

# COMSOL Multiphysics

Application Builder Reference Manual

# Application Builder Reference Manual

© 1998–2018 COMSOL

Protected by patents listed on [www.comsol.com/patents](http://www.comsol.com/patents), and U.S. Patents 7,519,518; 7,596,474; 7,623,991; 8,457,932; 8,954,302; 9,098,106; 9,146,652; 9,323,503; 9,372,673; and 9,454,625. Patents pending.

This Documentation and the Programs described herein are furnished under the COMSOL Software License Agreement ([www.comsol.com/comsol-license-agreement](http://www.comsol.com/comsol-license-agreement)) and may be used or copied only under the terms of the license agreement.

COMSOL, the COMSOL logo, COMSOL Multiphysics, COMSOL Desktop, COMSOL Server, and LiveLink are either registered trademarks or trademarks of COMSOL AB. All other trademarks are the property of their respective owners, and COMSOL AB and its subsidiaries and products are not affiliated with, endorsed by, sponsored by, or supported by those trademark owners. For a list of such trademark owners, see [www.comsol.com/trademarks](http://www.comsol.com/trademarks).

Version: COMSOL 5.4

## Contact Information

Visit the Contact COMSOL page at [www.comsol.com/contact](http://www.comsol.com/contact) to submit general inquiries, contact Technical Support, or search for an address and phone number. You can also visit the Worldwide Sales Offices page at [www.comsol.com/contact/offices](http://www.comsol.com/contact/offices) for address and contact information.

If you need to contact Support, an online request form is located at the COMSOL Access page at [www.comsol.com/support/case](http://www.comsol.com/support/case). Other useful links include:

- Support Center: [www.comsol.com/support](http://www.comsol.com/support)
- Product Download: [www.comsol.com/product-download](http://www.comsol.com/product-download)
- Product Updates: [www.comsol.com/support/updates](http://www.comsol.com/support/updates)
- COMSOL Blog: [www.comsol.com/blogs](http://www.comsol.com/blogs)
- Discussion Forum: [www.comsol.com/community](http://www.comsol.com/community)
- Events: [www.comsol.com/events](http://www.comsol.com/events)
- COMSOL Video Gallery: [www.comsol.com/video](http://www.comsol.com/video)
- Support Knowledge Base: [www.comsol.com/support/knowledgebase](http://www.comsol.com/support/knowledgebase)

Part number: CM020010

# Contents

## Chapter 1: Introduction

<b>About the Application Builder</b>	<b>8</b>
What Can You Do with the Application Builder? . . . . .	8
Accessing the Documentation . . . . .	8
<b>Overview of the Manual</b>	<b>10</b>

## Chapter 2: Application Builder Tools

<b>Introduction</b>	<b>12</b>
<b>Starting the Application Builder</b>	<b>13</b>
Launching the Application Builder . . . . .	13
Opening the Application Builder from the COMSOL Desktop. . . . .	14
Application Examples in the Application Libraries . . . . .	14
The Application Builder and the Application Tree . . . . .	14
The Home Toolbar . . . . .	14
Cutting, Copying, Duplicating, Deleting, and Pasting Components . . . . .	17
Copying Application Components Between Applications. . . . .	17
Compiling Apps . . . . .	17
<b>The Inputs Branch</b>	<b>19</b>
The Application Argument Node. . . . .	19
<b>The Main Window Branch</b>	<b>21</b>
The Main Window Node. . . . .	21
The Menu Bar Node . . . . .	22
The Toolbar Node . . . . .	22
The Menu Node . . . . .	22
The File Menu Node . . . . .	23
The Ribbon Node . . . . .	23
The Ribbon Tab Node. . . . .	23
The Ribbon Section Node . . . . .	23
The Item Node . . . . .	24
The Toggle Item Node. . . . .	26
The Separator Node . . . . .	28
<b>The Forms Branch</b>	<b>29</b>
The Forms Node. . . . .	29
The Form Node . . . . .	29
<b>The Events Branch</b>	<b>33</b>
The Events Node. . . . .	33
The Event Node . . . . .	33
<b>The Declarations Branch</b>	<b>36</b>
Array Syntax . . . . .	37

The String Node . . . . .	37
The Boolean Node . . . . .	38
The Integer Node . . . . .	38
The Double Node . . . . .	39
The Array 1D String Node . . . . .	39
The Array 1D Boolean Node . . . . .	40
The Array 1D Integer Node . . . . .	40
The Array 1D Double Node . . . . .	41
The Array 2D String Node . . . . .	41
The Array 2D Boolean Node . . . . .	42
The Array 2D Integer Node . . . . .	42
The Array 2D Double Node . . . . .	43
The Choice List Node . . . . .	43
Activation Condition . . . . .	44
The File Node . . . . .	45
The Unit Set Node . . . . .	45
The Graphics Data Node. . . . .	46
Adding Shortcuts . . . . .	46
Editing Initial Values and Arguments in Declarations and Command Sequences	47
<b>The Methods Branch</b>	<b>50</b>
The Methods Branch . . . . .	50
The Method Node . . . . .	50
<b>The Libraries Branch</b>	<b>51</b>
Images . . . . .	51
Sounds. . . . .	51
Files . . . . .	52
<b>Planning and Preparing an Application</b>	<b>53</b>
Preparing an Application . . . . .	53
<b>Creating Applications from Models</b>	<b>54</b>
Copy as Code to Clipboard. . . . .	54
Testing the Application . . . . .	55
<b>Keyboard Shortcuts</b>	<b>56</b>

## Chapter 3: Working with Forms

<b>Introduction</b>	<b>60</b>
Overview of the Forms and Tools for Creating Forms. . . . .	60
Working with a Form and Using the New Form Wizard . . . . .	60
Data Access. . . . .	63
The Form Toolbar . . . . .	63
The Form Window Layout Modes . . . . .	65
The Sketch Mode. . . . .	65
The Grid Mode . . . . .	66
Previewing and Testing the Form. . . . .	70
Running Local Methods in Form Objects . . . . .	70

<b>The Form Objects</b>	<b>71</b>
Overview of the Form Objects . . . . .	71
Input Field . . . . .	72
Button . . . . .	76
Toggle Button . . . . .	79
Check Box . . . . .	83
Combo Box. . . . .	85
Text Label . . . . .	88
Unit. . . . .	89
Equation . . . . .	90
Line . . . . .	92
Data Display . . . . .	93
Graphics . . . . .	95
Web Page . . . . .	98
Image . . . . .	100
Video . . . . .	101
Progress Bar . . . . .	103
Log . . . . .	104
Message Log . . . . .	106
Results Table . . . . .	107
Form . . . . .	109
Form Collection . . . . .	110
Card Stack . . . . .	112
Card . . . . .	114
File Import . . . . .	116
Information Card Stack . . . . .	118
Array Input . . . . .	120
Radio Button . . . . .	123
Selection Input. . . . .	126
Text. . . . .	128
List Box . . . . .	130
Table . . . . .	133
The Edit Custom Toolbar Item Dialog Box . . . . .	137
Slider . . . . .	139
Hyperlink. . . . .	142
Toolbar . . . . .	143
Spacer . . . . .	145

## Chapter 4: Working with Methods

<b>Overview</b>	<b>148</b>
Opening a Method Editor Window . . . . .	148
Coding and Methods Overview . . . . .	148
The Application Builder Window. . . . .	149
The Method Windows. . . . .	149
The Method Toolbar . . . . .	149
The Method Nodes and Method Editor Windows . . . . .	151
The Utility Class Node . . . . .	152
The External Java Library Node . . . . .	153
The External C Library Node . . . . .	153
Using External C Libraries . . . . .	154
File Schemes and File Handling. . . . .	157

Getting Files to and from the Client File System . . . . .	158
<b>Creating Methods</b>	<b>160</b>
Syntax Highlighting and Comments . . . . .	160
Code Completion and Tooltip Help. . . . .	161
Code Folding . . . . .	166
Adding Language Elements . . . . .	167
Adding Model Expressions . . . . .	167
Adding Model Code and Form Objects . . . . .	168
Going to Node the Source Code is Mapped To . . . . .	170
Recording Code . . . . .	170
Using Shortcuts . . . . .	171
Creating Local Variables and Their Type Declarations . . . . .	172
Calling Other Methods Directly . . . . .	173
Using Properties Defined in Declarations as Variables . . . . .	173
Searching and Finding Text . . . . .	173
Indentation and Whitespace Formatting . . . . .	174
Brace Matching . . . . .	174
<b>Debugging Methods for Applications</b>	<b>175</b>
Indication of Compilation Errors . . . . .	175
Debugging Tools . . . . .	175
The Errors and Warnings Window . . . . .	176
Handling Runtime Errors in Methods . . . . .	177
Stopping a Running Method . . . . .	177

# Introduction

Read this guide to learn how to use the Application Builder, a set of tools for creating custom applications based on multiphysics models. The Application Builder is available directly in the COMSOL Desktop® and includes a comprehensive set of user interface and programming tools. See the *Introduction to the Application Builder* for an overview of the Application Builder and the Application Libraries, where you will find many example applications.

In this chapter:

- [About the Application Builder](#)
- [Overview of the Manual](#)

# About the Application Builder

In this section:

- [What Can You Do with the Application Builder?](#)
- [Accessing the Documentation](#)

---

## *What Can You Do with the Application Builder?*

The Application Builder provides tools for creating custom applications from COMSOL Multiphysics® models. You can use graphical tools and editors as well as built-in language elements and Java® code to tailor an application with the user inputs, design, and results that you want to include. Simulation applications have a variety of uses, such as:

- Efficiently testing various design parameters
- Streamlining the product development workflow

You, and other COMSOL users, can run applications in COMSOL Multiphysics. You can also make COMSOL applications available for colleagues and customers who do not use COMSOL Multiphysics by letting them connect to a COMSOL Server™ installed in a central location on your network or in the cloud. Your colleagues and customers can then run apps on a COMSOL Server through a web browser or through a COMSOL Client.

---

## *Accessing the Documentation*

A number of Internet resources provide more information about COMSOL, including licensing and technical information. The electronic documentation, topic-based (or context-based) help, and the Application Libraries are all accessed through the COMSOL Desktop®.



If you are reading the documentation as a PDF file on your computer, the [blue links](#) do not work to open content referenced in a different guide. However, if you are using the Help system in COMSOL Multiphysics, these links connect to other modules (as long as you have a license), application examples, and documentation sets.

---

### **CONTACTING COMSOL BY EMAIL**

For general product information, contact COMSOL at [info@comsol.com](mailto:info@comsol.com).

To receive technical support from COMSOL for the COMSOL products, please contact your local COMSOL representative or send your questions to [support@comsol.com](mailto:support@comsol.com). An automatic notification and case number is sent to you by email.



## COMSOL WEBSITES

COMSOL website	<a href="http://www.comsol.com">www.comsol.com</a>
Contact COMSOL	<a href="http://www.comsol.com/contact">www.comsol.com/contact</a>
Support Center	<a href="http://www.comsol.com/support">www.comsol.com/support</a>
Product Download	<a href="http://www.comsol.com/product-download">www.comsol.com/product-download</a>
Product Updates	<a href="http://www.comsol.com/support/updates">www.comsol.com/support/updates</a>
Discussion Forum	<a href="http://www.comsol.com/community">www.comsol.com/community</a>
Events	<a href="http://www.comsol.com/events">www.comsol.com/events</a>
COMSOL Blog	<a href="http://www.comsol.com/blogs">www.comsol.com/blogs</a>
COMSOL Video Gallery	<a href="http://www.comsol.com/video">www.comsol.com/video</a>
Support Knowledge Base	<a href="http://www.comsol.com/support/knowledgebase">www.comsol.com/support/knowledgebase</a>

# Overview of the Manual

The *Application Builder Reference Manual* contains comprehensive information about the tools and functionality in the Application Builder. In addition to this introduction, you will find the following chapters:

- [Application Builder Tools](#): An overview of the tools in the Application Builder as well as details about the nodes in the Application Builder tree and the tools on the **Home** ribbon toolbar.
- [Working with Forms](#): Information about the tools in the Application Builder for creating and designing forms for an application.
- [Working with Methods](#): Information about the tools in the Application Builder for creating and writing code for methods and classes to extend the functionality of an application.

See also the following documentation:

- *Introduction to the Application Builder*, for an overview and introduction to the Application Builder.
- *Application Programming Guide*, for a guide to writing code for applications using the method editor.
- *COMSOL Multiphysics Programming Reference Manual*, for information about commands for the model object, which are available for programming in methods that you can add to your applications.

# Application Builder Tools

This chapter provides an overview of the tools available in the Application Builder to help you to create custom applications. The tools include editors and user interface components for designing forms and windows, and method editors and coding and debugging tools for adding application-specific actions and methods.

In this chapter:

- [Introduction](#)
- [Starting the Application Builder](#)
- [The Main Window Branch](#)
- [The Forms Branch](#)
- [The Events Branch](#)
- [The Declarations Branch](#)
- [The Methods Branch](#)
- [The Libraries Branch](#)
- [Planning and Preparing an Application](#)
- [Creating Applications from Models](#)
- [Keyboard Shortcuts](#)

# Introduction



The Application Builder includes a comprehensive set of tools for creating and deploying applications based on COMSOL models. The main parts of the Application Builder that you use to create applications are the following:

- The *COMSOL Desktop* with the Application Builder window, which contains a tree with the nodes that define the application, and ribbon toolbars with tools for creating applications. See [The Application Builder and the Application Tree](#) and further sections in this chapter about the branches and nodes in the Application Builder tree.
- The tools for creating and designing forms (user interfaces) with various form components (user interface controls) that are adapted for the application. See the [Working with Forms](#) chapter for more information.
- The tools for creating and editing methods and classes for including custom code that can be connected to user interface events, for example. See the [Working with Methods](#) chapter for more information.

To get started, you can explore and run applications that are included in COMSOL Multiphysics and in many of the add-on products. In the **Application Libraries** window, the applications that appear under **Applications** are runnable application examples that you can open and inspect to learn how to create forms (user interfaces) and methods (code for added functionality in the application). See the *COMSOL Multiphysics Reference Manual* for details about the **Application Libraries** window.

# Starting the Application Builder

## *Launching the Application Builder*

To start the Application Builder, click the **Application Builder** button (  ) on the **Home** toolbar. The COMSOL Desktop then switches to display the toolbar for the Application Builder (or opens in a separate desktop window if you select the **Use separate desktop window for Application Builder** check box on the **Application Builder** page in the **Preferences** dialog box). When you have finished developing your application, you can run it by choosing **Run Application** from the **File** menu and pointing to the MPH-file for the application. You can also browse and run applications in [The Main Window Branch](#). The figure below also shows how the applications can be accessed and run in the COMSOL Desktop. To return from the Application Builder to the Model Builder, click Model Builder on the Application Builder's **Home** toolbar (  ).

The Application Builder window works analogously to the Model Builder with an *application tree*, context menus, toolbars, and *Settings* windows for every application tree node. The nodes in the application tree represent forms, events, methods, and other parts of the runnable application. You can add the graphical user interface (GUI) components (form objects) from the **Form** contextual ribbon toolbar and then position them interactively using the graphics tools in the Application Builder (see [Working with Forms](#)). You can also see a preview of what the form or entire application will look like when the application is run. The Application Builder is available for Microsoft Windows<sup>®</sup> operating system installations, but applications created using the Application Builder can be deployed and used on all operating systems, including running in a web browser together with COMSOL Server.

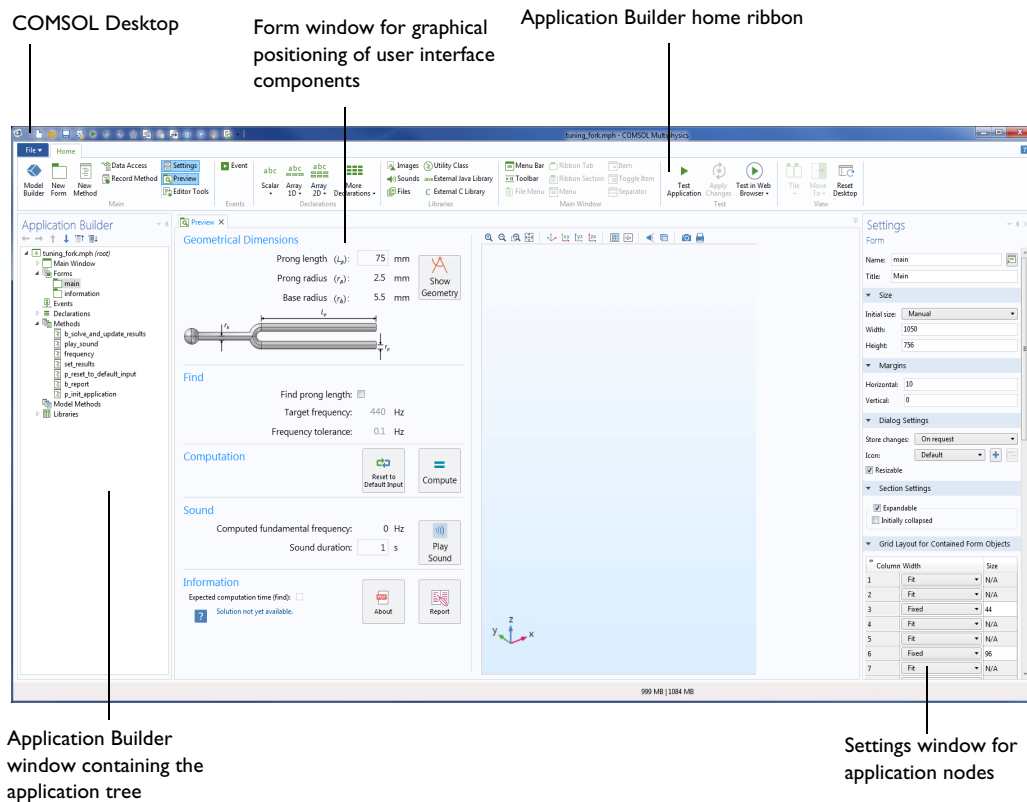




Figure 2-1: The Application Builder adapts the COMSOL Desktop for the design of applications.

## Opening the Application Builder from the COMSOL Desktop

---

When you are in the COMSOL Desktop, you can toggle between the Application Builder and COMSOL Multiphysics. On the **Home** toolbar, click **Application Builder**  to open the Application Editor, where you can modify the user interface of the application as well as create and edit code for the application. You can also press Ctrl+Alt+A.

Conversely, when you are in the Application Builder, you can click **Model Builder**  on the **Home** toolbar to return to COMSOL Multiphysics. You can also press Ctrl+Alt+M.

## Application Examples in the Application Libraries


---


In the **Application Libraries** window, you can browse the application libraries for each COMSOL product. Most of them include an **Applications** folder, which contains runnable application examples that you can open in the Application Builder and run. You can use these applications as inspiration, for example, to see how to use form objects in a user interface design or create methods for extending the functionality of your app.

## The Application Builder and the Application Tree

---

The **Application Builder** window on the COMSOL Desktop contains the *application tree*. This tree displays nodes from the Application Builder's data structure and nodes from the embedded model. All nodes from the embedded model behave exactly like the nodes in the model tree when working with the Model Builder. The application tree also includes a root node of the application and the following branches under the root node:

- A **Compiler** node () for creating standalone executable simulation apps if added from the **Home** ribbon (requires the COMSOL Compiler). See [Compiling Apps](#).
- [The Inputs Branch](#)
- [The Main Window Branch](#)
- [The Forms Branch](#)
- [The Events Branch](#)
- [The Declarations Branch](#)
- [The Methods Branch](#)
- [The Libraries Branch](#)

The root node () in the Application Builder is similar to the root node in the Model Builder, but it also contains an **Application** section with settings specific to applications (see [The Root Settings and Properties Windows](#) in the *COMSOL Multiphysics Reference Manual* for more information).



## The Home Toolbar









---

The Home toolbar is always available and contains buttons for accessing the most common functionality in the Application Builder.

### THE MAIN SECTION

This section contains the following buttons for moving to various windows and creating new forms and methods:

- The **Model Builder** button () , to switch from the Application Builder to the Model Builder windows and the standard COMSOL Desktop.
- The **New Form** button () , to create a new form using the New Form wizard. See [Working with a Form and Using the New Form Wizard](#).

