DERIVATIVE"
  direction="BIDIR" unit="" index="1" description="" equationBound="" protected="false" final="false"/
  >
  <variable name="der(x)" value="1.0" defaultValue="1.0" type="Real" kind="DERIVATIVE"
  direction="BIDIR" unit="" index="0" description="" equationBound="" protected="false" final="false"/
  >
  <variable name="x" value="1.0" defaultValue="1.0" type="Real" kind="STATE" direction="BIDIR"
  unit="" index="0" description="" equationBound="" protected="false" final="false"/>
</model>

<Options>
  <Option name="Solver">
  <OptionValue name="MaxStepSize" value="0">
  </OptionValue>
  <OptionValue name="EventHysteresisEpsilon" value="1e-10">
  </OptionValue>
```

```
    <OptionValue name="NonLinearSolverTolerance" value="1e-12">
    </OptionValue>
    <OptionValue name="MaxNoInternalSteps" value="0">
    </OptionValue>
    </Option>
    <Option name="CheckForMultipleMixedSolutions">
    <OptionValue name="enable" value="false">
    </OptionValue>
    </Option>
    <Option name="CheckMinMax">
    <OptionValue name="enable" value="false">
    </OptionValue>
    </Option>
    <Option name="StopAtSteadyState">
    <OptionValue name="enable" value="false">
    </OptionValue>
    </Option>
    <Option name="OutputSettings">
    <OptionValue name="StoreStates" value="true">
    </OptionValue>
    <OptionValue name="StoreDerivatives" value="true">
    </OptionValue>
    <OptionValue name="StoreAlgebraics" value="true">
    </OptionValue>
    <OptionValue name="StoreParameters" value="true">
    </OptionValue>
    <OptionValue name="StoreConstants" value="false">
    </OptionValue>
    <OptionValue name="StoreProtected" value="true">
    </OptionValue>
    <OptionValue name="StoreEventPoints" value="true">
    </OptionValue>
    <OptionValue name="FormatDoublePrecision" value="true">
    </OptionValue>
    </Option>
  </Options>

  </simulation>
```

C.1.1 Element Simulation

The contents of a `simulation` element should be a `model` element and optionally an `Options` element. The attributes of the `simulation` element describe the basic experiment settings.

Attributes

- `version`
  The .sim file format version.
- `name`
  The display name of the simulation.

- `start`
  Specifies the start time of simulation.
- `end`
  Specifies the end time of simulation.
- `stepSize`
  Specifies the length of the interval between output points from the solver. When set, `outputSteps` must be equal to zero.
- `outputSteps`
  Specifies the number output intervals that the solver generates. To get `Automatic`, set it to `-1`. To use `stepSize` instead, set it to `0`.
- `tolerance`
  Specifies the local tolerance that is used in each solver step.
- `method`
  Specifies the solver used to solve the dynamic system; possible values are DASSL or CVODES.
- `filePath`
  The path to the simulation executable.
- `resultFilePath`
  The path to the result file.

C.1.2 Element Model

The contents of the `model` element should be the number of `variable` elements specified by the attribute `numOfVariables`.

Attributes

- `name` [read-only]
  Model name.
- `description` [read-only]
  Model description.
- `numOfVariables` [read-only]
  The number of `variable` elements in the model.

C.1.3 Element Variable

The `variable` element does not have any content; all information is stored as attributes.

Attributes

- `name` [read-only]
  Variable name.
- `value`
  The value of the variable.
- `defaultValue` [read-only]
  The original value of the variable in the Modelica code.
- `type` [read-only]
  The type of the variable (`Real`, `Integer`, `Boolean`, `String`).
- `kind` [read-only]
  Specifies the variable kind (`VARIABLE`, `STATE`, `DUMMY_DER`, `DUMMY_STATE`, `DISCRETE`, `PARAM`, `CONSTANT`, `DERIVATIVE`).
- `direction` [read-only]
  The direction of the variable (`BIDIR`, `INPUTDIR`, `OUTPUTDIR`).
- `unit` [read-only]
  The unit of the variable.
- `index` [read-only]
  For internal use.
- `description` [read-only]
  The description of the variable.
- `equationBound` [read-only]
  The equation that is used to calculate the start value of the variable (if `useEquation` is `true`).
- `useEquation`
  Indicates if the bound equation should be used to calculate the start value of the variable or if the value attribute should be used.
- `protected` [read-only]
  Flag that specifies if the variable is protected or not.
- `final` [read-only]
  Flag that specifies whether the variable has a final modifier.

C.1.4 Element Options

The `Options` element should contain one or more `Option` elements.

C.1.5 Element Option

The `Option` element should contain one or more `OptionValue` elements. Valid option names are:

- `Solver`
  Valid `OptionValue` names:
  - `MaxStepSize`
  - `EventHysteresisEpsilon`
  - `NonLinearSolverTolerance`
  - `MaxNoInternalSteps`
- `CheckForMultipleMixedSolutions`
  Valid `OptionValue` names:
  - `enable`
- `CheckMinMax`
  Valid `OptionValue` names:
  - `enable`
- `StopAtSteadyState`
  Valid `OptionValue` names:
  - `enable`
- `OutputSettings`
  Valid `OptionValue` names:
  - `StoreStates`
  - `StoreDerivatives`
  - `StoreAlgebraics`
  - `StoreParameters`
  - `StoreProtected`
  - `StoreEventPoints`

Attributes

- `name`
  The name of this `Option`.

C.1.6 Element OptionValue

Attributes

- `name`
  The name of this `OptionValue`.
- `value`
  The value of this `OptionValue`.

C.2 Simulation Result Files

Simulation result files (.mat) are generated during simulation. The files are read by *Simulation Center*, but may also be accessed from within *Mathematica* to extract specific values or datasets.

At the end of each simulation, the results of the simulation are stored in a level-4 MAT-file according to this specification. This binary file has a structure that can be interpreted as a sequence of matrices: `Aclass`, `name`, `description`, `dataInfo`, `data_1`, ..., `data_n`.

There are two different ways in which the matrices may be stored.

- `binNormal`: matrix is stored as a non-transposed binary matrix.
- `binTrans`: matrix is stored as a transposed binary matrix.

The first matrix, `Aclass`, is always stored in the `binNormal` format by *SystemModeler*. The format of the rest of the matrices is given by the `Aclass` matrix; see page 157.

There are four informative matrices, followed by one or more data matrices containing the values for each simulation variable. Every matrix has a header with a specific structure that is described in detail in this chapter.

The contents of the MAT-file is a set of matrices, where each matrix is preceded by a header.

C.2.1 Headers

Each matrix is preceded by a header with six entries. The first five entries contain information about what data is stored and how, while the sixth entry is the name of the matrix.

type

An integer with storage information. If the integer is represented as `MOPT`, where `M` is the thousands digit, `O` is the hundreds digit, `P` is the tens digit, and `T` is the ones digit, then:

- `M` indicates the numeric format of binary numbers on the machine that wrote the file. It is possible to use the table below to determine the number to use for your machine. This should be the same for all the headers in the MAT-file.

`0`	IEEE Little Endian (PC, 386, 486, DEC Risc)

`1`	IEEE Big Endian (Macintosh SPARC, Apollo, SGI, HP 9000/300, other Motorola systems)
`2`	VAX D-float
`3`	VAX G-float
`4`	Cray

- `O` is always 0 (zero) and is reserved for future use.
- `P` specifies the format of the data according to the table below.

`0`	double-precision (64-bit) floating-point numbers
`1`	single-precision (32-bit) floating-point numbers
`2`	32-bit signed integers
`3`	16-bit signed integers
`4`	16-bit unsigned integers
`5`	8-bit unsigned integers

The precision used by the save command depends on the size and type of each matrix. Matrices with any non-integer entries and matrices with 10,000 or fewer elements are saved in floating-point formats requiring 8 bytes per real element. Matrices with all integer entries and more than 10,000 elements are saved in the following formats (see table below), requiring fewer bytes per element.

[0:255]	1
[0:65535]	2
[-32767:32767]	2
[-2^31+1:2^32-1]	4
other	8

- `T` indicates the matrix type according to the table below. Note that the elements of a text matrix are stored as floating-point numbers between 0 and 255 representing

ASCII-encoded characters.

`0`	Numeric (Full) matrix
`1`	Text matrix
`2`	Sparse matrix

mrows

An integer specifying the number of rows in the matrix.

ncols

An integer specifying the number of columns in the matrix.

imagf

An integer specifying whether the matrix has an imaginary part. If set to 0, there is only real data; if set to 1, the matrix has an imaginary part.

namlen

An integer specifying the length in bytes plus one (null terminated) of the matrix name.

name

A null-terminated string of `namlen` ASCII bytes.

C.2.2 Informative Matrices

The matrices `Aclass`, `name`, and `description` are of type `char`, while the `dataInfo` matrix is of type `int32_t`. Each of the matrices contains information about the simulation and the variables.

Aclass

`Aclass` is a matrix with a (4,*) size; the number of columns depends on the lengths of the strings they contain. The information we get from this matrix is the format of the file (file format version and storage format) as well as the model name.

Below is a simplified and schematic representation of the matrix `Aclass`.