- The **New Method** button (  ), to create a new **Global Method** or **Form Method** node open its code in a new editor tab. See [The Method Nodes and Method Editor Windows](#).
- The **Data Access** button (  ), to add model-dependent data and properties, as well as application-specific properties that can be modified from a running application through the data access functionality. See [Data Access](#).
- The **Record Method** button (  ), to create a new method by starting a recording session of operations on the embedded model that you can later use as code in that method. When the recording starts, the button changes to the **Stop Recording** button (  ), which you click to end the recording. See [Recording Code](#).
- The **Compiler** button (  ), to compile an app created using the Application Builder into a standalone, runnable app (with a license for the COMSOL Compiler). See [Compiling Apps](#).
- The **Settings** button (  ), to move, open, or close the **Settings** window.
- The **Preview** button (  ), to show or hide the **Preview** window for a live preview of the forms and methods in the application. In the **Preview** window, you can scroll to get a preview of the forms and methods in the application, which can be useful, for example, if you are working on a method that interacts with a form. To show a preview of a form in the **Preview** window, select a form or method node in the **Application Builder** window.
- The **Editor Tools** button (  ), to show or hide the **Editor Tools** window, where you can choose common COMSOL Multiphysics model operations and insert them into a method or generate form objects based on them. See [Adding Model Code and Form Objects](#).


#### THE INPUTS SECTION

This section contains a button for adding an application argument. See the [The Inputs Branch](#).

- Click the **Application Argument** button (  ), to add an **Application Argument** node for defining an input argument for an application. See [The Application Argument Node](#).









#### THE EVENTS SECTION

This section contains a button for creating events. See [The Events Branch](#).

- Click the **Events** button (  ), to add an **Event** node for defining an event. See [The Event Node](#).







#### THE DECLARATIONS SECTION

This section contains buttons for creating variable, file, and choice list declarations. See [The Declarations Branch](#). The buttons include:

- The **Scalar** button (  ), to open a menu where you can choose to add a string, Boolean, integer, or double scalar.
- The **Array 1D** button (  ), to open a menu where you can choose to add a string, Boolean, integer, or double 1D array.
- The **Array 2D** button (  ), to open a menu where you can choose to add a string, Boolean, integer, or double 2D array (matrix).
- Under **More Declarations** (  ), choose one of the following items:
  - **File** (  ), to add a file declaration for accessing files.
  - **Choice List** (  ), to add a choice list for defining a list of allowed values.
  - **Unit Set** (  ), to add a unit set.
  - **Graphics Data** (  ), to add a graphics data for accessing picked data from a graphics window.










## THE LIBRARIES SECTION

This section contains the following buttons for opening the images, sounds, and file libraries and for adding external code and utility classes:

- The **Images** button () , to open the Images library. See [Images](#).
- The **Sounds** button () , to open the Sounds library. See [Sounds](#).
- The **Files** button () , to open the Files library. See [Files](#).
- The **Utility Class** button () , to add a Java utility class. See [The Utility Class Node](#).
- The **External Java Library** button () , to add an external Java library. See [The External Java Library Node](#).
- The **External C Library** button () , to add a dynamically linked native code library. See [The External C Library Node](#).




## THE MAIN WINDOW SECTION

This section contains buttons for adding menus and toolbars to the application's main window. The buttons are only available when it is possible to add the respective menus or toolbars (depending on which menu type you use and the current node under **Main Window**) and include:


- The **Menu Bar** button () , to add a menu bar. See [The Menu Bar Node](#).
- The **Toolbar** button () , to add a toolbar. See [The Toolbar Node](#).
- The **File Menu** button () , to add a file menu. See [The File Menu Node](#).
- The **Ribbon Tab** button () , to add a ribbon tab to a ribbon. See [The Ribbon Tab Node](#).
- The **Ribbon Section** button () , to add a ribbon section to a ribbon. See [The Ribbon Section Node](#).
- The **Menu** button () , to add a menu. See [The Menu Node](#).
- The **Item** button () , to add a menu item. See [The Item Node](#).
- The **Toggle Item** button () , to add a toggle item. See [The Toggle Item Node](#).
- The **Separator** button () , to add a separator to separate groups of related menu items in a menu. See [The Separator Node](#).

## THE TEST SECTION

This section contains the following tools for testing the application:

- The **Test Application** button () , to launch the application in a separate window so that you can test it. See [Testing the Application](#).
- The **Apply Changes** button () , to compile and apply code changes to the running application (a so-called hot code swap). See [Applying Changes to a Running Application](#).
- The **Test in Web Browser** button () , to test run the application in a web browser. See [Testing the Application](#).




## THE COMPARE SECTION

Click the **Compare** button () to open a **Select Application** window where you can select another application (as an MPH-file) and compare it to the current application. The results of the comparison appear in a **Comparison Result** window. See [Comparing Models and Applications](#) in the *COMSOL Multiphysics Reference Manual* for more information.



## THE VIEW SECTION

The **View** section contains the following buttons for rearranging the views in the Application Builder desktop window:

- The **Tile** () and **Move To** () buttons, to rearrange the windows in the Application Builder.
- The **Reset Desktop** button (), to reset the desktop layout to the default state.

## *Cutting, Copying, Duplicating, Deleting, and Pasting Components*

---

Nodes that you add to the Application Builder tree, such as forms, methods, and declarations, can be cut, copied, duplicated, deleted, and pasted (see the section below for information about copying between applications). You can use buttons on the Quick Access Toolbar, or right-click a node and choose one of the following options:

- **Cut** (**Ctrl+X** in cases where that shortcut is supported)
- **Copy**
- **Duplicate** **Ctrl+Shift+D**
- **Delete** **Del**

If you cut or copy a node, you can paste it using the context menu on the parent node, such as **Paste Choice List** on the context menu for the **Declarations** node.

## *Copying Application Components Between Applications*

---

You can copy and paste many parts of an application to another app. For example, when you have copied a form or some form objects, it is possible to paste them between two running instances of COMSOL Multiphysics, or in one running instance after opening a new application. You can also copy methods, utility methods, external libraries, file declarations, choice list declarations, menu items, menu dividers, menus, ribbon sections, and ribbon tabs. The copy operation can handle references to various objects in the application. When you copy and paste between applications, the following information regarding references to methods and files can be useful:

- Local methods are included in the copy-paste operation. However, the code in the methods is unchanged.
- All image references are copied and added during paste if needed. If there is an existing image with the same name, it will be used instead of the copied version.
- No files or sounds are copied.

If there are potential issues with the paste operation, a message window appears. You can then choose to cancel the paste operation.



## *Compiling Apps*

---

You can compile apps developed using the Application Builder, so that they become standalone, runnable apps that can be deployed and run on any supported platform (Windows, Linux, and macOS) without a license file.



Compiling apps requires a license for the COMSOL Compiler.

To compile an app, click the **Compiler** button () in the **Home** toolbar's **Main** section and specify the output directory, the platforms to compile for, a splash screen, and possibly some additional settings (see below); then click the **Create Executable** button (). You can also use the `comsol compile` command to compile an app on any supported platform (see the [The COMSOL Commands](#) in the *COMSOL Multiphysics Reference Manual*).

When launching a compiled app for the first time, you have to accept a license agreement, and then some installation components are loaded before the splash screen appears and the app becomes active. For subsequent launches of apps, the splash screen appears directly.


The **Compiler** node's **Settings** window contains the following sections:

#### EXECUTABLE



In the **Output directory** field, specify the output directory where the compiled app will be stored, or click the **Browse** button to browse for a location.

From the **Set path to runtime** list, choose **Default**, **Ask**, or **Specify**. The runtime is the COMSOL Multiphysics computational engine that the executable app runs with. By default, it is unpacked in a COMSOL folder in the AppData folder, which typically is `C:\Users\<username>\AppData\Local`. That is the behavior when you choose **Default**. If you choose **Ask**, the app will ask for the appropriate runtime folder where the COMSOL Multiphysics runtime will be unpacked. If you choose **Specify**, then enter the paths to the respective runtime in the **Path to runtime Windows**, **Path to runtime Linux**, and **Path to runtime macOS** fields under **Specify folder**. You can use that option if you want to unpack the COMSOL Multiphysics runtime to another specified location on the file system.

Under **Platforms**, select the **Windows**, **Linux**, and **macOS** check boxes as desired (depending on the platforms where you want to users to be able to run the apps). By default, only the **Windows** check box is selected.

To compile the app into an executable app, click the **Create Executable** button () at the top of the **Settings** window. The size of a created app is about 300 MB.

#### SPLASH



From the **Splash** list, select an image to use as the splash-screen image when launching an app. The **Default** splash screen is a generic default splash screen for compiled apps. Click the **Add Image to Library and Use Here** button () to add any other BMP image to the list of splash-screen images and use it. Under **Preview**, you can see what the currently selected splash-screen image looks like. Click the **Export** button () to export the image used for the splash screen to a file.

#### OPTIONAL DATA

If your app is using the CAD kernel and functionality from the LiveLink CAD products, the CAD Import Module, or the Design Module, select the **CAD libraries** check box to include those libraries in the executable app.




If your license includes the Heat Transfer Module and your app makes use of weather data, select the **Weather data** check box to add weather data to the executable app.

# The Inputs Branch

Under the **Inputs** branch node () , you can add **Application Argument** nodes () for adding inputs to applications.

## *The Application Argument Node*

---

To add an **Application Argument** node () , right-click the **Inputs** node () and choose **Application Argument**, or click the **Application Argument** button () on the **Home** toolbar. Use an **Application Argument** node to define an input argument for an application. You can add several **Application Argument** nodes to support multiple input arguments to the application. The order is not important because it is the name that is used to map the command-line arguments

### *About Command-Line Arguments*

To specify command-line arguments when running an application, you specify the names using `appargnames` and the corresponding values using `appargvalues`. This syntax makes it possible to mix both general built-in arguments supported for all applications and custom arguments supported for a specific application. The following example shows a command line where the renderer is set to DirectX and the custom application arguments `width` and `height` are also specified:

```
comsol -run myapp.mph -3drend dx9 -appargnames width,height -appargvalues 12.3,7.4
```

For specifying values that are arrays, the same syntax as when specifying the initial values for the corresponding declarations is supported. That is, use `{` and `}` to specify the start and end of arrays or rows in 2D arrays and then commas to separate the individual values.

In addition to specifying arguments individually on the command line, you can also specify them in a file using the `-appargsfile` argument. The format of the file must use Java property file syntax as in the following example:

```
width = 1.7
height = 4.3
voltages = {4,6.7,11.2}
intmatrix = {{1,5,6},{7,9,1}}
```

Setting arguments from a file can also be combined with setting some arguments explicitly from the command line as shown in the following example:

```
comsol -run myapp.mph -appargsfile myfile.txt -appargnames width,height -appargvalues 12.3,7.4
```

If an argument appears both in the file and explicitly using `-appargvalues`, the value given using `-appargvalues` overrides the value given in the file.

To use several files with values, use the command-line parameters `-appargvarlist` and `-appargfilelist`. You can use them in a similar way as `-appargnames` and `-appargvalues`. The only difference is that `-appargfilelist` is a comma-separated list of files. In this case, the file only contains the values and does not need to contain the name of the input parameter to assign the values to. This approach has the advantage that files exported on the spreadsheet format from COMSOL Multiphysics can be directly be passed as input. The following command line shows an example of this:



```
comsol -run myapp.mph -appargvarlist temperature,voltage -appargfilelist
<path to file with temperature data>,<path to file with voltage data>
```

### *About the Settings Window*

In the **Name** field, specify a name for this node that also becomes the name of the input argument.

In addition, the **Settings** window includes the following sections:

## SOURCE

This section contains a tree with a filtered view of the trees in the **Application Builder** window. The nodes either represent some sort of data or have children that do. For an Application Argument, the tree contains all the declarations in the application (that is, all primitive declarations such as Boolean, integer, double, and string scalar and array data types). The value for a command-line argument with the given name will end up in the declaration that you select. The input data will also be automatically converted to the data type of the declaration. An error will occur if the conversion fails, such as if the user of the application specifies 3.7 as the value for a command-line argument where an integer is expected. When you select a node that represents data, the **Use as Source** toolbar button below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. You can also click the **Create New Declaration and Use It as Source** button (  ) in the **Source** section header to create a new variable declaration for the application argument and use it as the source. A **Create and Use Declaration** dialog box opens, so that you can select the data type of the source (if applicable) and its name and initial value. The name cannot be in conflict with any existing variable declaration. Click the **Edit Node** toolbar button (  ) below the tree to move to the corresponding node.

After selecting a node as the source, the node appears as the selected source under **Selected source**.

## HELP TEXT

In the text field in this section you can add some text that will appear as a help text when a user calls the application from the command line using the `-help` argument. For example, if the application's name is `myapp.mph`, the following command displays the help for the application's input arguments:

```
comsol -help myapp.mph
```


# The Main Window Branch

From the **Main Window** node, you can add the following nodes:

- A **Menu Bar** and a **Toolbar**, if you have selected **Menu bar** from the **Menu type** list in the **Settings** window for the **Main Window** node.
- A **File Menu**, if you have selected **Ribbon** from the **Menu type** list in the **Settings** window for the **Main Window** node. A **Ribbon** node is always available, to which you can add **Ribbon Tab** and **Ribbon Section** nodes.

## *The Main Window Node*



---

The **Main Window** node (  ) represents the main window of an application and is the top-level node for the user interface. It contains the window layout, the main menu specification, and an optional ribbon specification.

The **Settings** window for the **Main Window** contains the following sections.

### **GENERAL**

The **Title** field contains the title of the application. The title is the text at the top of the main window. By default, it is the same as the title for the model used to create the application. The **Show filename in title** check box is selected by default. Clear it to exclude the application's filename from the title.

To add an icon for the application other than the default COMSOL icon, specify an image file to use from the **Icon** list, which includes all of the images in the **Images** library, or click the **Add Image to Library and Use Here** button (  ) to locate an image to use on the file system. This image then becomes a part of the **Images** library and is selected as the icon in the main window of the application. The **Default** setting loads `cube.png`. Click the **Export** button (  ) to export the image to the file system (for use in another application, for example).

From the **Menu type** list, select the type of menu for the application.

- Select **Menu bar** (the default) to use a main menu at the top of the application. The **Main Window** branch then includes a **Menu Bar** node and a **Toolbar** node, to which you can add menus and submenus with menu items, actions, and separators.
- Select **Ribbon** to use a Windows-style ribbon toolbar at the top of the application. You can then, under **Main Window**, add a **File Menu** node (for an **Exit** command, for example) and a **Ribbon** node, to which you can add **Ribbon Tab** nodes. Under **Ribbon Tab** nodes, you can add **Ribbon Section** nodes to which you can add menus and submenus with menu items, actions, and separators.

The **Status bar** list controls what to show in the status bar. Select **Progress** when running (the default, which adds a COMSOL progress bar in the lower-right corner of the application; see [Progress Bar](#) for adding custom progress bars) or **None**.

### **MAIN FORM**

This section contains a reference to the form that the main window displays. Select the form to display from the **Form** list.

### **SIZE**

From the **Initial size** list, select how to set the initial size of the application window:

- Select **Maximized** to open the application window in a maximized state that fills the screen.

- Select **Use main form's size** (the default) to adapt the application window size to the size of its main form, which is the form selected in the **Main Form** section above.
- Select **Manual** to specify the application window's initial size in the **Width** (default: 1280 pixels) and **Height** (default: 800 pixels) fields that appear.

Select the **Center on screen** check box to make the app centered in the middle of the computer screen when launched.

## ABOUT DIALOG

In this section, you can specify the contents of the About dialog box and the placement of the link to that dialog box.

From the **Placement** list, choose one of the following options for the placement of the link:


- **Automatic** (the default), which puts the link to the About dialog box in the following place:
  - The File menu, if the **Menu type** list is set to **Ribbon** and there is an added **File Menu**
  - The toolbar, if the **Menu type** list is set to **Menu bar** and there is a toolbar but no menus
  - Else, the lower-right corner
- **File menu** or **Menu bar** (depending on the setting in the **Menu type** list).
- **Ribbon** or **Toolbar** (depending on the setting in the **Menu type** list).
- **Lower-right corner** (an **About** hyperlink in the lower-right corner of the application's main form). If there is no form in the application, it uses the automatic placement instead.

Clear the **Show COMSOL Information** check box if you do not want to include the standard information (logotype, version number, and information about products used).

In the **Custom text** field, add any text that you want to include in the About dialog box. The text appears above the license, patent, and trademark information. If you include a web address, it will appear as a hyperlink in the About dialog box. The web address must be a valid URL that starts with `http://` or `www`.


### *The Menu Bar Node*

---

The **Menu Bar** node () represents the top level of the main menu of the main window of an application, where each **Menu** child node represents a menu on the title bar of the main window. Right-click the **Menu Bar** node to add **Menu** nodes to the main menu. The leftmost menu includes a **Close Application** item by default.


### *The Toolbar Node*

---



The **Toolbar** node () represents a toolbar at the top of the main window of an application (below the main menu), where you can add **Menu** child nodes representing drop-down toolbar menus, **Item** nodes representing toolbar buttons with an action connected to them, and **Separator** nodes to separate groups of related toolbar buttons. Right-click this node to add other nodes to the toolbar.

### *The Menu Node*

---

The **Menu** node () adds another menu level under the parent menu, which can be any other menu type. From a **Menu** node, you can right-click to add another **Menu** node as a submenu. You can also right-click to add an **Item** node for a menu item with an action or a **Separator** node to insert a menu separator (see [The Item Node](#) and [The Separator Node](#)). You can add **Menu** nodes (and **Item** and **Separator** nodes) to additional levels to create additional levels of submenus. Enter the name of the menu object in the **Name** field.


Specify the title of the new menu level in the **Title** field.

For **Menu** nodes directly under a **File Menu** node, a **Toolbar** node, or a **Ribbon Section** node, you can add an icon to the menu. To do so, specify an image file to use from the **Icon** list, which includes all images in the **Images** library, or click the **Add Image to Library and Use Here** button (  ) to locate an image to use on the file system. This image then becomes a part of the **Images** library and is selected as the icon for this menu. If you do not want to use an icon, select **None** from the **Icon** list. Click the **Export** button (  ) to export the image to the file system (for use in another application, for example).

For **Menu** nodes directly under a **Ribbon Section**, you can also select **Large** (the default) or **Small** from the **Size** list. This size controls the size of the button in the ribbon.


---

### *The File Menu Node*

The **File Menu** node (  ) represents a **File** menu in the upper-left corner of an application with a ribbon toolbar. From the **File Menu** node, you can right-click to add a **Menu** node as a submenu. You can also right-click to add an **Item** node for a menu item with an action or a **Separator** to separate groups of related menu items. The File menu should contain standard items such as saving or exiting the application.


---

### *The Ribbon Node*



The **Ribbon** node (  ) represents a ribbon toolbar at the top of the main window of an application, where you can add **Ribbon Tab** child nodes representing ribbon tabs. For the ribbon tabs, you can add ribbon sections with drop-down menus and buttons. Right-click this node to add **Ribbon Tab** nodes for the ribbon.

---

### *The Ribbon Tab Node*


The **Ribbon Tab** node (  ) adds a ribbon tab to a ribbon at the top of the application window. The ribbon tab can have sections that contain the items from menus that you include by adding menus to **Ribbon Section** subnodes. You can add such ribbon sections by right-clicking this node and selecting **Ribbon Section**. Enter the name of the ribbon tab object in the **Name** field.

Specify a **Title** for the ribbon tab.

To add an icon to the ribbon tab, specify an image file to use from the **Icon** list, which includes all images in the **Images** library, or click the **Add Image to Library and Use Here** button (  ) to locate an image to use on the file system. This image then becomes a part of the **Images** library and is selected as the icon for this ribbon tab. If you do not want to use an icon, select **None** from the **Icon** list. Click the **Export** button (  ) to export the image to the file system (for use in another application, for example).

---

### *The Ribbon Section Node*


The **Ribbon Section** node (  ) adds a section to a ribbon. You can right-click **Ribbon Section** node to add **Menu** nodes, **Item** nodes for buttons with a direct action, and **Separator** nodes that define the contents of the ribbon section. Enter the name of the ribbon section object in the **Name** field.




If you use submenus on ribbon menu buttons, they provide headers in the menu, which you can use to group some items, instead of submenus, as in a standard menu bar. This approach works best for ribbons if you do not add top-level menu items below the submenu.

---


Specify a **Title** for the ribbon section.