```
-------------------------------------
Atrajectory
1.1
GettingStarted.MultiDomain.DCMotor
binTrans
-------------------------------------
```

The first line is always the mandatory string `Atrajectory`. The second line is the version of the file format. The third line is the full name of the simulated model. The fourth and last line determines the format of the matrices that follow.

name

The name matrix contains the names of the variables in the file, including states, variables, parameters, constants, and so on. The n^{th} row contains the name of the n^{th} variable. The global length of a row depends on the maximal length of all the strings.

Below is a simplified and schematic representation of the name matrix.

```
-------------------------------------
Time
inductor1.i
inertial.phi
...
...
-------------------------------------
```

description

This third matrix contains description texts for all variables. This corresponds to the strings you may add when defining a variable in Modelica. If `nVars` is the total number of variables, the matrix description is a `(nVars,*)` -sized matrix.

Below is a simplified and schematic representation of the description matrix.

```
-------------------------------------
Time
Current flowing from pin p to n [A]
Absolute rotation angle of component
...
...
-------------------------------------
```

If a variable does not have an associated description, the line is filled with blanks.

dataInfo

This fourth matrix `dataInfo` is a `(nVars,4)`-sized matrix of type `int32_t`, where `nVars` is the total number of variables. The matrix contains information on how and where each variable is stored in the data matrices. The *n*th row describes the *n*th variable.

Here is the correspondence between column values and their meaning:

- `dataInfo(n,1) = j` means that the *n*th variable data is stored in matrix `data_j`.
- `dataInfo(n,2) = k` means that the *n*th variable data is stored in column `abs(k)` of matrix `data_j`, with `sign(k)` used as the sign.
- `dataInfo(n,3) = 0` is reserved for future use; always `0`.
- `dataInfo(n,4) = -1` is reserved for future use; always `-1`.

Note that `dataInfo(1,1) = 0` means that column number `dataInfo(1,2)` is the abscissa data names `name(1,*)` of all data matrices. This is usually `Time`. This special rule is necessary, since otherwise the abscissa name has to be repeated in the matrix `name` and the names in it would no longer be unique.

With one-based indexing, the structure of the matrix can be schematically given by the following example:

```
-------------------------------------
0   1   0   -1
1   2   0   -1
1   3   0   -1
2   2   0   -1
...
...
-------------------------------------
```

where

- `name(1,*)` is abscissa data stored in `data_*(*, 1)`
- `name(2,*)` is stored in `data_1(*, 2)`
- `name(3,*)` is stored in `data_1(*, 3)`
- `name(4,*)` is stored in `data_2(*, 2)`

and so forth until the `nVars` row of data. Immediately following is the data matrix, containing the value of each variable at each time step.

C.2.3 Data Matrices

There can be several matrices for a simulation, and each of them has a specific name `data_<index>`, where `<index>` is the index of the matrix. Data matrices are always of type `double`. The information concerning these matrices is extracted from the previous matrix, `dataInfo`, as is the location of the values of interest.

C.3 Input Variable Data Files

An input variable data file contains data for one or more input variables in a simulation. It can either be a simulation result file (.mat) as described in Section C.2 on page 155, or an ASCII text file (.txt) as described in this section.

An ASCII text file consists of a file header and one or more matrices, specifying input variables and their values at different points in time.

C.3.1 File Header

The first line of the file must always consist of exactly two characters, namely `#1`.

C.3.2 Matrices

Each matrix in the file represents an input variable and its values at different points in time. A matrix is always preceded with a header specifying the type, name, and size of the matrix. This header must be exactly one line, and may include an optional comment. The format of the header is:

```
double name(rows, columns)   #comment
```

The type of the matrix must always be `double`. The number of rows in the matrix corresponds to the number of time points specified for the variable, and the number of columns is the size of the variable plus one, where the first column is used to specify time and the other columns contain the data to be interpolated.

For a scalar variable, the number of columns in the matrix would be 2; for an array of length 3, the number of columns in the matrix would be 4. For a matrix with dimensions [3,2], the number of columns would be 7 (3*2 + 1), and so on.

For each matrix in the file, the first column must either be monotonically increasing or monotonically decreasing. An event is specified using two consecutive rows with the same time value.

C.3.3 Example

For a simulation with three input variables `s`, `t`, and `u`, where `s` is a scalar, `t` is an array of size `2`, and `u` is a matrix of size `[2,2]`, we could use an input variable data file with the following contents:

```
#1
double s(5,2) # scalar
0.2 55.4
0.4 41.7
0.8 21.2
1.6 11.3
3.2 10.7

double t(4, 3) # array of size 2
1.0  0.1 3.2
2.0 22.3 4.4
3.0 31.7 5.0
5.0 34.0 5.2