To add an icon to the ribbon section, specify an image file to use from the **Icon** list, which includes all of the images in the **Images** library, or click the **Add Image to Library and Use Here** button (  ) to locate an image to use on the

file system. This image then becomes a part of the **Images** library and is selected as the icon for this ribbon section. If you do not want to use an icon, select **None** from the **Icon** list. Click the **Export** button () to export the image to the file system (for use in another application, for example).

### *The Item Node*

---

The **Item** node () is a menu option that runs a method as a menu item in a parent menu or as a button on a toolbar. You can add an **Item** node under a **Menu** node or to a **Toolbar** object in a form. Enter the name of the item object in the **Name** field.

In the **Text** field, enter the text to display on the menu item. To add an icon to the menu item, specify an image file to use from the **Icon** list, which includes all of the images in the **Images** library, or click the **Add Image to Library and Use Here** button () to locate an image to use on the file system. This image then becomes a part of the **Images** library and is selected as the icon for this menu item. If you do not want to use an icon, select **None** from the **Icon** list. Click the **Export** button () to export the image to the file system (for use in another application, for example).

For **Item** nodes directly under a **Ribbon Section**, you can select **Large** (the default) or **Small** from the **Size** list. This size controls the size of the button in the ribbon.

For **Item** nodes directly under a **Ribbon Section**, you can also add a tooltip, a descriptive text that displays when the user hovers the pointer over the ribbon button, in the **Tooltip** field.

You can define a shortcut for the item that you enter in the **Keyboard shortcut** field. To add a keyboard shortcut, make the **Keyboard shortcut** field active, and then type a keyboard shortcut on the keyboard.

You must use a modifier in the keyboard shortcut, not just a plain letter (for example, CTRL+SHIFT+D). The shortcut can include the Ctrl key (CTRL), Alt key (ALT), and Shift key (SHIFT). Note that the Ctrl key is interpreted as Command on OS X. Avoid using the following keys in your shortcut:

- Backspace, as it can be used to clear a shortcut
- Delete, as it can be used to clear a shortcut
- Escape
- Alt on its own (to avoid conflicts with File menu shortcuts)



It is possible to override other keyboard shortcuts, so take care when choosing the shortcut key combinations to use.

---

In addition, the **Settings** window contains the following section.

#### **CHOOSE COMMANDS TO RUN**

This section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either support a command or have children that do. When you select a node that supports one or more commands, the corresponding command toolbar buttons become enabled in the toolbar below the tree. You can also right-click a node to get a list of available commands for that particular node. Once you click on a command with a node selected (or press Enter to add a command with its default command such as **Run**, **Plot**, or **Set Value**), the command and node appear in the last row of the table in the **Choose Commands to Run** section of the **Settings** window. This table contains all of the nodes that run. You can delete and move commands using the toolbar below the table.

In the **Model** branch, all of the nodes that represent some sort of data value, such as a parameter under the **Parameters** node, support the **Set Value** command. Add a **Set Value** command to the table to enable the third



column, **Arguments**. In this column you type the value to set. For data that represents arrays, use curly braces and commas to enter the array elements. For example, enter {1, 2, 3} to set a three-element array with the values 1, 2, and 3. See [The Array 1D String Node](#) for more details about how to enter arrays and matrices. For nodes that represent a file import, such as a **Filename** node under an **Interpolation** function node, an **Import File** command is available. You can also add a **Plot** command for all **View** nodes, providing the name of a Graphics object as the argument.







The tree includes a number of branches from the application tree in addition to the **Model** branch:

- The **Forms** branch: **Form** nodes support the commands **Show**, which sets the form as the main form of the application (that is, the content of the application window will be this form), and **Show as Dialog**, which brings up the form as a separate dialog window.
- The **GUI Commands** branch: The commands under this branch are grouped in three subcategories:
  - **File Commands**: These include **Save Application** (to save the application under its current name); **Save Application As** (to open a file browser dialog allowing the user to save the application in a suitable location); **Save Application on Server**; **Save Application on Server As**; **Open File** (to open an application file resource specified using a valid URI path in the **Arguments** column); **Save File As** (similarly, to allow the user to save the file under a name specified in the **Arguments** column); and **Exit Application** (to close the running application). If the application is run on COMSOL Server, the **Save Application on Server** and **Save Application on Server As** commands save the current state as a new application in the COMSOL Server Application Library.
  - **Graphics Commands**: Here you find the commands **Zoom Extents**, **Reset Current View**, **Scene Light**, **Transparency**, **Orthographic Projection**, **Print**, **Select All**, **Clear Selection**, **Show Selection Colors**, and **Show Material Color and Texture**. For all graphics commands, add the name of the Graphics object that you want to apply the command to as an argument.
  - **Model Commands**: Here you find the commands **Clear All Solutions** and **Clear All Meshes**.

Double-click or right-click any of the nodes above to add a **Run** command.


- The **Declarations** branch: This branch contains any variable declarations you have added under the **Application Builder** window's **Declarations** branch grouped by type. Like parameters, they support the **Set Value** command.
- The **Methods** branch: **Method** nodes support the **Run** command.
- The **Libraries** branch: Under **Sounds**, you can choose between sound files to play in a command sequence.



When you click one of the commands underneath the tree, the command appears under **Command** in the list below. There is also a **Symbol** column and an **Arguments** column, where you can enter any applicable arguments that the command uses. A tooltip appears, indicating what type of argument the command expects. For example, for the `downloadtoclient` command, the argument is a file and its path, such as `embedded:///myfile.txt`, and there is a separate dialog box that helps you define such an argument. See [File Schemes and File Handling](#) for more information.

Click the **Convert to Method** toolbar button () and choose **Convert to Method** or **Convert to Local Method** to convert the entire list of commands in the table to a global or local method that contains the equivalent code. After this operation, the list of commands only contains a single **Run** operation on the created method. When you select a method under **Command**, or there is exactly one method in the list, you can go to the editor window for that method by clicking the **Go to Method** button (). For information about the **Edit Argument** button (), see [Editing Initial Values and Arguments in Declarations and Command Sequences](#). Use the **Move Up** () , **Move Down** () , and **Delete** () toolbar buttons to organize and remove commands from the list (and the local method, if deleted).

## The Toggle Item Node

---

The **Toggle Item** node () is a menu option that toggles some state as a menu item in a parent menu or as a button on a toolbar. A toggle item can be useful for switching between a tabbed and a tiled look for a form collection, for example. You can add a **Toggle Item** node under a **Menu** node or to a **Toolbar** object in a form. Enter the name of the toggle item object in the **Name** field.

In the **Text** field, enter the text to display on the toggle menu item. To add an icon to the toggle menu item, specify an image file to use from the **Icon** list, which includes all of the images in the **Images** library, or click the **Add Image to Library and Use Here** button () to locate an image to use on the file system. That image then becomes a part of the **Images** library and is selected as the icon for this menu item. If you do not want to use an icon, select **None** from the **Icon** list. Click the **Export** button () to export the image to the file system (for use in another application, for example).

For **Toggle Item** nodes directly under a **Ribbon Section**, you can select **Large** (the default) or **Small** from the **Size** list. This size controls the size of the button in the ribbon.

For **Toggle Item** nodes directly under a **Ribbon Section**, you can also add a tooltip, which is a descriptive text that displays when the user hovers the pointer over the ribbon button, in the **Tooltip** field.

You can define a shortcut for the item that you enter in the **Keyboard shortcut** field. To add a keyboard shortcut, make the **Keyboard shortcut** field active, and then type a keyboard shortcut on the keyboard.

You must use a modifier in the keyboard shortcut, not just a plain letter (for example, CTRL+SHIFT+D). The shortcut can include the Ctrl key (CTRL), Alt key (ALT), and Shift key (SHIFT). Note that the Ctrl key is interpreted as Command on OS X. Avoid using the following keys in your shortcut:

- Backspace, as it can be used to clear a shortcut
- Delete, as it can be used to clear a shortcut
- Escape
- Alt on its own (to avoid conflicts with File menu shortcuts)






It is possible to override other keyboard shortcuts, so take care when choosing the shortcut key combinations to use.

---

In addition, the **Settings** window contains the following sections.


### SOURCE

This section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either represent some sort of data or have children that do. For a toggle item, strings and Boolean variable declarations (representing on and off states for the toggle item) under **Declarations** are available as sources. When you select a node that represents data, the **Use as Source** toolbar button below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. You can also click the **Create New Declaration and Use It as Source** button () or the **Create New Form Declaration and Use It as Source** button () in the **Source** section header to create a new global or local (in the form) variable declaration for the toggle item and use it as the source. A **Create and Use Variable** dialog box opens, so that you can select the data type of the source (if applicable) and its name. The name cannot be in conflict with any existing variable declaration. Click the **Edit Node** toolbar button () below the tree to move to the corresponding node.

After selecting a node as the source, the node appears as the selected source under **Selected source**.



If you try to use the same data source in several form objects, you may encounter some strange side effects. The default value for the source may not be what you expect. You may also experience serious errors if the default value of one form object is invalid for one of the other form objects.

Typically, you only see the available parameters under the **Parameters** node, variables under a **Variables** node, and the data nodes defined under the **Declarations** branch in the Application Builder part of the application tree (underneath the **Events** branch). You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button () in the **Source for Data Change** section header, which takes you automatically to the Model Builder. Then, select a node in the **Model Builder** branch with data you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Select the check box to include this data as an available source node for events.

Any restrictions on a data source are passed along to the user interface of the application. For example, a **Load type** list in a **Boundary Load** node for structural mechanics only allows three values. Any form object using this data as its source can only support a subset of these values.

For a **Toggle Item** object, you can also specify if the initial value should be selected (on) or cleared (off). From the initial value list, select **Custom value** (the default) or **From data source**. For **Custom value**, choose **Selected** or **Cleared** from the **Initial state** list.

#### CHOOSE COMMANDS TO RUN

This section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either support a command or have children that do. When you select a node that supports one or more commands, the corresponding command toolbar buttons become enabled in the toolbar below the tree. You can also right-click a node to get a list of available commands for that particular node. Once you click on a command with a node selected (or press Enter to add a command with its default command such as **Run**, **Plot**, or **Set Value**), the command and node appear in the last row of the table in the **Choose Commands to Run** section of the **Settings** window. This table contains all of the nodes that run. You can delete and move commands using the toolbar below the table.

In the **Model** branch, all of the nodes that represent some sort of data value, such as a parameter under the **Parameters** node, support the **Set Value** command. Add a **Set Value** command to the table to enable the third column, **Arguments**. In this column, you type the value to set. For data that represents arrays, use curly braces and commas to enter the array elements. For example, enter {1,2,3} to set a three-element array with the values 1, 2, and 3. See [The Array 1D String Node](#) for more details about how to enter arrays and matrices. For nodes that represent a file import, such as a **Filename** node under an **Interpolation** function node, an **Import File** command is available. You can also add a **Plot** command for all **View** nodes, providing the name of a Graphics object as the argument.

The tree includes a number of branches from the application tree in addition to the **Model** branch:

- The **Forms** branch: **Form** nodes support the commands **Show**, which sets the form as the main form of the application (that is, the content of the application window will be this form), and **Show as Dialog**, which brings up the form as a separate dialog window.
- The **GUI Commands** branch: The commands under this branch are grouped in three subcategories:
  - **File Commands**: These include **Save Application** (to save the application under its current name); **Save Application As** (to open a file browser dialog allowing the user to save the application in a suitable location); **Save Application on Server**; **Save Application on Server As**; **Open File** (to open an application file resource specified







using a valid URI path in the **Arguments** column); **Save File As** (similarly, to allow the user to save the file under a name specified in the **Arguments** column); and **Exit Application** (to close the running application). If the application is run on COMSOL Server, the **Save Application on Server** and **Save Application on Server As** commands save the current state as a new application in the COMSOL Server Application Library.

- **Graphics Commands:** Here you find the commands **Zoom Extents**, **Reset Current View**, **Scene Light**, **Transparency**, **Orthographic Projection**, **Print**, **Select All**, **Clear Selection**, **Show Selection Colors**, and **Show Material Color and Texture**. For all graphics commands, add the name of the graphics object that you want to apply the command to as an argument.
- **Model Commands:** Here you find the commands **Clear All Solutions** and **Clear All Meshes**.

Double-click or right-click any of the nodes above to add a **Run** command.


- The **Declarations** branch: This branch contains any variable declarations you have added under the **Application Builder** window's **Declarations** branch grouped by type. Like parameters, they support the **Set Value** command.
- The **Form Declarations** branch: This branch contains any variable declarations you have added under a **Declarations** branch under the current **Form** node. Like parameters, they support the **Set Value** command.
- The **Methods** branch: **Method** nodes support the **Run** command.
- The **Form Methods** branch: **Method** nodes under the current **Form** node support the **Run** command.
- The **Libraries** branch: Under **Sounds**, you can choose between sound files to play in a command sequence.

When you click one of the commands underneath the tree, the command appears under **Command** in the list below. There is also a **Symbol** column and an **Arguments** column, where you can enter any applicable arguments that the command uses. A tooltip appears indicating what type of argument the command expects. For example, for the `downloadtoclient` command, the argument is a file and its path, such as `embedded:///myfile.txt`, and there is a separate dialog box that helps you define such an argument. See [File Schemes and File Handling](#) for more information.


Click the **Convert to Method** toolbar button () and choose **Convert to Method** or **Convert to Form Method** to convert the entire list of commands in the table to a global or form method that contains the equivalent code. After this operation, the list of commands only contains a single **Run** operation on the created method. When you select a method under **Command**, or there is exactly one method in the list, you can go to the editor window for that method by clicking the **Go to Method** button (). For information about the **Edit Argument** button () , see [Editing Initial Values and Arguments in Declarations and Command Sequences](#). Use the **Move Up** () , **Move Down** () , and **Delete** () toolbar buttons to organize and remove commands from the list (and the local method, if deleted).

### *The Separator Node*

---


Add a **Separator** node () under a **Menu** node to add a separator (horizontal line or divider) to the parent menu to separate groups of related menu items. You can also add a separator to a toolbar object in a form.

# The Forms Branch

Under the **Forms** branch () you find all forms that define the application's user interface. You can include a form in the user interface by running a command that shows the form as a dialog. You can do this from a **Button**, **Item**, **Event**, or **Method** node. See [Working with Forms](#) for more information about forms and form objects and the tools available for creating forms.

## *The Forms Node*

---

The **Forms** node () is a placeholder for all **Form** nodes that you add as subnodes. In its **Settings** window, you find the following settings for the appearance of forms.

### **APPEARANCE**

The settings in this section define the text used in the forms that you add, but you can also choose not to inherit the settings to specify a specific style in a form.

From the **Text color** list, select **System** (the default, as defined by the operating system); one of the predefined colors; or select **Custom** to choose a custom text color from the color palette.


From the **Background color** list, select a color to use as the background in the forms: **System** (the default, as defined by the operating system); one of the predefined colors; or **Custom**, which make it possible to select a custom background color from the color palette.



From the **Font** list, choose a text font: **System** (the default, as defined by the operating system) or another font in the list. You may need to choose a font that supports the characters in your local language. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **System** size for the font. By default, form objects inherit the font and font size from these settings.

Under **Applies to new form objects**, select any of the **Bold**, **Italic**, or **Underline** check boxes to make the text appear in any combination of a boldface font, italics, or underlined by default in form objects that you add to the form.

## *The Form Node*

---

Add a **Form** node () to create a *form*, which is a general user interface area for an application. It can, for example, represent the content of a desktop window, a dialog window, or a tabbed pane. A form has to be referenced by another node to be part of the user interface, but you can choose to show a form as a dialog after the user has clicked a button. To edit or test the form, use the following options on the **Form** node's context menu:


- Right-click the **Form** node and choose **Edit** () to open the form window, where you can interactively create and design the form (see [Working with Forms](#)).
- Right-click the **Form** node and choose **Preview Form** () to test the form by opening it as a preview in a separate window that you can inspect.

You can right-click the **Form** node and choose **New Method** to add a form method that is local to the form. You can also add form declarations such as scalar and array strings, Booleans, integers, and doubles that are available as local declarations within the form object.

Enter the name of the form object in the **Name** field.

Specify a **Title** for the form. The default title is **Form 1**, for the first form in the application.

From the **Icon** list, select an icon image to use as the icon for the form when used as a settings form in the **Model Builder**. The **Default** icon is the standard icon for a Form object. Click the **Add Image to Library and Use Here** button

( **+** ) to add any other image to the list of icons and use it. Click the **Export** button (  ) to export the image used for as th Form object’s icon to a file.

The **Show in Model Builder** check box is selected by default. The Form object then appears in the **Settings Form** list in the **Developer** ribbon in the Model Builder mode and can be added as a settings form in the **Model Builder** (see [Creating and Using Settings Forms and Dialogs](#) in the *COMSOL Multiphysics Reference Manual*).



For Settings Forms in the **Model Builder**, **Graphics** form objects are not supported.

---

## SIZE

The size properties are applicable when the form appears in a dialog box. By default, the Application Builder automatically determines the initial size based on the form contents. From the **Initial size** list, choose **Automatic** (the default) or **Manual** to specify the initial size in the **Width** and **Height** fields (default: 40 pixels).

## MARGINS

In this section, you can adjust the form’s **Horizontal** and **Vertical** margins if desired (default: 20 pixels).

## DIALOG SETTINGS

From the **Store changes** list, select **On request** (the default) to store data changes when the user clicks, for example, an OK or Apply button (and where a Cancel button can discard any pending changes); or select **Immediately** to store data immediately when a change is made. This setting applies when the form appears as a dialog. Use the **Immediately** setting to create dynamic dialogs where you, for example, have a direct connection between a slider and an input field.

Select the **Resizable** check box to make it possible for users to resize the dialog if desired.

## SECTION SETTINGS

The **Expandable** check box is selected by default. Clear it if you do not want users to be able to expand and collapse the section. With this setting selected, you also have the option to make the section’s state initially collapsed by selecting the **Initially collapsed** check box.

## SKETCH GRID



The **Sketch Grid** section is only available when you have selected the sketch mode for the form.

---

In this section, you find settings for the grid that you can display in the sketch mode (see [Showing Grid Lines and Snapping to the Grid](#)) and for the snapping of form objects to that grid.

You specify the grid size by entering values in the **Column width** (default: 100 pixels) and **Row height** (default: 20 pixels) fields.

Select the **Align grid to margin** check box to make the grid lines align with the left and top margins.

The **Snap zone** slider controls how exact you need to be when resizing a form object to make it snap to the grid. By default, the snap zone is set to its maximum value so that the object quickly resizes to snap to the grid. Move the slider from **Large** to **Small** to make the snap zone smaller if desired.

Select the **Snap only to grid** check box to make the resizing of form objects snap only to the grid and not to the borders of other form objects, for example.

## GRID LAYOUT FOR CONTAINED FORM OBJECTS



The **Sketch Grid** section is only available when you have selected the grid mode for the form.

There are two tables in this section of the **Settings** window: one for the columns and one for the rows in the grid. In the **Column** and **Row** columns, you find the column and row numbers, respectively, each starting at 1 from the left and from the top. You can control how each row and column fills up the space in the form. Each table has a **Width** (columns) or **Height** (rows) column with lists that contain the following options: **Fit** (the default), **Grow**, and **Fixed**.

- The **Fit** option makes the column or row use the space needed for the contained object to fit. Columns and rows with this setting will not grow in size.
- The **Grow** option makes it possible for the column or row in the grid to expand by using available space in the form when a user of the application increases the size of the form by dragging the corner of the application window, for example.
- The **Fixed** option specifies that the grid layout has a certain width or height for its column or row, specified in the third **Size** column of the table. For the other options, the **Size** column is not applicable and displays N/A. The added width or height in pixels appears in the column or row header in the form's editing window. Columns and rows with this setting will not grow in size.



From the **Inherit columns** list, select a form object from which to inherit its column settings. The default is **None**; that is, the columns settings are not inherited.

### APPEARANCE

In this section, you can control the appearance of the text and background for the form:

From the **Text color** list, select **System** (the default, as defined by the operating system); one of the predefined colors; or select **Custom** to choose a custom text color from the color palette.

From the **Background color** list, select a color to use as the background in the forms: **System** (the default, as defined by the operating system); one of the predefined colors; or **Custom**, which make it possible to select a custom background color from the color palette.

From the **Background image** list, choose a background image if you want to use such an image in the form. The default is **None** for no background image. To add an image to the image library and use it as a background image, click the **Add Image to Library and Use Here** button (  ). Click the **Export** button (  ) to save the background image to a PNG file.