double u(4, 5) # matrix of size [2,2], with an event at time 0.3.
0.8 4.0 4.1 4.2 4.3
0.3 3.0 3.1 3.2 3.3
0.3 2.0 2.1 2.2 2.3
0.1 1.0 1.1 1.2 1.3
```

Worth noting is that for the matrix `u`, the data should be specified in row major order, i.e. at time `0.8`, `u[1,1] = 4.0`, `u[1.1] = 4.1`, `u[2,1] = 4.2`, and `u[2,2] = 4.3`. You may also notice that we have specified an event at time `0.3` for the input variable `u`.

Appendix D: **Communication with Simulation via TCP**

This appendix describes the protocol used for the interface to a *SystemModeler* simulation that can be used for reading and writing data to a running simulation. A network protocol with two sessions is used: Wolfram *SystemModeler*-Simulation Control Session (WSM-SCS) and Wolfram *SystemModeler*-Simulation Data Session (WSM-SDS). The figure below illustrates the setup.

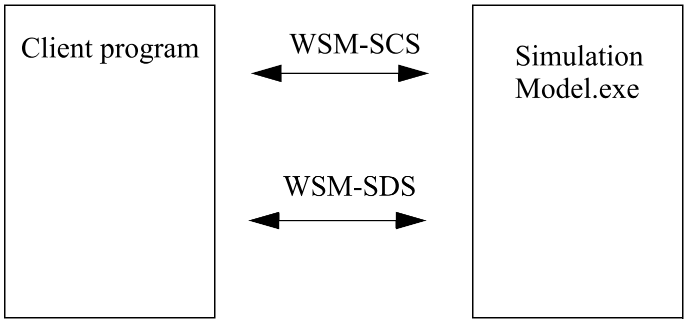


Figure D-1: *SystemModeler* simulation sessions.

The protocol allows for setting both parameter and input variables and retrieving (output) variables from each integration step. If a real-time simulation should be achieved, the XML input file to the simulator is set up to use a single step solver with a fixed step size, which determines the sample rate of the simulator. An option can be given to state that the simulator should synchronize with the computer's real-time clock, if desired.

D.1 WSM-SCS/SDS

The *SystemModeler*-Simulation Control Session is used to control a running simulation server by issuing start/stop commands and setting input variable values. The protocol runs over TCP and a message consists of a header and a payload.

The *SystemModeler*-Simulation Data Session is used to receive simulation data from a running simulation server. An SDS session must be paired with an SCS session that is used to set up the variable subscriptions.

D.1.1 WSMCOM Packets

Each packet consists of a header describing the type of package as well as the length of the payload.

WSMCOM Header

The header consists of information about which protocol to use, which type of package is sent, packet specific data, and the length of the payload.

Bits		
0-7	Protocol version	
8-15	Packet type	1 - `HELLO_SCS` 2 - `HELLO_SDS` 3 - `CMD` 4 - `CMD_REPLY` 5 - `CMD_ERROR` 6 - `SDS_OUTPUT_DATA` 8 - `SDS_INPUT_DATA`
16-31	Packet specific	
32-63	Payload length	

HELLO_SCS Packet

The client initiates an SCS session by sending a `HELLO_SCS` packet; the payload should be empty. Upon successful initiation, the server will reply with a `CMD_REPLY` packet where the payload will be `{SESSION_ID_SCS}`. If it fails, the server will reply with a `CMD_ERROR` packet where the payload will be a description of the problem.

HELLO_SDS Packet

The client initiates an SDS session by sending a `HELLO_SDS` packet; the payload should be the `SESSION_ID_SCS` given to the corresponding SCS session. If successful, the server will reply with a `CMD_REPLY` packet where the payload will be `{SESSION_ID_SDS}`. If it fails, the server will reply with a `CMD_ERROR` packet where the payload will be a description of the problem.

After the simulation has been started, the server will start to stream `SDS_OUTPUT` packets to the client after every time step according the subscription setup with the corresponding SCS session. The SDS session cannot be used to send commands to the server.

CMD Packet

The client sends an interactive command to the server by sending a CMD packet; the payload should contain the command. On success the server will reply with a `CMD_REPLY` packet and the payload will contain the result of the command. On failure the server will reply with a `CMD_ERROR` packet where the payload will be a description of the problem. For a list of commands, see Section D.1.2 on page 166.

CMD_REPLY Packet

The server sends a `CMD_REPLY` packet as a reply to a `HELLO_SCS`, `HELLO_SDS`, or `CMD` packet.

CMD_ERROR Packet

The server sends a `CMD_ERROR` packet as a reply to a `HELLO_SCS`, `HELLO_SDS`, or `CMD` packet that results in an error on the server. The payload will contain a description of the problem.

CMD_REPLY and CMD_ERROR Payloads

All payloads in `CMD_REPLY` and `CMD_ERROR` packets are formatted as arrays enclosed in `{}`. The elements in the array are separated with a comma. Valid array elements are:

- Strings enclosed in double quotes
- Integer values

- Real values
- Other arrays enclosed in `{}`

Example: array with string values: `{"var1", "var2", "var3"}`

Example: array with real values: `{1.0, 2.0, 3.0}`

Example: array with nested array, strings, and numbers: `{{"var1", 1.0}, {"var2", 2.0}}`

SDS_OUTPUT_DATA Packet

When an SDS session has been initiated by sending a `HELLO_SDS` packet, the server will start to send `SDS_OUTPUT_DATA` packets with the values of the subscribed variables after each solver step. Bit 16-23 in the packet header is the subscription ID.

The payload consists of the values for all subscribed variables in binary format (`double`).

SDS_INPUT_DATA Packet

The client can send an `SDS_INPUT_DATA` packet on an SDS session to set all input variables; the payload should consist of the values in binary format (`double`) for all input variables in the order indicated by `getInputVariableNames()`.

D.1.2 SCS Interactive Commands

All interactive commands are sent as a CMD package on a SCS session.

getInputVariableNames()

Returns a list with the names of all input variables in the model.