If you choose to use a background image, you can also specify the following alignment settings under **Image position and size**:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, **Fill**, or **Repeat**.
- From the **Vertical alignment** list, choose **Top**, **Middle**, **Bottom**, **Fill**, or **Repeat**.

Choose **Fill** to automatically stretch the background image to fill the form window in the horizontal or vertical direction (or in both directions). Choose **Repeat** to repeat (tile) the images horizontally, vertically, or in both directions.



### EVENTS

In this section, you can connect local methods to events that are triggered when loading and closing the form. The methods can perform some initialization or clean up, for example, when loading and closing the form.

The default in the lists for **On load** and **On close** is **None**, which means that no method runs when an event is triggered for loading or closing the form. Click the **Create Local Method** button (  ) to create a local method for **On load** or **On close**. The corresponding list selection then changes to **Local method**. Click the **Go to Source** button (  ) to open the method editor window, where you can create or modify the code for the **On load** or **On close** method. Click the **Remove Local Method** button (  ) to delete the local method.




# The Events Branch


An event is any activity (for example, clicking a button, typing a keyboard shortcut, loading a form, or changing the value of a variable) that signals a need for the application to carry out one or more actions. Each action can be a sequence of commands of the type described earlier and can also include running methods. The **Events** branch (  ) contains all events that listen to changes on various data entities, such as global parameters or string data. Right-click the **Events** node and choose **Event** (  ) to add events to the application. The main **Events** node contains options for events connected to starting and closing an application.


## *The Events Node*

---

The **Events** node (  ) is the top node under which you can add **Event** nodes to define events. In the **Settings** window for this node, you can add the following events.


### **EVENTS**

From the **On startup** list, select a method that runs before the application window opens. It is therefore not possible to call a plot, for example, or other user-interface-related code (for such methods, you can use an **On load** event for forms). A possible use is to set up some special settings for the application. To add a local method for this event, click the **Create Local Method** button (  ).

From the **About to shutdown** list, select a method that runs before the application closes. You can use it to clear or remove some files, for example. To add a local method for this event, click the **Create Local Method** button (  ). The method you refer to can return a Boolean value. When it does, and the return value is false, the shutdown will be canceled.

## *The Event Node*



---

The **Event** node (  ) adds an event that listens to a change in the runtime model. If a change occurs, it runs an action. It can listen to data field changes, the creation of features, and the removal of features. Enter the name of the event object in the **Name** field.

The **Settings** window contains the following sections.

### **SOURCE FOR DATA CHANGE EVENT**


This section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either represent some sort of data or have children that do. For an event, variable declarations under **Declarations** are available as sources. In addition, under **Model**, global parameters, variables under **Definitions** in a component, and explicit selection nodes are available as sources. With an explicit selection as the source, you can have some method that runs whenever that explicit selection changes (the change can be triggered by code in some method or from user interaction with a selection input or a form object). For example, the application can run a method when the user clicks boundaries in the graphics.

When you select a node that represents data, the **Use as Source** toolbar button below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. You can also click the **Create New Declaration and Use It as Source** button (  ) in the **Source** section header to create a new variable declaration for the event and use it as the source. A **Create and Use Variable** dialog box opens, so that you can select the data type of the source (if applicable) and its name. The name cannot be in conflict with any existing variable declaration. Click the **Edit Node** toolbar button (  ) below the tree to move to the corresponding node.

After selecting a node as the source, the node appears as the selected source under **Selected source**.



If you try to use the same data source in several form objects, you may encounter some strange side effects. The default value for the source may not be what you expect. You may also experience serious errors if the default value of one form object is invalid for one of the other form objects.

Typically, you only see the available parameters under the **Parameters** node, variables under a **Variables** node, and the data nodes defined under the **Declarations** branch in Application Builder part of the application tree (underneath the **Events** branch). You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button () in the **Source for Data Change** section header, which takes you automatically to the Model Builder. Then, select a node in the **Model Builder** branch with data you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Select the check box to include that data as an available source node for events.

Any restrictions on a data source are passed along to the user interface of the application. For example, a **Load type** list in a **Boundary Load** node for structural mechanics only allows three values. Any form object using this data as its source can only support a subset of those values.

### CHOOSE COMMANDS TO RUN

In this section, you choose the commands to run for the event. The section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either support a command or have children that do. When you select a node that supports one or more commands, the corresponding command toolbar buttons become enabled in the toolbar below the tree. You can also right-click a node to get a list of available commands for that particular node. Once you click on a command with a node selected (or press Enter to add a command with its default command such as **Run**, **Plot**, or **Set Value**), the command and node appear in the last row of the table below the tree. This table contains all nodes that run, and you can delete and move commands using the toolbar below the table.

In the **Model** branch, all nodes that represent some sort of data value, such as a parameter under the **Parameters** node, support the **Set Value** command. When adding a **Set Value** command to the table, the third column, **Arguments**, becomes enabled. In this column, you type the value to set. For data that represents arrays, use curly braces and commas to enter the array elements. For example, enter `{1,2,3}` to set a three-element array with the values 1, 2, and 3. See [The Array ID String Node](#) for more details on how to enter arrays and matrices. For nodes that represent a file import, such as a **Filename** node under an **Interpolation** function node, an **Import File** command is available.

The tree includes a number of branches from the application tree in addition to the **Model** branch:

- The **Forms** branch: **Form** nodes support the commands **Show**, which sets the form as the main form of the application (that is, the content of the application window will be this form), and **Show as Dialog**, which brings up the form as a separate dialog window.
- The **GUI Commands** branch: The commands under this branch are grouped in three subcategories:
  - **File Commands**: These include **Save Application** (to save the application under its current name); **Save Application As** (to open a file browser dialog allowing the user to save the application in a suitable location); **Save Application on Server**; **Save Application on Server As**; **Open File** (to open an application file resource specified using a valid URI path in the **Arguments** column); **Save File As** (similarly, to allow the user to save the file under a name specified in the **Arguments** column); and **Exit Application** (to close the running application). If the application is run on COMSOL Server, the **Save Application on Server** and **Save Application on Server As** commands save the current state as a new application in the COMSOL Server Application Library.
  - **Graphics Commands**: Here you find the commands **Zoom Extents**, **Reset Current View**, **Scene Light**, **Transparency**, **Orthographic Projection**, **Print**, **Select All**, **Clear Selection**, **Show Selection Colors**, and **Show Material Color and**







**Texture.** For all graphics commands, add the name of the Graphics object that you want to apply the command to as an argument.


- **Model Commands:** Here you find the commands **Clear All Solutions** and **Clear All Meshes**.

Double-click or right-click any of the nodes above to add a **Run** command.


- The **Declarations** branch: This branch contains any variable declarations you have added under the **Application Builder** window's **Declarations** branch grouped by type. Like parameters, they support the **Set Value** command.
- The **Methods** branch: **Method** nodes support the **Run** command.
- The **Libraries** branch: Under **Sounds**, you can choose between sound files to play in a command sequence.

When you click one of the commands underneath the tree, the command appears under **Command** in the list below. There is also a **Symbol** column and an **Arguments** column, where you can enter any applicable arguments that the command uses. A tooltip appears, indicating what type of argument the command expects. For example, for the `downloadtoclient` command, the argument is both a filename and its path, such as `embedded:///myfile.txt`, and there is a separate dialog box that helps you define such an argument. See [File Schemes and File Handling](#) for more information.

Click the **Convert to Method** toolbar button () and choose **Convert to Method** or **Convert to Local Method** to convert the entire list of commands in the table to a global or local method that contains the equivalent code. After this operation, the list of commands only contains a single **Run** operation on the created method. When you select a method under **Command**, or there is exactly one method in the list, you can go to the editor window for that method by clicking the **Go to Method** button (). For information about the **Edit Argument** button (), see [Editing Initial Values and Arguments in Declarations and Command Sequences](#). Use the **Move Up** () , **Move Down** () , and **Delete** () toolbar buttons to organize and remove commands from the list (and also remove the local method, if deleted).

You can extend the list of available nodes by clicking the **Switch to Model Builder and Activate Data Access** button () in the **Choose Commands to Run** section header, which takes you automatically to the Model Builder, and then selecting a node in the **Model Builder** branch that you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Select the check box to include that data as an available source node for events.


# The Declarations Branch


All nodes under the **Declarations** branch (  ) specify some sort of declaration, typically new data fields that you can bind to various form objects. The **String** node, for example, declares one or more data fields that stores a string value. In that sense, it is equivalent to a global parameter, but for global parameters, the string value has to be a valid expression and the string data field has no such restriction. See below for details about the following declaration nodes, which you add by right-clicking the **Declarations** node and selecting the desired declaration.



- Under **Scalar**:
  - [The String Node](#)
  - [The Boolean Node](#)
  - [The Integer Node](#)
  - [The Double Node](#)
- Under **Array 1D**:
  - [The Array 1D String Node](#)
  - [The Array 1D Boolean Node](#)
  - [The Array 1D Integer Node](#)
  - [The Array 1D Double Node](#)
- Under **Array 2D**:
  - [The Array 2D String Node](#)
  - [The Array 2D Boolean Node](#)
  - [The Array 2D Integer Node](#)
  - [The Array 2D Double Node](#)
- [The Choice List Node](#)
- [The File Node](#)
- [The Unit Set Node](#)
- [The Graphics Data Node](#)



You can only add one of each data type declaration under **Scalar**, **Array 1D**, and **Array 2D**, so after adding such declarations, they disappear from the context menu.

In addition, a **Shortcuts** node (  ) appears if you have added shortcuts to form objects or menu items. See [Adding Shortcuts](#).

See [Editing Initial Values and Arguments in Declarations and Command Sequences](#) for information about tools for editing the initial values in the scalar and array nodes using the window that opens when you click  under the lists of variables.

For all lists of variables in the nodes under **Scalar**, **Array 1D**, and **Array 2D**, click the **Save to File** button (  ) and enter a **File name** in the **Save to File** dialog box, including the extension `.txt`. Click **Save** to save the text file. The information is saved in space-separated columns in the same order as displayed on screen. Use the **Load from File** button (  ) and **Load from File** dialog box to import data in text files, generated by, for example, a spreadsheet

program. Data must be separated by spaces or tabs (or be in a Microsoft® Excel® workbook spreadsheet if the license includes LiveLink™ for Excel®).



Most declaration nodes can also be added as form declarations under **Form** nodes for local use in forms.

## Array Syntax

The default value for arrays can be an array of arbitrary length that you type using a special syntax. An array definition must start and end with curly braces (`{` and `}`), and each element is separated with a comma. To indicate a string, it is good practice to surround it with single quotes (`'`). When you need a special character inside of an array element (including spaces and commas), you must surround the element with single quotes (`'`). If the string itself includes an apostrophe (`'`), use two single quotes (`''`). See the following examples:

ARRAY SYNTAX	RESULTING ARRAY
<code>{1, 2, 3}</code>	A 3-element array with the elements 1, 2, and 3.
<code>{}</code>	An empty array.
<code>{'one, two', 'three by four'}</code>	A 2-element array with elements containing special syntax.
<code>{{1, 2, 3}, {4, 5, 6}}</code>	A 2-element array containing two 3-element arrays (a 2-by-3 matrix).
<code>{{1, 2}, {'one, two', 'Poisson''s ratio'}}</code>	A 2-element array containing two 2-element arrays (a 2-by-2 matrix).

Nonrectangular arrays are possible but are seldom useful in an application context.

The parsing returns a single-level string array when there is one level of curly braces, and a double array (or string matrix) when there are two nested levels of curly braces.

## The String Node

The **String** node (`abc`) declares one or more named strings that you can access from form objects and methods. You can use a string as a source in many of the form objects, such as input fields, combo boxes, and check boxes. The **Settings** window contains the following section:

### LIST OF VARIABLES

This section contains a single table, where you specify one string stored per row. Specify the name in the **Name** column and the initial value for the string in the **Initial value** column. You can also add an optional description of the string in the **Description** column.



Although you can specify the initial value for a string, that value may be overwritten if you use the string as a source to a form object that specifies a different value.

Use the **Move Up** (↑), **Move Down** (↓), and **Delete** (☒) toolbar buttons to organize and remove choices from the list.


### Code Access

In the code of a method, you access a string named `string1` as if it were a normal Java String variable:

```
String s = string1;  
string1 = "newValue";
```

## The Boolean Node

---

The **Boolean** node (  ) declares one or more named scalar Boolean variables that you can access from form objects and methods. You can use a Boolean variable as a source in check boxes, for example. The **Settings** window contains the following section.

### LIST OF VARIABLES

This section contains a single table, where you specify one Boolean variable stored per row. Specify the name in the **Name** column and the initial value in the **Initial value** column (either `true` or `false`; the default is `false`). You can also add an optional description of the Boolean variable in the **Description** column.



Although you can specify the initial value for a Boolean variable, that value may be overwritten if you use the Boolean variable as a source to a form object that specifies a different value.

---

As a string, you can use `on` or `off`, `true` or `false`, or `yes` or `no` (all case insensitive) as Boolean values.

Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) toolbar buttons to organize and remove choices from the list.


### Code Access

In the code of a method, you access a Boolean variable named `bool1` as if it were a normal Java boolean variable:

```
boolean b = bool1;  
bool1 = newValue;
```

## The Integer Node

---

The **Integer** node (  ) declares one or more named scalar integers that you can access from form objects and methods. You can use an integer as a source in an input for some values that must be an integer. The **Settings** window contains the following section.

### LIST OF VARIABLES

This section contains a single table, where you specify one integer stored per row. Specify the name in the **Name** column and the initial value in the **Initial value** column (the default is `0`). You can also add an optional description of the integer in the **Description** column.



Although you can specify the initial value for an integer, that value may be overwritten if you use the integer as a source to a form object that specifies a different value.

---

Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) toolbar buttons to organize and remove choices from the list.

### Code Access

In the code of a method, you access an integer named `int1` as if it were a normal Java `int` variable:

```
int i = int1;  
int1 = newValue;
```

## The Double Node

---

The **Double** node (1.23) declares one or more named scalar double floating-point values that you can access from form objects and methods. You can use a double as a source in some input that is a floating-point number (some scalar value). The **Settings** window contains the following section.

### LIST OF VARIABLES

This section contains a single table, where you specify one double stored per row. Specify the name in the **Name** column and the initial value in the **Initial value** column (the default is 0.0). You can also add an optional description of the double in the **Description** column.



Although you can specify the initial value for a double, that value may be overwritten if you use the double as a source to a form object that specifies a different value.

---

Use the **Move Up** (↑), **Move Down** (↓), and **Delete** (☰) toolbar buttons to organize and remove choices from the list.

### Code Access

In the code of a method, you access a double named `dbl1` as if it were a normal Java double variable:

```
double d = dbl1;  
dbl1 = newValue;
```

## The Array ID String Node

---

The **Array ID String** node (3.65) declares one or more named string arrays that you can access from form objects and methods. The number of elements in the string array is not restricted in any way, and you typically use a string array to store a column in a table. The **Settings** window contains the following section.

### LIST OF VARIABLES

This section contains a single table, where you specify one string array stored per row. Specify the name in the **Name** column, the initial values in the array in the **Initial values** column, and the new element value for each element in the **New element value** column. The new element value specifies the value that a new element of the string array gets in certain add operations (for example, in a table). You can also add an optional description of the string array in the **Description** column.



Although you can specify the initial values for a string array, those values may be overwritten if you use the array as a source to a form object that specifies different values.

---

Use the **Move Up** (↑), **Move Down** (↓), and **Delete** (☰) toolbar buttons to organize and remove choices from the list.


### Code Access

In the code of a method, you access a string array named `array1` as if it were a normal Java `String[]` variable:

```
String[] sa = array1;  
array1 = new String[]{"element1", "element2"};
```

## The Array ID Boolean Node

---

The **Array ID Boolean** node (  ) declares one or more named Boolean arrays that you can access from form objects and methods. The number of elements in the Boolean array is not restricted in any way, and you typically use a Boolean array to specify some list of Boolean values. The **Settings** window contains the following section.

### LIST OF VARIABLES

This section contains a single table, where you specify one Boolean array stored per row. Specify the name in the **Name** column; the initial values in the **Initial values** column; and the new element value for each element (the default, `false`, represents the first value) in the **New element value** column. The new element value specifies the value that a new element of the Boolean array gets in certain add operations (for example, in a table). You can also add an optional description of the Boolean array in the **Description** column.



Although you can specify the initial values for a Boolean array, those values may be overwritten if you use the array as a source to a form object that specifies different values.

---

When the Boolean array is declared as a string, you can use `on` or `off`; `true` or `false`; or `yes` or `no` (all case insensitive) as Boolean values.

Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) toolbar buttons to organize and remove choices from the list.


#### Code Access

In the code of a method, you access a Boolean array named `boolarray1` as if it were a normal Java `boolean[]` variable:

```
boolean[] ba = boolarray1;  
boolarray1 = new boolean[]{value1, value2};
```

## The Array ID Integer Node

---

The **Array ID Integer** node (  ) declares one or more named integer arrays that you can access from form objects and methods. The number of elements in the integer array is not restricted in any way, and you typically use an integer array to specify an array of values that can only be integers. The **Settings** window contains the following section.

### LIST OF VARIABLES

This section contains a single table, where you specify one integer array stored per row. Specify the name in the **Name** column; the initial values in the **Initial values** column; and the new element value for each element (the default, `0`, represents the first value) in the **New element value** column. The new element value specifies the value that a new element of the integer array gets in certain add operations (for example, in a table). You can also add an optional description of the integer array in the **Description** column.



Although you can specify the initial values for an integer array, those values may be overwritten if you use the array as a source to a form object that specifies different values.

---

Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) toolbar buttons to organize and remove choices from the list.

#### Code Access


In the code of a method, you access an integer array named `intarray1` as if it were a normal Java `int[]` variable:



```
int[] ia = intarray1;
intarray1 = new int[]{value1, value2};
```

### The Array 1D Double Node

---

The **Array 1D Double** node () declares one or more named double floating-point arrays that you can access from form objects and methods. The number of elements in the double array is not restricted in any way, and you typically use a double array to specify some array input with floating-point values. The **Settings** window contains the following section.

#### LIST OF VARIABLES

This section contains a single table, where you specify one double array stored per row. Specify the name in the **Name** column; the initial values in the **Initial values** column; and the new element value for each element (the default, 0.0, represents the first value) in the **New element value** column. The new element value specifies the value that a new element of the double floating-point array gets in certain add operations (for example, in a table). You can also add an optional description of the double array in the **Description** column.



Although you can specify the initial values for a double array, those values may be overwritten if you use the array as a source to a form object that specifies different values.

---

Use the **Move Up** () , **Move Down** () , and **Delete** () toolbar buttons to organize and remove choices from the list.


#### Code Access

In the code of a method, you access a double array named `dblarray1` as if it were a normal Java `double[]` variable:

```
double[] da = dblarray1;
dblarray1 = new double[]{value1, value2};
```

### The Array 2D String Node

---

The **Array 2D String** node () declares one or more named 2D string arrays (matrices) that you can access from form objects and methods. The number of elements in the 2D string array is not restricted in any way, but you can specify the number of columns. The **Settings** window contains the following section.

#### LIST OF VARIABLES

This section contains a single table, where you specify one 2D string array stored per row.

Specify the name in the **Name** column, the number of columns in the 2D string array from the list in the **Number of columns** column (use **Undefined** if you do not know how many columns the array contains), the initial values in the **Initial values** column, and the new element value in the **New element value** column. The new element value specifies the value that a new element of the 2D string array gets in certain add operations. You can also add an optional description of the 2D string array in the **Description** column.



Although you can specify the initial values for a 2D string array, those values may be overwritten if you use the array as a source to a form object that specifies different values.

---

Use the **Move Up** () , **Move Down** () , and **Delete** () toolbar buttons to organize and remove choices from the list.


### Code Access

In the code of a method, you access a 2D string array named `matrix1` as if it were a normal Java `String[][]` variable:

```
String[][] sm = matrix1;  
matrix1 = new String[][]{{"element11", "element12"}, {"element21", "element22"}};
```

### The Array 2D Boolean Node

---

The **Array 2D Boolean** node (  ) declares one or more named 2D Boolean arrays (matrices) that you can access from form objects and methods. The number of elements in the 2D Boolean arrays is not restricted in any way, but you can specify the number of columns. The **Settings** window contains the following section.

#### LIST OF VARIABLES

This section contains a single table, where you specify one 2D Boolean array stored per row.