```
cmd: getInputVariableNames()
reply: {"var1", "var2", "var3", …}
```

getModelName()

Returns the name of the model.

```
cmd: getModelName()
reply: {"IntroductoryExamples.HelloWorld"}
```

getOutputVariableNames()

Returns a list with the names of all output variables in the model.
```
cmd: getOutputVariableNames()
reply: {"var1", "var2", "var3", …}
```

getParameterNames()

Returns a list with the names of all parameters in the model.
```
cmd: getParameterNames()
reply: {"var1", "var2", "var3", …}
```

getStateDerivativeVariableNames()

Returns a list with the names of all state derivative variables in the model.
```
cmd: getStateDerivativeVariableNames()
reply: {"var1", "var2", "var3", …}
```

getStateVariableNames()

Returns a list with the names of all state variables in the model.
```
cmd: getStateVariableNames()
reply: {"var1", "var2", "var3", …}
```

getVariableNames()

Returns a list with the names of all variables in the model.
```
cmd: getVariableNames()
reply: {"var1", "var2", "var3", …}
```

getAlgebraicVariableNames()

Returns a list with the names of all algebraic variables in the model.
```
cmd: getAlgebraicVariableNames()
reply: {"var1", "var2", "var3", …}
```

setSubscription({"var1", "var2", "var3", ...})

Sets the subscription for the corresponding SDS session; returns a `SUBSCRIPTION_ID` used to identify the packets received in the SDS session.
```
cmd: setSubscription({"var1", "var2", "var3", …})
```

```
reply: {SUBSCRIPTION_ID}
```

setInputValues({"var1", value1, "var2", value2, ...})

Sets input variable values; the new values will be used in the next solver step.

```
cmd: setInputVariables({"var1", value1, "var2", value2, ...})
reply: {true}
```

getTime()

Returns the current simulation time.

```
cmd: getTime()
reply: {2.54}
```

suspendSimulation()

Suspends the simulation until a `continueSimulation()` or `terminateSimulation()` command is issued.

```
cmd: suspendSimulation()
reply: {true}
```

continueSimulation()

Continues the simulation after a `suspendSimulation()` command.

```
cmd: continueSimulation()
reply: {true}
```

startSimulation()

Starts the simulation if the simulation is configured to wait for a `startSimulation()` command before starting.

```
cmd: startSimulation()
reply: {true}
```

stopSimulation()

Stops the simulation when the current solver step has finished.

```
cmd: stopSimulation()
reply: {true}
```

setParameterValues({"var1", value1, "var2", value2, ...})

Sets parameter values; the new values will be used in the next solver step.

```
cmd: setParameterValues({"var1", value1, "var2", value2, ...})
reply: {true}
```

getVariableValues({"var1", "var2", ...})

Returns a list of variable values.

```
cmd: getVariableValues({"var1", "var2", ...})
reply: {value1, value2, ...}
```

Appendix E: **Third-Party Licenses**

The usage rights for a simulation executable produced by *SystemModeler* are defined in the *SystemModeler* license agreement. In addition, the third-party licenses listed in this appendix apply.

E.1 Expat

E.2 SuperLU

E.3 SUNDIALS

Index

For further details, visit:

- www.wolfram.com/system-modeler
- reference.wolfram.com/system-modeler