Specify the name in the **Name** column, the number of columns in the 2D Boolean array from the list in the **Number of columns** column (use **Undefined** if you do not know how many columns the array contains), the initial values in the **Initial values** column, and the new element value in the **New element value** column. The new element value specifies the value that a new element of the 2D Boolean array gets in certain add operations. You can also add an optional description of the 2D Boolean array in the **Description** column.



Although you can specify the initial values for a 2D Boolean array, those values may be overwritten if you use the array as a source to a form object that specifies different values.

---

As a string, you can use `on` or `off`; `true` or `false`; or `yes` or `no` (all case insensitive) as Boolean values.

Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) toolbar buttons to organize and remove choices from the list.


### Code Access

In the code of a method, you access a 2D Boolean array named `boolmatrix1` as if it were a normal Java `boolean[][]` variable:

```
boolean[][] bm = boolmatrix1;  
boolmatrix1 = new boolean[][]{{value11, value12}, {value21, value22}};
```

### The Array 2D Integer Node

---

The **Array 2D Integer** node (  ) declares one or more named 2D integer arrays (matrices) that you can access from form objects and methods. The number of elements in the 2D integer array is not restricted in any way, but you can specify the number of columns. The **Settings** window contains the following section.

#### LIST OF VARIABLES

This section contains a single table, where you specify one 2D integer array stored per row.

Specify the name in the **Name** column, the number of columns in the 2D integer array from the list in the **Number of columns** column (use **Undefined** if you do not know how many columns the array contains), the initial values in the **Initial values** column, and the new element value in the **New element value** column. The new element value

specifies the value that a new element of the 2D integer array gets in certain add operations. You can also add an optional description of the 2D integer array in the **Description** column.



Although you can specify the initial values for a 2D integer array, those values may be overwritten if you use the array as a source to a form object that specifies different values.

---

Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) toolbar buttons to organize and remove choices from the list.


#### Code Access

In the code of a method, you access a 2D integer array named `intmatrix1` as if it were a normal Java `int[][]` variable:

```
int[][] im = intmatrix1;
intmatrix1 = new int[][]{{value11, value12}, {value21, value22}};
```

#### The Array 2D Double Node

---

The **Array 2D Double** node (  ) declares one or more named 2D arrays (matrices) of double floating-point values that you can access from form objects and methods. The number of elements in the 2D double array is not restricted in any way, but you can specify the number of columns. The **Settings** window contains the following section.

#### LIST OF VARIABLES

This section contains a single table, where you specify one 2D double array stored per row.

Specify the name in the **Name** column, the number of columns in the 2D double array from the list in the **Number of columns** column (use **Undefined** if you do not know how many columns the matrix contains), the initial values in the **Initial values** column, and the new element value in the **New element value** column (default: 0.0). The new element value specifies the value that a new element of the 2D double array gets in certain add operations. You can also add an optional description of the 2D double array in the **Description** column.



Although you can specify the initial values for a 2D double array, those values may be overwritten if you use the array as a source to a form object that specifies different values.

---

Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) toolbar buttons to organize and remove choices from the list.


#### Code Access

In the code of a method, you access a 2D double array named `dblmatrix1` as if it were a normal Java `double[][]` variable:







```
double[][] dm = dblmatrix1;
dblmatrix1 = new double[][]{{value11, value12}, {value21, value22}};
```

#### The Choice List Node

---

The **Choice List** node (  ) contains a list of choices for combo boxes, lists, and radio buttons that refer to this node as part of their allowed values. You can specify a label to display in the **Label** field and the name of the choice list object in the **Name** field. The **Settings** window contains the following section.

## LIST CONTENTS

This section includes a table with a **Value** column and a **Display name** column. Enter the property values that the user can choose from in the first column and the corresponding text to show in the combo box list in the second column. Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) toolbar buttons to organize and remove choices from the list. Click the **Clear Table** button (  ) to remove all table contents. Click the **Save to File** button (  ) and enter a **File name** in the **Save to File** dialog box. Click **Save** to save the file. Use the **Load from File** button (  ) and **Load from File** dialog box to import data in text files, generated by a spreadsheet program, for example.




If the data source of the choice list is a global parameter or variable, it is recommended that you use plain numbers as values. Otherwise, you have to make sure that values can be evaluated as a valid expression. An invalid expression for a global parameter generates an error.

Internally, a choice list is an  $N$ -by-2 double string array, where  $N$  is the number of rows in the table. When running an application, you can change the content of this list by setting a new double string array to the property with the name you specified in the **Name** field.



You can right-click the **Choice List** node to add an **Activation Condition** subnode.

### *Activation Condition*

The **Activation Condition** node (  ) is a condition that can evaluate to true or false, which decides if the parent node is active or inactive. For example, if the parent is a **Choice List** node, a false activation condition for a choice list excludes the list from the combo box that uses it. You can specify a label to display in the **Label** field and the name of the activation condition object in the **Name** field.

The **Settings** window of the activation condition contains the following sections.


## SOURCE


This section contains a tree with a filtered view of the tree in the **Application Builder** window. The nodes either represent some sort of data or have children that do. For an activation condition, variable declarations under **Declarations** are available as sources. When you select a node that represents data, the **Use as Source** toolbar button below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. You can also click the **Create** button (  ) in the **Source** section header to create a new variable declaration for the form object and use it as the source. A **Create and Use Variable** dialog box opens, so that you can select the data type of the source (if applicable) and its name. The name cannot be in conflict with any existing variable declaration. Click the **Edit Node** toolbar button (  ) below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.

After selecting a node as the source, the node appears as the selected source in the **Selected source** label.




If you try to use the same data source in several form objects, you may encounter some strange side effects. The default value for the source may not be what you expect. You may also experience serious errors if the default value of one form object is invalid for one of the other form objects.

Typically, you only see the available parameters under the **Parameters** node, variables under a **Variables** node, and the data nodes defined under the **Declarations** branch. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button (  ) in the **Source** section header, which automatically takes you to the Model Builder. Then, select a node in the **Model Builder** branch whose data you want to access. With this button active, the **Settings** window of the selected node displays a green check box next to the settings that you can include by selecting that check box (for example, in a **Boundary Load** node in a solid mechanics model). The data


from that node then appears as nodes with the icon  in the tree under **Source**. Any restrictions on a data source are passed along to the application's user interface. The **Load type** list in a **Boundary Load** node, for example, only allows three values (**Load defined as force per unit area**, **Total force**, and **Pressure**), and any form object using this data as its source can only support a subset of those three values.

#### CONDITION

In the table under **Condition**, enter the values to check against the value of the data field. A matching value normally causes the condition to be true, but select the **Invert condition on input values** check box to invert the condition. Click **Delete** () to remove an activating value from the table.

### *The File Node*

---

The **File** node () makes it possible to refer to files when developing an application. You can declare an external file that can be imported into the application or uploaded from the client at runtime. The File declaration uploads the file to the application at runtime. It is possible to access it using, for example, the file scheme `embedded:///file1`, if the name of the File declaration is `file1` (you enter the name in the **Name** field of the **Settings** window). For more information about the file schemes, see [File Schemes and File Handling](#). To make it possible for a user to browse and select a file, use a **File Import** object and select the File node as the source. You can also enter a label to display in the **Label** field.

The **Settings** window for the file declaration contains the following section.


#### FILE LOCATION

From the **Target directory** list, choose one of the following directories, depending on the type and use of a file:

- Select **User** to store it in the user's directory for the application.
- Select **Temporary** (the default) to store it in a temporary file only as long as the application is active.

### *The Unit Set Node*

---

The **Unit Set** node () provides a way of defining sets of units with a list of units for applicable physical quantities. Defining unit sets makes it possible for the user of an app to select the unit to use for some input. Input Field and Slider form objects can use a specific unit list of a unit set to specify the display unit of the form object. Typically, you can use a combo box object to specify the unit set to use (by making it the source). You can also use a unit set as the source in radio button, list box, and combo box form objects.

The **Settings** window contains the following sections.

#### UNIT GROUPS

In this table, add the units that you want to use in the application, either for multiple quantities in a centralized unit set or for a single quantity using multiple **Unit Set** nodes, one for each quantity. In the **Value** column, type the value that you want to use for the unit, and in the **Display name** column, type the name of the unit that will appear when it is used in a combo box object, for example. The value and display name can be the same (SI or `inch`, for example). Each unit row that you add in this section adds a corresponding column in the table under **Unit Lists**.





Use the **Move Up** () , **Move Down** () , and **Delete** () toolbar buttons to organize and remove choices from the list.

From the **Initial value** list, choose the unit set that will be used when launching the application.

#### UNIT LISTS


This section contains a table, where you add physical properties as rows and their corresponding units for all defined unit groups in the columns. Specify the name in the **Name** column (`mass` or `length`, for example) and then the

corresponding units in the following columns; for example, kg and m in an **SI** column for SI units and lb and ft in an **Imperial** column for imperial units. Or, if using a separate **Unit Set** node for a length, for example, the columns could be cm, m, and in for centimeters, meters, and inches, respectively).

Click the **Add** button (  ) to add another physical property as a new row to the list. Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) toolbar buttons to organize and remove rows from the list.

## *The Graphics Data Node*

---

The **Graphics Data** node (  ) contains options for all properties used for data picking in the Graphics windows.

The **Settings** window contains the following sections.

### **INITIAL VALUES**

Enter the initial coordinates for the data as comma-separated values in the **Coordinate** field.

Enter the initial value for the results evaluation in plots in the **Results evaluation** field.

### **INITIAL VALUES FOR 3D GEOMETRY SOURCE**

In this section you define the 3D geometry source for 3D probes.

From the **Geometric entity level** list, choose **Domain** (the default) or **Boundary**.




Under **Domain settings**, choose a **Line entry method**: **Point and surface normal**, **Point and direction**, **Two points**, or **None**. For the **Depth** along line, enter a value between 0 and 1, and for the **Point being modified**, choose **First point** or **Second point**.

The values defined here for the results evaluation, geometric entity level, depth along line, and point being modified are available as sources in, for example, input field objects. They are also available with Set Value commands in the **Choose Commands to Run** section for buttons, for example.

## *Adding Shortcuts*

---

You can create shortcuts to form objects, to toolbars, menu items, and ribbons, and to most model nodes in the **Model Builder**. Shortcuts are available in application method code as Java variables, like other declarations, but the variables for shortcuts are read-only variables.





In the **Settings** window for such objects and model nodes, there is a **Create Shortcut** button (  ) to the right of the **Name** field (**Label** field in the **Model Builder**). When you click that button, a **Create Shortcut** window opens, where you can edit the **Name** of the shortcut (by default, it is the same as the name of the object for which you create the shortcut). You can also create a shortcut using Ctrl+K. Created shortcuts appear in the **Shortcuts** node (  ) under **Declarations**. You can only create one shortcut for each object. If you try to create another shortcut when one exists already, click the **Rename Shortcut** (  ) button. The **Rename Shortcut** window then opens, where you can edit the name of the shortcut.

Shortcuts are automatically updated when you rename, move, copy, or duplicate objects.


The **Shortcuts** node's **Settings** window contains the following section.

### **LIST OF SHORTCUTS**

In the **Name** column, you can edit the names of the shortcuts, if needed. The **Target** column lists the full names of the target objects. In the **Description** column, you can edit the descriptions of the shortcuts.

Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) toolbar buttons to organize and remove shortcuts from the list. Click the **Go to Source** button (  ) to move to the object to which the selected shortcut refers.

## Editing Initial Values and Arguments in Declarations and Command Sequences

The **Edit Initial Value**, **Edit Initial Values**, or **Edit Argument** button (  ) is available under the tables in declarations and command sequences. You find the **Edit Initial Value** or **Edit Initial Values** button under the tables in scalar, array 1D, and array 2D declaration nodes. The same button, but as **Edit Argument**, is available in the **Settings** window for the following nodes that support a command sequence: **Button**, **Event**, and **Item** nodes. It is enabled for commands that use arguments. The **Edit Argument** is also available for defining inputs to methods.

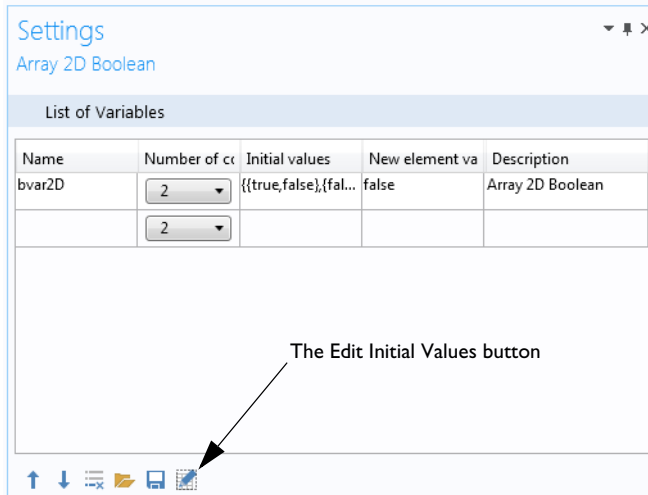


Figure 2-2: The Edit Initial Values button in the Settings window of an Array 2D Boolean node.

When you click the **Edit Initial Values** button, you can edit the selected row of the **Initial values** column in the window that opens. For example, the following image shows how you can edit the rows and columns of the Boolean 2D array.

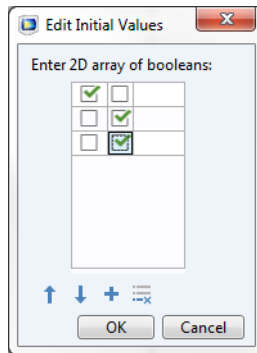




Figure 2-3: The Edit Initial Values window for an Array 2D Boolean node.

When you are done and click **OK**, the initial values are pasted into the **Initial values** column with proper array syntax. For example, the edits above produce the following array:

```
{{true,false},{false,true},{false,false}}
```

You can also use the **Load** (  ) and **Save** (  ) buttons to load the initial values from a text file or save them to a text file.

For command sequences, the toolbar button works in a similar manner. The difference is that this table supports more types of data to enter in the **Arguments** column. The **Set Value** command on data sources shows identical

windows as the nodes under **Declarations**. Other commands, like plotting a plot group or evaluating data into a result table, use their own **Edit Argument** windows.

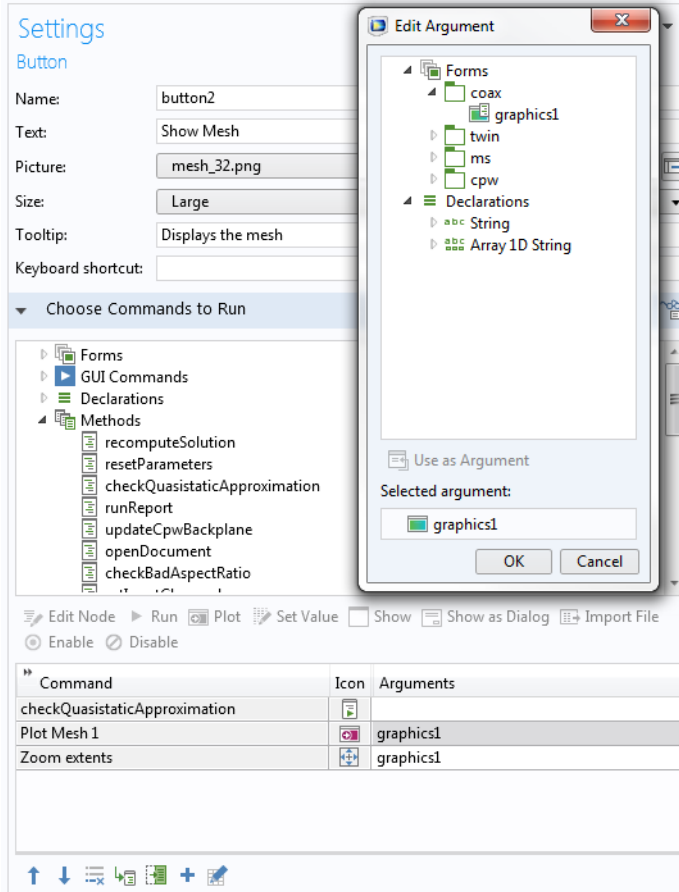


Figure 2-4: A command sequence with a plot command and its Edit Argument window.



For some command sequence commands, you can use declarations in the arguments. Such declarations are possible to use for commands to plot graphics, view built-in commands, run numerical features, set declarations, and run methods. For the commands to plot graphics, view built-in commands, and run numerical features, you can use this functionality in the **Edit Arguments** dialog box. You can also enter the arguments manually. To access a scalar, an array as an array, or a matrix as a matrix (as opposed to accessing individual array or matrix elements), use the name of the declaration itself (`output`, for example). Using an array as a scalar or a matrix as an array is done by adding an index (`output (1)`, for example). A matrix element can be retrieved as a scalar by adding two indices (`output (1,3)`, for example). You can use other variable declarations when specifying the indices (`output (n)`, for example, where `n` is a scalar integer declaration).

For the Plot graphics command and the View built-in commands, you can only use string-type declarations (with appropriate indices if necessary) containing a path to a graphics object. For the other cases, all types of declarations, with or without indices, can be used as long as there is a reasonable interpretation of the source declaration dimension in the target declaration (or method argument) dimension. If, in a plot command, there is a graphics object named `graphics1` and a string declaration named `graphics1`, for example, the content of the declaration will be used, unless single quotes like `'graphics1'` are used, in which case the graphics object is used. You can add a declaration (integer, double, or string) as an argument to the run numerical feature command. In that case, the table data, stripped of the initial parameter columns, which is produced by evaluating the numerical feature, will be



put in that declaration when the command is run. The command checks that the dimensions agree. When converting a command sequence into a method, those checks are generated into code.





In the windows used for editing the default value or setting values of a 2D array that has an undefined number of columns (such as the **Edit Argument** window), you can enter values incrementally by clicking the **Add Row** (  ) and **Add Column** (  ) buttons as required to create a 2D array with some combination of rows and columns.

---

# The Methods Branch




## *The Methods Branch*

---

The **Methods** branch (  ) contains references to any user-defined methods that you have added to the application. Right-click the **Methods** node and choose **New Method** to add a new method. From form objects that support methods that run in response to commands, you can add *local methods* that are defined in the **New Method** window that opens. Enter a method name in the **Name** field and then click **OK**. A new **Method** node (  ) with that name is then added, and an editor window for the method opens, where you can define the code for the new method (see [Working with Methods](#)).

## *The Method Node*

---

A **Method** node (  ) represents a method — a small program that runs when called from other nodes, such as **Event** nodes, **Item** nodes, **Buttons** nodes, and other **Method** nodes. The name of the **Method** node is the name of the method, which you can enter in the **Name** field. Right-click and choose **Edit** (  ) to open an editor window where you can edit the method. For local methods, click the **Go to Source** button (  ) to move to the form object from which the local methods is created. See [Creating Methods](#) and [The Method Nodes and Method Editor Windows](#) for information about creating methods.

Methods can also be used to run code in connection with a model defined in the **Model Builder** window. The **Show in Model Builder** check box is selected by default. The method then appears in the **Run Method** list in the **Developer** ribbon.




The **Show in Model Builder** check box is only active for global methods. Form methods and local methods are not possible to show in the Model Builder. Furthermore, for local methods, no settings are editable in the **Methods** node's **Settings** window.

---

The **Settings** window for a **Method** node contains the following section.

### **INPUTS AND OUTPUT**

In the table under **Inputs**, you can add one or more inputs to the method. Click the **Add** button (  ) to add an argument to the list.


In the **Name** column, you can edit the name of the argument (default: `arg1`, and so on). In the **Type** column, choose a data type (**String**, **Boolean**, **Integer**, **Double**, or a 1D or 2D array type). You can assign a default value in the **Default** column, and enter a description of the input in the **Description** column. The descriptions can be used to annotate the method parameters and as a label for inputs in **Method Call** features.

For method inputs of the double types, there is an optional unit definition that you define in the **Unit** column. The unit definition is used by **Method Call** features to append a label to the input text field and to convert values to the desired unit when running the method call.

Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) toolbar buttons to organize and remove commands from the list.

From the **Output** list, select an output data type: **None** (the default, for no output); **String**; **Boolean**; **Integer**; **Double**; or a 1D or 2D array type. For all output data types, you can enter a name for the method's output in the **Name** field (default: `out`). The value of the output variable is what the method returns. If it is unassigned, the method returns the default value for the type (`0`, `null`, or `false`).


# The Libraries Branch

The **Libraries** branch (  ) contains libraries of images, sounds, and files that you have added to the application. Some libraries also have some predefined examples available for you to use. You can also right-click the **Libraries** node and choose any of the following library components:


- Utility classes. See [The Utility Class Node](#).
- External Java libraries. See [The External Java Library Node](#).
- External native code libraries based on, for example, C code. See [The External C Library Node](#).





## *Images*

---

The **Images** node (  ) contains a library of images (as .bmp, .gif, or .jpg files, for example) that you can use in an application. The **Settings** window contains the following section.

### LIST OF IMAGES

The list contains all images available for use in an application in the **Name** column. You can add an optional description in the **Description** column. A few sample images are always included. To add an image file, click the **Add File to Library**  button, which opens a file browser where you can select and add an image to the list. Click any of the column headers to sort the list based on that column. There are three sorting modes: alphabetical, reverse alphabetical, and the original list order.

Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) buttons to edit and arrange the list of images. Click the **Export Selected Image File** button (  ) to save the selected image file in the image library to the file system.


You can use an image as an icon on a button, for example. Then select an image, such as `plot.png`, from the **Icon** list in the **Settings** window for a **Button** or **Toggle Button** object.

### PREVIEW

This area shows a preview of the selected image in the list of images above.




## *Sounds*

---


The **Sounds** node (  ) contains a library of sound files (.wav files) that you can use in an application. The **Settings** window contains the following section.

### LIST OF SOUNDS


The list contains all sound files available for use in an application in the **Name** column. You can add an optional description in the **Description** column. A few sample sounds are always included. To add a sound file, click the **+** button, which opens a file browser where you can select and add a sound to the list. Click any of the column headers to sort the list based on that column. There are three sorting modes: alphabetical, reverse alphabetical, and the original list order.





Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) buttons to edit and arrange the list of sounds. Click the **Preview** button (  ) to listen to the sound. Click the **Export Selected Sound File** button (  ) to save the selected sound file in the sound library to the file system.

You can use a sound to indicate some action connected to a button, for example. To play a sound when the application user clicks a button, connect the button to a method where you add a line of code. The method should call on the name of the sound file. For example, to play a sound from the file named `success.wav`, add the following line to the method: `playSound("success");`

The **Files** node (  ) contains a library of data files that you can use in an application. There are no file type restrictions for the files that you can add to this library. The **Settings** window contains the following section.

### LIST OF FILES

The list contains all files available for use in an application in the **Name** column. The **Copied from** column contains the full path and name of the file that the application copies. You can add an optional description in the **Description** column. To add a file, click the **Add File to Library**  button, which opens a file browser where you can select and add a file to the list. Click any of the column headers to sort the list based on that column. There are three sorting modes: alphabetical, reverse alphabetical, and the original list order.

Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) buttons to edit and arrange the list of files. Click the **Export Selected File** button (  ) to save the selected file in the library to the file system.

In the application, you can refer to a file that you have added using `embedded:///data.txt`, for example (to refer to the file `data.txt`). See [File Schemes and File Handling](#) for more information about the `embedded` file scheme.

# Planning and Preparing an Application

The following section provides some tips on what to think about when you plan and prepare an application. Typically, you have a COMSOL Multiphysics model or set of COMSOL Multiphysics models that provide a starting point for the development of an application.

## *Preparing an Application*

---

If you want to get the most out of the Application Builder, consider these steps and guidelines while creating your application.

Consider what the application should include. What parameters are of interest? What are the outputs that a user would like to see? Plots are useful, but usually some specific numerical results are important, such as a maximum stress, displacement, or temperature.

Make a sketch of the user interface and its controls and objects, outlining what types of inputs, menus, buttons, plots, and so on the application should include. Make a layout of the form or forms in the application.

Create a COMSOL Multiphysics model of the application, including the parameters that you want to use as inputs in the applications and the derived values and plots that you want to use as the outputs and results. To differentiate the derived values and plots, rename the nodes so that they have descriptive names.

Use descriptions for all of the parameters that you want to include in the application. You can then select them as inputs in the New Form wizard, which uses the descriptions as labels for the corresponding text fields.

Solve the COMSOL Multiphysics model and consider what studies you want to include in the application to produce the output that is of interest to the users.

Save the COMSOL Multiphysics model as an MPH-file that you can use as a starting point for the application.

The Application Builder includes built-in tools for tailoring an application that suits most needs. Consider whether there are some special aspects of the application that would require coding, and if so, prepare some information about what objects and properties you need to use. The next step is to start the Application Builder.

When using several plots to show various aspects of the solution, consider whether all plot buttons should also recompute the solution. Alternatively, you can use a separate button to compute the solution and just let the plot buttons update the plots using the current solution data.

# Creating Applications from Models

The following sections contain information about tools that can help you extract code from a model and test the application.

## *Copy as Code to Clipboard*

To create custom methods for an application, you can copy the contents of a node in the embedded model as the corresponding code. If you select a node in the embedded model and right-click, the context menu contains a submenu called **Copy as Code to Clipboard**.

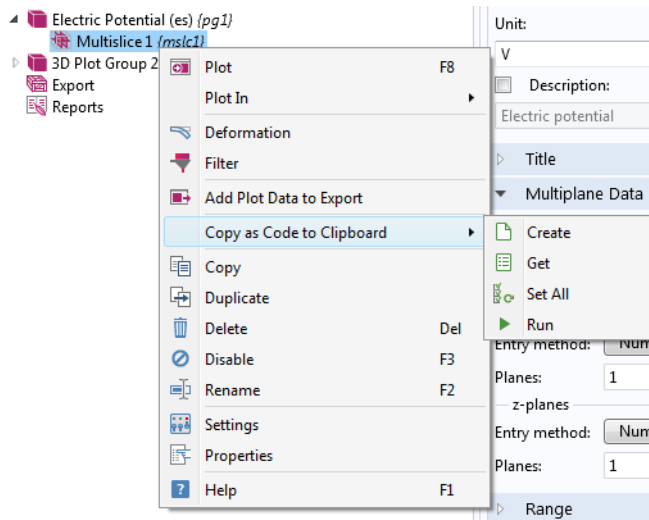


Figure 2-5: The options on the *Copy as Code to Clipboard* submenu.

The submenu contains up to four entries. All entries copy source code to the system clipboard. More specifically:

- **Create** (when applicable), to generate a code snippet to create the corresponding object.
- **Get**, to generate a code snippet to access the corresponding object.
- **Set All**, to generate a code snippet to set the nondefault properties in the corresponding object.
- **Run**, (when applicable) to generate a code snippet to run the corresponding sequence.

The **Create** and **Run** actions are only applicable for nodes that you can create and that you can run, respectively.

The **Copy as Code to Clipboard** actions are intended to make it easier to write code. It can sometimes be hard to figure out what code to write to access a particular model entity, but by using **Copy as Code to Clipboard>Get** and then pasting the result into a method editor window, you do not need to know the corresponding code. Similarly, to figure out how to set a particular property in a model entity, you can change the property in the embedded model, use **Copy as Code to Clipboard>Set All**, paste the result into a method editor window, and possibly edit the inserted code. The **Create** and **Run** actions provide the same functionality but for the tasks of creating and running the feature, respectively.



**Copy as Code to Clipboard>Set All** only considers properties that have been changed directly in the feature's settings. In particular, it does not consider properties set by inserting a geometry sequence.

## Testing the Application

On the **Home** ribbon tab, click **Test Application** (▶) in the **Test** section. A second desktop window opens, and a test run of your application starts. If you click the **Test Application** button again, the running application is replaced by a fresh copy of your application, taking any recent changes you have made in the builder into account.



If you work with your running application and then click **Test Application** again in the builder, any unsaved work you have made to your application test run will be lost.

You can also test the application to see how it runs in a web browser. Click the **Test in Web Browser** button (▶) to launch the application in a web browser using COMSOL Server. The application opens in a new browser window in the default browser. Click the down arrow in the bottom-right corner of the **Test in Web Browser** button (▶) to open a menu where you can choose to test the application in Google Chrome™, Firefox®, or Internet Explorer® browsers, if you have installed those browsers.

### APPLYING CHANGES TO A RUNNING APPLICATION

Some changes that you make to your builder model can be applied to the running application without restarting it. This can be convenient if you are developing methods that you need to test or if you want to try out layout changes. A hot code swap mechanism is used to update the running application with new code and settings from the builder application. If there is a conflict between the running application and the builder model, the hot code swap can fail and you have to restart the application instead.

To apply changes, go to the **Application** ribbon tab and click **Apply Changes** (↻). You can add, edit, and remove methods, events, declarations, forms, and widgets. This includes layout changes, editing command sequences, and labels. You can also use this button to change the model — for example, the plot settings. Note that when you change the model, there is a risk of conflicts with the model in the running application, preventing the hot code swap. If this conflict is detected before any code has been swapped into the running application, you can choose to continue running the application without applying the changes or to restart the application.

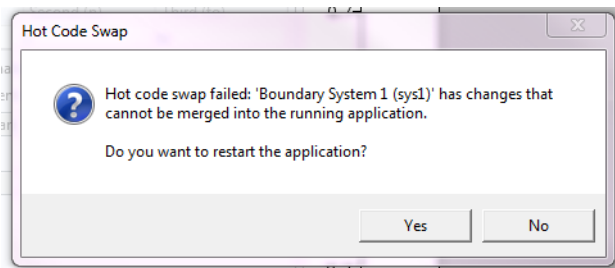
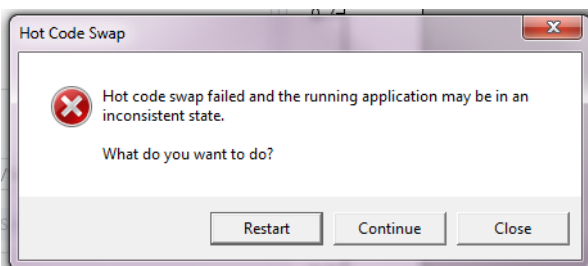


Figure 2-6: Error message when the hot code swap fails.

The conflicts can go undetected until some code has already been swapped into the running application. You can continue running the application, but its state is generally undefined. You can also choose to continue running, restart, or close the application.



# Keyboard Shortcuts

The following table lists the keyboard shortcuts generally available in the Application Builder or for use when working with a form in a form window or with a method in a method editor window.

TABLE 2-1: KEYBOARD SHORTCUTS

SHORTCUT	ACTION	GENERAL	FORM WINDOWS	METHOD EDITORS
Ctrl+A	Select all	√	√	√
Ctrl+D	Deselect all	√		
Ctrl+C	Copy	√	√	√
Ctrl+V	Paste		√	√
Ctrl+X	Cut		√	√
Del	Delete	√	√	√
Ctrl+N	Create a new application	√	√	√
Ctrl+S	Save an application	√	√	√
Ctrl+F8	Test an application	√	√	√
Ctrl+Shift+F8	Apply changes	√	√	√
Ctrl+R	Record code			√
F1	Display help	√	√	√
F2	Rename applicable nodes	√		
F3	Disable applicable nodes	√		
F4	Enable applicable nodes	√		
F11	Go to model node			√
Ctrl+I	Create local variable declarations			√
Ctrl+7	Toggle comment on and off			√
Ctrl+up arrow	Move applicable nodes up	√		
Ctrl+down arrow	Move applicable nodes down	√		
Ctrl+Z	Undo	√	√	√
Ctrl+Y (Ctrl+Shift+Z on Mac)	Redo	√	√	√
F5	Continue (in debugger)			√
F6	Step (in debugger)			√
F7	Step into (in debugger)			√
F8	Check syntax			√
Ctrl+F	Find search term in application, and search and replace text in methods	√		√
Ctrl+K	Create, use, or rename a shortcut to an object or menu	√	√	
Ctrl+space, Ctrl+/	Autocomplete method code			√
Ctrl+M	Move cursor between matching braces			√
Ctrl+Shift+M	Select text between matching braces			√
Ctrl+U	Make selected code lowercase			√
Ctrl+Shift+U	Make selected code uppercase			√
Ctrl+B	Toggle breakpoint on selected line			√



TABLE 2-1: KEYBOARD SHORTCUTS

SHORTCUT	ACTION	GENERAL	FORM WINDOWS	METHOD EDITORS
Ctrl+scroll wheel up	Zoom in, in method code window			√
Ctrl+scroll wheel down	Zoom out, in method code window			√
Ctrl+all arrow keys	Nudge position of form objects a little in sketch mode		√	
All arrow keys	Nudge position of form objects more in sketch mode; traverse and select cells in grid mode		√	
Shift+all arrow keys	Select multiple cells in grid mode		√	
Ctrl+Alt+A	Return to the Application Builder window	√		
Ctrl+Alt+M	Return to the Model Builder window	√		
Ctrl+Alt+double-click	Open the editor window for a method called from another method			√
Alt+F4	Close window	√	√	√
Ctrl+F4	Close a form or method window		√	√
Ctrl+Shift+F4	Close all form and method windows		√	√
Ctrl+Pause (Command+. on Mac)	Stop running a method when testing running applications.	√		



## Working with Forms

In this chapter, you will learn about the tools for creating forms, which are custom user interfaces for apps. Forms can have various form objects, such as input fields, graphics, and buttons. The Application Builder provides interactive tools for sketching and arranging form objects to create forms for your application.

In this chapter:

- [Introduction](#)
- [The Form Objects](#)



# Introduction

## *Overview of the Forms and Tools for Creating Forms*

---

The form tools in the Application Builder make it possible to interactively create and edit forms (also called panes or windows) with graphics and various form objects such as lists, text input fields, check boxes, and buttons. Forms can also be used as settings forms in the Model Builder (see [Creating and Using Settings Forms and Dialogs](#) in the *COMSOL Multiphysics Reference Manual*).


The desktop environment in the Application Builder contains the following parts:


- A **Form** contextual ribbon toolbar with buttons for creating new forms, inserting form objects, and controlling the grid and the layout.
- An **Application Builder** window, which contains the application tree without the **Model** branch.
- One or more *form windows* (editor windows), where you can insert graphics and other form objects and design the form interactively. Each form in the application has its own form window. You can control the maximum number of editor windows that can be open at the same time using the **Maximum number of editors before closing** setting on the **Application Builder** page in the **Preferences** dialog box. To disable checking for the number of editors, clear the associated check box.
- There is also a **Settings** window with settings for the selected form or form object. Use these settings for editing properties of the forms and form objects. To open or close this window, click the **Settings** button (  ) in the **Home** ribbon toolbar. If you select multiple form objects, the **Settings** window for **Multiple Objects** includes common settings for all selected form objects, typically in the **Position and Size** and **Appearance** sections.
- A separate **Form I** window (the name reflects the name of the form that you are previewing) displays a preview of the form so that you can check its layout and design. To open and close it, click the **Preview Form** button (  ) in the **Form** ribbon toolbar.

## *Working with a Form and Using the New Form Wizard*

---

You can start working with a form and access the **Form** ribbon toolbar in the following ways:

- Click to select an existing form window.
- Right-click a **Form** node in the application tree and select **Edit** (  ).

To create a new form, click the **New Form** button (  ). Then, use the wizard in the **New Form** window that opens.

The wizard takes you through the following steps.

- 1** In the **New Form** window, you can create a new form with basic form objects (components). For all added form objects, the **Preview** section to the right displays what the form will look like. You can change the label for the form in the **Form title** field (default: Form 1 for the first form) and its name in the **Form name** field (default: form1 for the first form). Select the **Labels on top** check box to put the labels for input fields and data display objects above the input or display instead of to the left of it. You can make additional adjustments and modifications in the form later.
- 2** Click the **Inputs/outputs** tab to add input fields where users can type in values of parameters used as inputs and numerical data displays where you can present numerical results. Under **Available**, you can select applicable variables defined for the application under **Declarations**, any parameter in the model under **Global Definitions**, and any variable in the model under **Component>Definitions** to define an input field. You can also add other model properties using [Data Access](#). Depending on the property, the input becomes a label and a text field, check box,

or combo box. The default labels are the descriptions defined for the parameters and variables. If possible, the wizard creates an input field with the setting **Append unit to number**.

You can also choose any evaluated value in the model under **Results>Derived Values** to display as numerical data and **Table** and **Evaluation Group** nodes (with tabulated data from derived values). Press Enter, double-click the selected input or output, or click the **Add Selected** → button to add the input field or data display to the form and the list under **Selected**. Press Enter, double-click, or click the **Remove Selected** ← button to remove an item from the **Selected** list.

**3** Click the **Graphics** tab to add any of the following Graphics windows (plots) to the form:


- Under **Component>Definitions**, add any available explicit selection node as the source for the initial graphics content in the graphics object.
- Under **Component**, add **Geometry** to plot the geometry.
- Under **Component**, add **Mesh** for a plot of the mesh.
- Under **Results**, add any of the available plot groups and nodes under **Export**.

Double-click or click the **Add Selected** → button to add the graphics to the form and the list under **Selected**.





Double-click or click the **Remove Selected** ← button to remove an item from the **Selected** list.

4 Click the **Buttons** tab to add buttons to the form. By default, the added buttons are large buttons. You can add buttons for the following actions that the user can perform in the application for the **Model** branch:




- Under **Component**, add **Plot Geometry** to add a button for updating a plot of the geometry or **Plot Mesh** to add a button for updating a plot of the mesh.
- Under **Component**, add **Plot Mesh** to add a button for updating a plot of the mesh.
- Add **Compute Study** to add a button for computing the study (running a simulation as defined by that study and presenting a default plot of the solution).
- Under **Results**, add plot buttons for plotting each plot group defined in the model (**Plot Stress**, for example, for a Stress plot group that plots stresses in a solid mechanics model). Such a button sends the plot to a graphics window in the application. You can also add buttons for exporting data for nodes under **Export**.

Under **Forms**, add a **Show form I** button (  ), for example, to create a button to show any of the existing forms in the application.








Under **GUI Commands>File Commands** you can add the following buttons for file-related operations:

- **Save Application** button (  ), to save the application with the current name.
- **Save Application As** button (  ), to save the application (with extension .mph) with a new name.
- **Save Application on Server** button (  ), to save the application on the server with the current name.
- **Save Application on Server As** button (  ), to save the application (with extension .mph) on the server with a new name.

If the application is run on COMSOL Server, the **Save Application on Server** and **Save Application on Server As** commands save the current application as a new application in the COMSOL Server Application Library.



- **Open File** button (  ), to open a file in an application.
- **Save File As** button (  ), to make it possible to choose and download a file to the client application.
- **Exit Application** button (  ), to exit from the application.


Under **GUI Commands>Graphics Commands**, you can add these additional buttons for graphics-related properties and operations:

- **Zoom Extents** button (  ), to make it possible to zoom in to the extents of the plotted object or geometry.
- **Reset Current View** button (  ), to make it possible to reset the current view in the graphics window to the view that you get when starting the application.
- **Scene Light** button (  ), to add scene light to a graphics window.
- **Transparency** button (  ), to make 3D graphics objects transparent.
- **Print** button (  ), to print the contents of a graphics object.
- **Select All** button (  ), to select all objects in the graphics window.
- **Clear Selection** button (  ), to clear the selection for all objects in the graphics window.

For all graphics commands, add the name of the Graphics object that you want to apply the command to as an argument.

Under **GUI Commands>Model Commands**, you can add these buttons for model-related operations:

- **Clear All Solutions** button (  ), to clear all solutions in an application.
- **Clear All Meshes** button (  ), to clear all meshes in an application.




Double-click or click the **Add Selected**  button to add the button to the form and the list under **Selected**.

Double-click or click the **Remove Selected**  button to remove an item from the **Selected** list.

5 Click **OK** to exit the wizard and generate the form's components that you have defined.

You can add model-dependent data and properties, as well as application-specific properties that can be modified from a running application, by using the data access functionality in the Application Builder. This is in addition to the predefined model data and model properties that you access in the **New Form** wizard. If data access is active, applicable data and properties in the **Settings** windows for many nodes under a **Component** node, as well as the **Settings** windows for some forms, form objects, and menu and toolbar items in the application, get a green check box to the left of the settings. Select this check box to select it as a data source and add the corresponding property to the tree under **Available** on the **Inputs/outputs** page in the **New Form** wizard and in the **Source** sections of applicable form objects. Depending on the type of data, it can become a text label and an input field, check box, or combo box.

To activate the data access:

- On the **Home** toolbar, click **Model Builder** () , click the **Developer** tab if needed, and then click **Data Access** () .
- In the **Source** section for applicable form objects, click the **Switch to Model Builder and Activate Data Access** button () to move directly to the **Model Builder** window with the data access activated.






## The Form Toolbar

---

The **Form** contextual ribbon toolbar provides access to functionality for creating and editing forms and form objects. This toolbar is available when a form window is active. This section briefly describes the buttons on the **Form** toolbar.



### THE MAIN SECTION

This section contains the following buttons for moving to various windows and to create new forms and methods:

- The **Model Builder** button () , to switch from the Application Builder to the Model Builder windows and the standard COMSOL Desktop.
- The **New Form** button () , to create a new form using the **New Form** wizard. See [Working with a Form and Using the New Form Wizard](#).
- The **New Method** buttons () , to create a new **Global Method** or **Form Method** node and open its code in a new editor tab. See [The Method Nodes and Method Editor Windows](#).
- The **Settings** button () , to open or close the **Settings** window.
- The **Preview** button () , to show or hide the **Preview** window for a live preview of the forms and methods in the application. In the **Preview** window, you can scroll to get a preview of all forms and methods in the application. This can be useful, for example, if you are working on a method that interacts with a form. To show a preview of a form in the **Preview** window, select a form or method node in the **Application Builder** window.



### THE FORMS SECTION

This section contains the following buttons for creating a new form and inserting form objects into the current form:

- The **New Form** button () , to create a new form using the **New Form** wizard. See [Working with a Form and Using the New Form Wizard](#).
- The **Insert Object** button () , to open a menu where you can choose from all types of form objects and add them to the current form. See [The Form Objects](#).



## THE LAYOUT SECTION

In this section, you can switch between the sketch layout and the grid layout.

- Click the **Grid** button (  ) to switch to the grid mode. See [The Grid Mode](#).
- Click the **Sketch** button (  ) to switch to the sketch mode. See [The Sketch Mode](#).










## THE SKETCH SECTION

This section contains buttons for tools available in the sketch mode. In the grid mode, these buttons are disabled.

- Click the **Show Grid Lines** button (  ) to add or remove an overlaid grid in the form window.
- Click the **Arrange** button (  ) to open a menu with tools to align the selected form objects.





## THE GRID SECTION

This section contains buttons for tools available in the grid mode. In the sketch mode, these buttons are disabled. In addition, most buttons are disabled unless you have selected cells in the grid that make the corresponding action possible. The buttons include:

- The **Row Settings** button (  ), to choose from a menu of settings for the row size for the selected rows. See [Row and Column Settings](#).
- The **Column Settings** button (  ), to choose from a menu of settings for the row size for the selected columns. See [Row and Column Settings](#).
- The **Insert** button (  ), to insert a row or a column next to the selected rows or columns. You can also open a menu to choose if you want to insert a row above or below, or if you want to insert a column to the left or to the right. See [Inserting and Removing Rows and Columns](#).
- The **Remove** button (  ), to remove the current row or column. See [Inserting and Removing Rows and Columns](#).
- The **Align** button (  ), to open a menu with tools to align and fill the contents in rows and columns. See [Aligning Form Objects](#)
- The **Merge Cell** button (  ), to merge the selected cells. See [Merging and Splitting Cells](#).
- The **Split Cell** button (  ), to split the selected cell. See [Merging and Splitting Cells](#).
- The **Extract Subform** button (  ), to extract the selected cells from the current form and insert them in a new form. See [Extracting a Subform](#).
- The **Rows & Columns** button (  ), to open a dialog box where you can specify the number of rows and columns in the grid. See [Changing the Grid Size](#).

## THE TEST SECTION




This section contains the following buttons for testing the application:

- The **Test Application** button (  ), to launch the application in a separate window so that you can test it. See [Testing the Application](#).
- The **Apply Changes** button (  ), to compile and apply code changes to the running application (so-called hot code swap). See [Applying Changes to a Running Application](#).
- The **Preview Form** button (  ), to open a window that contains a preview of the form. See [Previewing and Testing the Form](#).
- The **Test in Web Browser** button (  ), to test run the application in a web browser. See [Testing the Application](#).



## THE VIEW SECTION

The **View** section contains the following buttons for rearranging the views in the Application Builder desktop window:

- The **Tile** () and **Move To** () buttons, to rearrange the windows in the Application Builder.
- The **Reset Desktop** button (), to reset the desktop layout to the default state.



### *The Form Window Layout Modes*

---

There are two different layout modes that you can use when working in the **Form** windows (editor windows):

- The *sketch mode*, in which you can place the form objects freely and organize them easily by dragging them. The sketch mode is useful as a starting point to quickly make a layout that is roughly what you want. However, what you see in the sketch mode will not be exactly what you get when running the application.
- The *grid mode*, in which you can control in detail what the final layout of the form will look like.

The sketch mode computes the row and column layout from absolute coordinates, heights, and widths of each contained form object. The grid mode lets you specify the row and column size of the contained form objects.

You select the mode by clicking the **Sketch** button () or the **Grid** button () in the **Layout** group of the **Form** ribbon toolbar. See the following sections for more information about each mode.

You can specify which layout mode to use as the default on the **Forms** page in the **Preferences** dialog box.

## OPENING OTHER FORM WINDOWS FROM A FORM WINDOW

Some form objects are forms themselves, such as the grid panel, or contain forms, such as the card stack. To open a new **Form** window for these forms, Alt-click them.

## DRAGGING AND COPYING FORM OBJECTS

In both layout modes, you can drag-and-drop form objects to move them. If you instead want to make a copy of a form object and drag-and-drop it in a new location, drag the original form object, press the Ctrl key, and then release the copy in the desired location. You can cancel the drag operation by pressing Esc or by dropping the form objects outside of the form editor.

### *The Sketch Mode*


---

The sketch mode uses a canvas on which you can place form objects freely. However, the COMSOL Multiphysics software places the objects in a grid when creating the user interface for the running application. This grid does not allow for completely free positioning, which means that the layout you create in sketch mode will only be approximately what you get when running the application. Use the sketch mode as a starting point to quickly create a layout that is roughly what you want, then switch to the grid mode for the fine tuning. This creates a grid based on where you have placed the form objects in the sketch mode.



## ADDING AND SELECTING FORM OBJECTS

To add a form object, select the object from the palette that opens when you click the **Insert Object** button on the **Form** ribbon toolbar (see [The Form Objects](#) for details about the available form objects). The form object of the selected type is then added to the canvas and you can drag it to where you want it.


When you add a form object, it becomes selected. To select another form object, click it. You can select multiple form objects using Ctrl-click or by clicking and dragging a box to surround some form objects. Press the Shift key while dragging to add the surrounded form objects to the current selection.







To delete form objects, select them and then press Delete or click the **Delete** button (  ) on the Quick Access Toolbar.

### SHOWING GRID LINES AND SNAPPING TO THE GRID

To display the grid lines for the grid that divides the form and to which you can snap the form objects, click the **Show Grid Lines** button (  ) in the **Sketch** section of the **Form** ribbon toolbar. When you interactively resize a form object using its handles, the form object snaps to the grid. Click the **Show Grid Lines** button (  ) again to turn off the display of the grid lines and the snap-to-grid behavior. You control the grid and snap settings in the **Sketch Grid** section of the **Form** node's **Settings** window.

### ARRANGING FORM OBJECTS

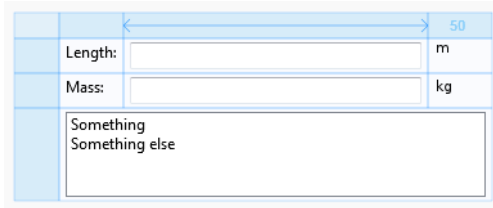
When dragging a form object, lines appear to help you align the objects relative to each other. There is also an **Arrange** menu (  ) in the **Sketch** section of the **Form** ribbon toolbar with the following tools to align the form objects:

- **Align Left** (  ), to left align selected form objects to the leftmost object's position.
- **Align Center** (  ), to horizontally center selected form objects.
- **Align Right** (  ), to right align selected form objects to the rightmost object's position.
- **Align Top** (  ), to top align selected form objects to the topmost object's position.
- **Align Middle** (  ), to vertically center selected form objects.
- **Align Bottom** (  ), to bottom align selected form objects to the bottommost object's position.

### *The Grid Mode*

---

COMSOL Multiphysics uses a grid when creating the user interface for the running application. In the grid mode, you can edit the grid and place form objects in the positions you want within the grid to obtain the desired layout.



*Figure 3-1: An example of a grid with form objects. The cells in the grid can span over multiple rows and columns. For example, the cell in the last row with the list box spans over all three columns.*

### ADDING AND SELECTING FORM OBJECTS

To add a new form object, first select an empty cell in the grid. Then select the object from the palette that opens when you click the **Insert Object** button on the **Form** ribbon toolbar (see [The Form Objects](#) for details about the available form objects). The new object is inserted in the cell, or if no cell is selected, in the first empty cell. If there is no empty cell, the software adds a new row and places the form object in the first cell in that row.

When you add a form object, it becomes selected. To select another form object or empty cell, click it. You can select multiple form objects and cells using Ctrl-click. You can also drag a box to select multiple form objects and cells. Dragging a box to select cells works when you start dragging either in an empty cell or outside the grid.

### COPYING, DUPLICATING, CUTTING, AND DELETING FORM OBJECTS

For all form objects that you have added, you can right-click any form object's cell, a group of cells, or an entire column or row to open a context menu with the following actions:

- **Cut** (✂), to cut (remove) the form object from the form and store it so you can paste it. You can also use Ctrl+X.
- **Copy** (📄), to make a copy of the form objects. You can also use Ctrl+C.
- **Duplicate** (📄➡), to make a duplicate of the form objects, which appears directly in the form.
- **Delete**, to delete the form objects. You can also press Delete to remove form objects.
- **Settings** (🗑), to open the **Settings** window for a form object, or a **Settings** window for common properties if you have selected multiple objects.
- **Help** (❓), to open the **Help** window and display information about the form or form object, when applicable.

For form objects that support methods that run on a data change, for example, the context menu also contains **Create Local Method** to create a local method, or **Edit Local Method** or **Edit Method** to open an existing local method or other method in an editor window.

If you right-click on the **Form** window's canvas, the context menu contains these additional actions:

- **Preview Form** (▶), to launch an application window to preview the contents of the form.
- **Paste Button** (📄), for example, to paste a form object (a button in this case) that you have copied or cut. You can also use Ctrl+V.

Using the arrow keys, you can traverse the cells in the grid, selecting one object at a time. You can also select multiple cells by pressing the Shift key while traversing with the arrow keys.

### MOVING A FORM OBJECT

You can move form objects to another cell by dragging and dropping them. Click a cell to select the form object in the cell. Then drag the object and drop it on the cell where you want to place it. If the cell where you drop the form object is occupied by another object, that object will be moved to the cell vacated by the object you dragged. This makes it easy to change places between two form objects.

### ROW AND COLUMN SETTINGS

You can control the size of the rows and columns in three ways. To choose which way to control it for a row or column, first select the row or column by clicking the header to the left of or above the grid. The image below shows a selected column.

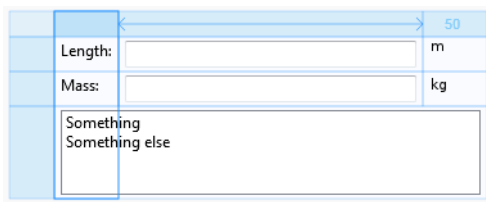








Figure 3-2: Selecting a column using the grid layout mode.

To change how to control the width of the column, use the **Column Settings** menu (📏) in the **Form** ribbon toolbar. The same menu is also available when you select a column in the form window and as a **Column** submenu when you select one or several cells and then right-click. The following options are available:

- **Fit Column** (📏➡), to make the width equal to the widest object in the column.


- **Grow Column** () , to change the column width as the width of the grid's container changes. The container can be the window where the form is used or a column in a parent form. Letting one or more columns grow is a good way of making a user interface that adjusts its size to fit the available space.
- **Fixed Column** () , to set the column width to an exact number. The width can be adjusted by dragging the column separator in the column header. When using this option, make sure that everything in the column fits the available space. Also consider that all objects will not have the same size on all clients on which the application might run.





Similarly, to change how to control the height of a row, use the **Row Settings** menu () in the **Form** ribbon toolbar. The same menu is also available when you select a row in the form window and as a **Row** submenu when you select one or several cells and then right-click. The following options are available:


- **Fit Row** () , to make the height equal to the highest object in the row.
- **Grow Row** () , to change the row height as the height of the grid's container changes. The container can be the window where the form is used or a row in a parent form.
- **Fixed Row** () , to set the row height to an exact number. The height can be adjusted by dragging the row separator in the row header. When using this option, make sure that everything in the row fits the available space. Also consider that all objects will not have the same size on all clients on which the application might run.



The headers indicate the row and column setting: When you use **Grow Row**, the header has an arrow symbol like the middle column in [Figure 3-2](#). When you use **Fixed Row**, the size is shown in the header (shown in the right column in [Figure 3-2](#)).

#### INSERTING AND REMOVING ROWS AND COLUMNS

When you have selected a row or column, you can use the **Insert** menu () , in the **Grid** section of the **Form** ribbon toolbar or on the context menu when right-clicking a selected row or column, to insert rows or columns. Choose from the following options:


- **Insert Above** () , to insert a new row above the selected row.
- **Insert Below** () , to insert a new row below the selected row.
- **Insert Left** () , to insert a new column to the left of the selected column.
- **Insert Right** () , to insert a new column to the right of the selected column.





You can also use the **Remove** menu () to remove rows or columns:





- **Remove Row** () , to remove the selected rows from the form.
- **Remove Column** () , to remove the selected columns from the form.

The applicable remove command is also available on the context menu when right-clicking a selected row or column.

#### ALIGNING FORM OBJECTS

You can adjust the alignment of form objects within the cells using the tools on the **Align** menu () in the **Grid** section of the **Form** ribbon toolbar. The same menu is also available as an **Align** submenu when you select one or several cells and then right-click. To change the alignment, first select the cells with the form objects that you want to align, then select one of the following alignment options:

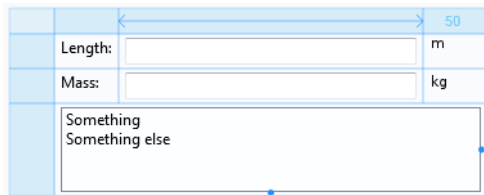
- **Fill Horizontally** () , to make the form objects occupy all of the available space in the column. The assigned width of the form object is not considered when filling; instead, the column gives the size.
- **Align Left** () , to left align selected form objects to the left ends of their cells.
- **Align Center** () , to horizontally center selected form objects in their cells.
- **Align Right** () , to right align selected form objects to the right ends of their cells.

- **Fill Vertically** (  ), to make the form objects occupy all of the available space in the row. The assigned height of the form object is not considered when filling; instead, the row gives the size.
- **Align Top** (  ), to top align selected form objects to the top ends of their cells.
- **Align Middle** (  ), to vertically center selected form objects in their cells.
- **Align Bottom** (  ), to bottom align selected form objects to the bottom ends of their cells.

### ADJUSTING THE SIZE OF A FORM OBJECT

Some form objects can be resized by dragging, but not all objects can be resized. One example is labels, where the size is given by the label text.


To resize a form object, first select it by clicking the cell with the object. The image below shows a selected list box.




*Figure 3-3: A selected list box can be dragged to change its width and height.*


Once selected, you can adjust the size by dragging the handles in the right and bottom sides of the list box. If the object has the horizontal alignment set to fill when dragging horizontally, the alignment is changed to left. Similarly, when dragging vertically, the alignment is changed to top. A blue guideline appears to make it easier to align an object with other objects in the same form.

### MERGING AND SPLITTING CELLS

To make a form object span multiple rows or columns, you can merge cells. To merge cells, select them and then click the **Merge Cells** button (  ) in the **Grid** section of the **Form** ribbon toolbar. When merging, there must be at most one form object in the cells to be merged because the new cell can only contain one form object.

The inverse of merging cells is to split a cell that spans more than one row or column. To split a cell, select it and then click **Split Cell** (  ) in the **Grid** section of the **Form** ribbon toolbar. If there was any form object in the original cell, it is placed in the top left of the new cells.

### EXTRACTING A SUBFORM

If you want to extract a rectangular area (a subgrid) from a form to a new form, select the part of the grid that you want to extract, right-click, and choose **Extract Subform** (  ). You can also click the **Extract Subform** button in the **Grid** section of the **Form** ribbon toolbar. The extracted subgrid, with the form objects in that part of the original form, appears in a new form, and the corresponding form window opens. The cells in the original form change into a **Form** reference object referring to the new form.

### RESIZING THE GRID AND THE FORM

If there is at least one row or column that is expandable, the whole grid in a form can be resized. As the growing row or column adjusts its size to the available space given by its container, you can use the resizing to see how the form looks when the available space changes.

Clicking in the top-left corner of the grid selects the whole grid and shows handles that indicate that you can resize the grid. The image below shows a selected grid with a drag handle in the right border. You can grab and resize the border anywhere along the borderline.

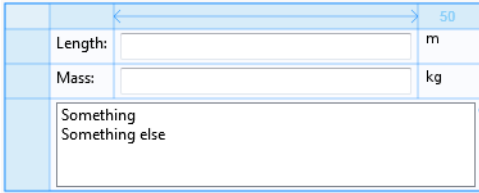



Figure 3-4: A grid handle to the right shows that you can resize the grid.


You can resize a form without first selecting it for resizable forms that use the grid mode and have rows or columns that can grow. Resize the form by dragging its right or bottom border (a resize cursor appears when resizing is available). The size to which you resize the form in the Form Editor is used as its initial size if the form's **Size** setting is set to Automatic.

### CHANGING THE GRID SIZE

To change the number of rows and columns in the grid, click the **Rows & Columns** button (  ) in the **Grid** section of the **Form** ribbon toolbar. In the **Rows & Columns** dialog box that opens, specify the number of rows and columns in the **Rows** and **Columns** fields, and then click **OK**. If the new size is smaller than the current grid size, rows and columns are removed starting from the bottom-right corner of the grid. Also, the resizing of the grid removes any form objects that do not fit in the new grid.

### Previewing and Testing the Form

---

To see what the form looks like and test if input fields and other form objects behave as specified, you can launch the form in a separate window. To do so, click the **Preview Form** (  ) button on the **Form** ribbon toolbar.

### Running Local Methods in Form Objects

---

For some form objects, such as buttons, you can add commands to run when the user clicks the button. These commands can be a number of predefined commands (such as plotting). For other form objects, such as input fields, you can add local methods for events, which occur when the form object gains or loses focus. When you have added a method to run in a form object, the display of the form object in the form window shows the method connection.

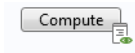


Figure 3-5: A Compute button with an indication that it runs a method when the user clicks it.

If the method has a compilation error, the method connection indicator shows a red cross to indicate that it will not be possible to test the button until the compilation error has been corrected.




Figure 3-6: A Compute button with an indication that the method has a compilation error.

Local methods in form objects can be converted to global methods or form methods.

# The Form Objects

## *Overview of the Form Objects*

---

When you click the **Insert Object** button () on the **Form** ribbon toolbar, a palette (gallery) with the following types of form objects appears.

- A set of **Input** form objects:
  - An [Input Field](#) object for user inputs
  - A [Button](#) object
  - A [Toggle Button](#) object
  - A [Check Box](#) object
  - A [Combo Box](#) object
- A set of **Labels**:
  - A [Text Label](#) object
  - A [Unit](#) object
  - An [Equation](#) object
  - A [Line](#) object
- A set of **Display** objects:
  - A [Data Display](#) object
  - A [Graphics](#) object
  - A [Web Page](#) object
  - An [Image](#) object
  - A [Video](#) object
  - A [Progress Bar](#) object
  - A [Log](#) object
  - A [Message Log](#) object
  - A [Results Table](#) object
- A set of **Subforms**:
  - A [Form](#) object
  - A [Form Collection](#) object
  - A [Card Stack](#) object
  - A [Card](#) object, which you can add and edit from a Card Stack object
- A set of **Composite** form objects:
  - A [File Import](#) object
  - An [Information Card Stack](#) object for displaying application information cards
  - An [Array Input](#) object
  - A [Radio Button](#) object
  - A [Selection Input](#) object

- A set of **Miscellaneous** form objects:
  - A **Text** object
  - A **List Box** object
  - A **Table** object
  - A **Slider** object
  - A **Hyperlink** object
  - A **Toolbar** object
  - A **Spacer** object

See the following sections for details about each type of form object. Select the form object that you want to insert. A form object of that type is then inserted into the selected form’s editor window. For a selected form object, the **Settings** window displays its properties, which you can edit to control the object’s appearance and behavior. If you select multiple form objects, the **Settings** window displays common properties only.

### *Input Field*

---

The **Input Field** () is a form object for entering single-lined text.


Enter the name of the input field object in the **Name** field.

By default, the **Editable** check box is selected, so that users of the application can change the value in the input field. Clear the **Editable** check box to display the initial value as a read-only value.





In the **Tooltip** field, enter text that will appear as a tooltip when the user hovers the pointer over the input field. The input field objects also have a built-in tooltip that shows the entered text if it does not fit into the input field, or any error or warning.

In addition, the **Settings** window contains the following sections.


#### **SOURCE**

In this section, you define the data source for the input field. The section contains a tree with a filtered view of the tree in the **Application Builder** window. The nodes either represent some sort of data or have children that do. For an input field, parameters and variables in the COMSOL Multiphysics model and scalar variables that you define under **Declarations** are available as the data source. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button () in the **Source** section header, and then selecting a node in the **Model Builder** with data you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Click to select the check box to include the data as an available source node for input fields.



If the **Editable** check box is cleared so that the field is read-only, you can choose to use one of the following information nodes as the source. They are found under the main **Model** node and each **Study** node.

- The **Expected Computation Time** node () under **Model>Information** (): The expected computation time is a value that you can enter in the **Expected** field in the **Root** node’s **Settings** window.
- The **Last Computation Time** node () under **Model>Information**: This node shows the last measured computation time for the last computed study.
- The **Last Computation Time** node () under each **Model>Study>Information**: This node shows the last measured computation time for the study.

When you start an application for the first time, the last measured times are reset, displaying **Not available yet**.

When you select a node that represents data, you enable the **Use as Source** toolbar button () below the tree. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as



the selected source. You can also click the **Create** button (  ) in the **Source** section header to create a new variable declaration and use it as the source. A **Create and Use Variable** dialog box opens, so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. Click the **Edit Node** toolbar button (  ) below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.

After selecting a node as the source, the node appears as the selected source under **Selected source**.





If you try to use the same data source in several form objects, you may encounter some strange side effects. The value for the source may not be what you expect. You may also experience serious errors if the initial value of one form object is invalid for one of the other form objects.


From the **Initial value** list, select **From data source** to link this input field to the data source defined in the **Source** section and use the values specified by that source. **From data source** is the default setting for new input fields. Select **Custom value** to enter the initial value for the input field in the **Value** field. This value overwrites any value you specified for that string.

#### DATA VALIDATION

Use the settings in this section to validate user inputs with respect to units and values.

From the **Unit dimension check** list, choose one of the following options:

- **None** (the default): No check is done of the unit or value.
- **Compatible with physical quantity**: A check is done to make sure that the input's unit is compatible with the physical quantity that you specify using the **Physical quantity** list underneath the **Unit dimension check** list. Click the **Select Quantity** button (  ) to open the **Physical Quantity** dialog box to browse to find a physical quantity to use. You can also type a search string in the text field at the top of the dialog box and then click the **Filter** button (  ) to filter the list of physical quantities. For example, type `potential` and click the **Filter** button to only list physical quantities that represent some kind of potential.
- If the value or expression that the user enters is incompatible, the value or expression is highlighted in orange as a warning and a tooltip displays to describe the unit mismatch. If there is a unit mismatch, the application uses the numerical value of the entered value or expression and adds the default unit as specified (for example, `9 [kg]` is converted to `9 [m]` if the expected physical quantity is a length).
- **Compatible with unit expression**: A check is done to make sure that the input's unit is compatible with the unit expression that you specify in the **Unit expression** field underneath the **Unit dimension check** list. The app user gets the same type of information and handling as in the previous case if there is a unit mismatch.
- **Append unit to number**: The app user can type in a number without appending a unit using the unit syntax with square brackets. The application appends the unit expression that you specify in the **Unit expression** field underneath the **Unit dimension check** list.
- **Append unit from unit set**: The input field appends a unit from a **Unit Set** node added under **Declarations** (see [The Unit Set Node](#)). You specify the unit set to use from the **Unit set** list and the unit to use from **Unit list**, which lists all defined properties and their units from the select unit tests. There is also a **No unit** option.

When applicable, click the **Add Unit Label** button (  ) to add a unit label linked to the input field and place it to the right of the input field.

For the options **None**, **Append unit to number**, and **Append unit from unit set**, you can also use a filter to validate the numerical input. Under **Numerical validation**, choose a filter from the **Filter** list:

- **None**, for the **None** option only, where it is the default setting.

- **Double** (the default for the **Append unit to number** option), for checking that the entered value is a floating-point number (double). Select the check boxes for **Minimum** and **Maximum** to specify a lower and upper limit, respectively, in the corresponding text fields. The limits can be numerical values or expressions, including parameters defined in the global **Parameters** node's **Settings** window. The limits dynamically change when the parameter values change. It is possible that, after a change in the minimum or maximum value, the value in an input field becomes temporarily invalid. If the parameter or expression for the minimum or maximum value does not have the same dimension as the unit entered in the **Unit expression** field, the parameter or expression appears in yellow, and a tooltip describes the unit mismatch. If the parameter or expression has the same dimension but another unit than the unit entered in the **Unit expression** field, a unit conversion to that unit occurs.
- **Integer**, for checking that the entered value is an integer. Select the check boxes for **Minimum** and **Maximum** to specify a lower and upper limit (as integer values), respectively, in the corresponding text fields.
- **Regular expression** (for the **None** option only), to use a regular expression for matching the input string and issue a custom error. Specify the expression to check against in the **Regular expression** field and the error message (default: `Invalid input`) in the **Error message** field.

The regular expression consists of a pattern that is matched to the user input. The pattern can contain ordinary characters and special characters. Special characters have a special meaning and do not represent themselves. The special character `\` is used for quoting special characters to have those characters represent themselves; for example, `\\` represents a backslash. When you use `\` on ordinary characters, it makes them have a special meaning. For example, `\t` means a tab character, and `\n` means a newline character. The special character `.` (a dot) stands for any character. `[abc]` (characters in brackets) stands for any of the characters `a`, `b`, or `c`. `[^abc]` stands for any characters except for `a`, `b`, or `c`. `(abc)` (characters in parentheses) represents the sequence of characters `abc`. `*` (an asterisk) means repetition, including zero times. `+` (plus) means repetition, at least one time. For a complete description of regular expressions, see the documentation of the `java.util.regex` class in Java.

Some examples:

- `a.c` means the letter `a` followed by any character followed by the letter `c`.
- `[^ab]c` means any character but `a` and `b`, followed by the letter `c`.
- `a*b+` means any number of the letter `a` (including zero occurrences) followed by at least one `b`.

If you test the application using **Test Application** and a data validation error message occurs, the error message contains extra information about the type of form object and its path to help you locate the origin of the error message. The extra information does not appear when running an application through **Run Application** or COMSOL Server.

## POSITION AND SIZE

This section contains all layout settings for an input field in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the input field using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the Application Builder. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width of the input field. Enter a width (in points) in the **Width** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. The **Height** field is unavailable because the height of the input field is determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the input field using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form

**Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

## **APPEARANCE**

In this section, you can control the appearance of the background and the text in the input field:

From the **Text color** list, select **Inherit** (the default) to inherit the text color from the setting in the **Form** node, or select one of the predefined colors, such as **Black**. Select **Custom** to choose a custom text color from the color palette.

From the **Background color** list, select a color to use as the background in the input field: **White** (the default), **Transparent**, any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

From the **Text alignment** list, select an alignment for the text in the input field: **Left**, **Center**, or **Right**.

The font and the font size for the text use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **Default size** for the font.

Select the **Bold** check box to use a boldface font and the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the input field when users run the application. By default, the input field is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the input field is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.


## **EVENTS**

For certain types of form objects, you can specify a method to run when an event such as data entry occurs. The **On data change** list contains **None** (the default) and any available methods. To add a local method for an event, click the **Create Local Method** button ( **+** ) to the right of the **On data change** list. The selected method in the **On data change** list then changes to **Local method**. An empty `onDataChange` editor window opens, in which you can define the local method. You can also Ctrl+Alt-click on the input field object or right-click it to create a local method or (by choosing **Edit Method** or **Edit Local Method**) to open the method associated with the command. To open the

selected method, click the **Go to Source** button (  ). Click the **Remove Local Method** button (  ) to delete the local method.

For events triggered by data change, the event is triggered after the new data value is stored in the data source.


---

	If you select more than one form object and they all support the <b>On data change</b> event, you can specify a method to, for example, make them inform users that plots and outputs are invalid.
---	--

---



## Button

---

The **Button** (  ) form object represents a push button. When a user presses the button, the program runs a list of commands.

Enter the name of the Button object in the **Name** field.

In the **Text** field, enter the text that appears on the button.

To add an icon (image) to appear on the button, specify an image file to use from the **Icon** list, which includes all images in the **Images** library, or click the **Add Image to Library and Use Here** button (  ) to locate an image to use on the file system. That image then becomes a part of the **Images** library and selected as the icon to use on the button. The icon replaces the text on the button if the button's size is normal. For large buttons, both the text and the icon appear, unless they are empty. To display only the text, select **None** from the **Icon** list. Click the **Export** button (  ) to export the image to the file system (for use in another application, for example).

From the **Style** list, choose **Large** for a large square button, **Small** (the default), or **Flat**. The **Flat** style is the same size as **Large** but with no border and, as the default, a transparent button background. This look appears on Windows and in the web client. On macOS and Linux, the **Flat** and **Large** styles look the same.


The text you enter in the **Tooltip** field becomes the tooltip of the button.

You can also define a shortcut for the button action that you enter in the **Keyboard shortcut** field. To add a keyboard shortcut, make the **Keyboard shortcut** field active, and then type a keyboard shortcut on the keyboard:

You must use a modifier in the keyboard shortcut, not just a plain letter (for example, CTRL+SHIFT+D). The shortcut can include the Ctrl key (CTRL), Alt key (ALT), and Shift key (SHIFT). Note that the Ctrl key is interpreted as Command on OS X. Avoid using the following keys in your shortcut:

- Backspace, as it can be used to clear a shortcut
- Delete, as it can be used to clear a shortcut
- Escape
- Alt on its own (to avoid conflicts with File menu shortcuts)

---

	It is possible to override other keyboard shortcuts, so take care when choosing the shortcut key combinations to use.
---	---

---

In addition, the **Settings** window contains the following sections.

### CHOOSE COMMANDS TO RUN

This section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either support a command or have children that do. When you select a node that supports one or more commands, the corresponding command toolbar buttons become enabled in the toolbar below the tree. You can also right-click a node to get a list of available commands for that particular node. Once you click on a command

with a node selected (or press Enter or double-click to add a command with its default command such as **Run**, **Plot**, or **Set Value**), the command and node appear in the last row of the table below the tree. This table contains all nodes that run. You can delete and move commands using the toolbar below the table.

In the **Model** branch, all nodes that represent some sort of data value, such as a parameter under the **Parameters** node, support the **Set Value** command. When adding a **Set Value** command to the table, the third column, **Arguments**, becomes enabled. In this column, you type the value to set. For data that represents arrays, use curly braces and commas to enter the array elements. For example, enter {1, 2, 3} to set a three-element array with the values 1, 2, and 3. See [The Array 1D String Node](#) for more details on how to enter arrays and matrices. For nodes that represent a file import, such as a **Filename** node under an **Interpolation** function node, an **Import File** command is available. You can also add a **Plot** command for all **View** nodes and plots, providing the name of a Graphics object as the argument. For nodes under **Export**, you can add a **Run** command. In the case of an **Animation** node under **Export**, where the animation is done using a player as the target, you can provide the name of a Graphics object as the argument.


The tree includes a number of branches from the application tree in addition to the **Model** branch:






- The **Forms** branch: **Form** nodes support the commands **Show**, which sets the form as the main form of the application (that is, the content of the application window will be this form), and **Show as Dialog**, which brings up the form as a separate dialog window.
- The **GUI Commands** branch: The commands under this branch are grouped in three subcategories:
  - **File Commands**: These include **Save Application** (to save the application under its current name); **Save Application As** (to open a file browser dialog allowing the user to save the application in a suitable location); **Save Application on Server**; **Save Application on Server As**; **Open File** (to open an application file resource specified using a valid URI path in the **Arguments** column); **Save File As** (similarly, to allow the user to save the file under a name specified in the **Arguments** column); and **Exit Application** (to close the running application). If the application is run on COMSOL Server, the **Save Application on Server** and **Save Application on Server As** commands save the current state as a new application in the COMSOL Server Application Library.
  - **Graphics Commands**: Here you find the commands **Zoom Extents**, **Reset Current View**, **Scene Light**, **Transparency**, **Print**, **Select All**, and **Clear Selection**. For all graphics commands, add the name of the Graphics object that you want to apply the command to as an argument.
  - **Model Commands**: Here you find the commands **Clear All Solutions** and **Clear All Meshes**.

Double-click or right-click any of the nodes above to add a **Run** command.

- The **Declarations** branch: This branch contains any variable declarations you have added under the **Application Builder** window's **Declarations** branch grouped by type. Like parameters, they support the **Set Value** command.
- The **Form Declarations** branch: This branch contains any variable declarations you have added under a **Declarations** branch under the current **Form** node. Like parameters, they support the **Set Value** command.
- The **Methods** branch: **Method** nodes support the **Run** command.
- The **Form Methods** branch: **Method** nodes under the current **Form** node support the **Run** command.
- The **Libraries** branch: Under **Sounds**, you can choose between sound files to play in a command sequence.

When you click one of the buttons underneath the tree, the currently selected command appears in the **Command** column in the table below. There is also an **Icon** column and an **Arguments** column, where you can enter any applicable arguments that the command uses.

Click the **Convert to Method** toolbar button () and choose **Convert to Method**, **Convert to Form Method**, or **Convert to Local Method** to convert the entire list of commands in the table to a global, form, or local method that contains the equivalent code. After this operation, the list of commands only contains a single **Run** operation on the created method. You can also Ctrl+Alt-click on the button object or right-click it to create a local method or (by choosing **Edit Method**, **Edit Methods**, or **Edit Local Method**) to open the methods associated with the commands. When you

select a method under **Command**, or there is exactly one method in the list, you can go to the editor window for that method by clicking the **Go to Method** button () . For information about the **Edit Argument** button () , see [Editing Initial Values and Arguments in Declarations and Command Sequences](#). Use the **Move Up** () , **Move Down** () , and **Delete** () toolbar buttons to organize and remove commands from the list (and also remove the local method, if deleted).

## DIALOG ACTIONS

When you use the button in a dialog window, the button can control the dialog. Select the **Close dialog** check box to close the dialog window when the user clicks the button. All forms and form objects in a dialog window store their values temporarily. This is because the user might want to cancel the dialog window and close it without storing any data to the real data backing the forms and form objects. Select the **Store changes** check box to store the data when the user clicks the button.

## POSITION AND SIZE

This section contains all of the layout settings for a button in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the button using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the Application Builder. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

If you want to specify the width of the button, first select **Manual** from the **Width** list. Then enter the width (in points) in the associated field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0), or choose **Manual** to specify a minimum width in the text field underneath.

If you want to specify the height of the button, first select **Manual** from the **Height** list. Then enter the height (in points) in the associated field. If you have chosen **Fill** from the **Vertical alignment** list, you can instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically this means a minimum size of 0) or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify the absolute position of the button using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

## Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object

- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

## APPEARANCE

In this section, you can control the appearance of the text on the button:

From the **Text color** list, select a color to use for the text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

From the **Background color** list, select a color to use as the background color for the button: Choose **Default** (the platform then controls the background color), **Transparent**, any available color, or **Custom**, to choose the color from a color palette.


The font and the font size for the button label use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **Default size** for the font.

Select the **Bold** check box to use a boldface font and the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the button when users run the application. By default, the button is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the button is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.



## *Toggle Button*

---

The **Toggle Button** () form object represents a toggle button. A toggle button acts as two buttons in one, with one action (command sequence) that runs when a user selects the button and another action that runs when a user deselects the button.

Enter the name of the toggle button object in the **Name** field.

In the **Text** field, enter the text that appears on the button.

To add an icon (image) to appear on the button, specify an image file to use from the **Icon** list, which includes all images in the **Images** library, or click the **Add Image to Library and Use Here** button () to locate an image to use on the file system. That image then becomes a part of the **Images** library and selected as the icon to use on the button. The icon replaces the text on the button if the button's size is normal. For large buttons, both the text and the icon appear, unless they are empty. To display only the text, select **None** from the **Icon** list. Click the **Export** button () to export the image to the file system (for use in another application, for example).

From the **Style** list, choose **Large** for a large square button, **Small** (the default), or **Flat**. The **Flat** style is the same size as **Large** but with no border and, as the default, a transparent button background. This look appears on Windows and in the web client. On macOS and Linux, the **Flat** and **Large** styles look the same.

The text you enter in the **Tooltip** field becomes the tooltip of the button.

You can also define a shortcut for the button action that you enter in the **Keyboard shortcut** field. To add a keyboard shortcut, make the **Keyboard shortcut** field active and then type a keyboard shortcut on the keyboard.

You must use a modifier in the keyboard shortcut, not just a plain letter (for example, CTRL+SHIFT+D). The shortcut can include the Ctrl key (CTRL), Alt key (ALT), and Shift key (SHIFT). Note that the Ctrl key is interpreted as Command on OS X. Avoid using the following keys in your shortcut:

- Backspace, as it can be used to clear a shortcut


- Delete, as it can be used to clear a shortcut
- Escape
- Alt on its own (to avoid conflicts with File menu shortcuts).






It is possible to override other keyboard shortcuts, so take care when choosing the shortcut key combinations to use.

In addition, the **Settings** window contains the following sections.

### SOURCE

In this section you specify the data source for the toggle button. The section contains a tree with a filtered view of the tree in the **Application Builder** window. The nodes either represent some sort of data or have children that do. For a toggle button, the data source can be variables in the COMSOL Multiphysics model and scalar strings or Boolean variables that you define under **Declarations**. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button () in the **Source** section header, which takes you to the Model Builder, and then selecting a node in the **Model Builder** branch whose data you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Click to select the check box to include the data as an available source node for check boxes.

When you select a node that represents data, the **Use as Source** toolbar button () below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. You can also click the **Create New Declaration and Use It as Source** button () in the **Source** section header to create a new variable declaration and use it as the source. A **Create and Use Variable** dialog box opens, so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. Click the **Edit Node** toolbar button () below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.

After selecting a node as the source, the node appears as the selected source under **Selected source**.



If you try to use the same data source in several form objects, you may encounter some strange side effects. The initial value for the source may not be what you expect. You may also experience serious errors if the initial value of one form object is invalid for one of the other form objects.

From the **Initial value** list, select **From data source** to link this check box to a variable defined in the **Source** section above and use the value specified by that data source. Select **Custom value** to specify the initial state from the **Initial state** list: **Selected** (the default) or **Cleared**. The value for a selected toggle button is on and for a deselected toggle button is off.



If a global parameter is the data source of the check box, it is necessary to add two parameters, on and off, with values 1 and 0, respectively. Otherwise, you get an error when the COMSOL Multiphysics software looks for the parameters on and off.

### CHOOSE COMMANDS TO RUN

This section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either support a command or have children that do. When you select a node that supports one or more commands, the corresponding command toolbar buttons become enabled in the toolbar below the tree. You can



also right-click a node to get a list of available commands for that particular node. Once you click on a command with a node selected (or press Enter or double-click to add a command with its default command such as **Run**, **Plot**, or **Set Value**), the command and node appear in the last row of the table below the tree. This table contains all nodes that run. You can delete and move commands using the toolbar below the table.

In the **Model** branch, all nodes that represent some sort of data value, such as a parameter under the **Parameters** node, support the **Set Value** command. When adding a **Set Value** command to the table, the third column, **Arguments**, becomes enabled. In this column, you type the value to set. For data that represents arrays, use curly braces and commas to enter the array elements. For example, enter {1, 2, 3} to set a three-element array with the values 1, 2, and 3. See [The Array 1D String Node](#) for more details on how to enter arrays and matrices. For nodes that represent a file import, such as a **Filename** node under an **Interpolation** function node, an **Import File** command is available. You can also add a **Plot** command for all **View** nodes, providing the name of a Graphics object as the argument.





The tree includes a number of branches from the application tree in addition to the **Model** branch:


- The **Forms** branch: **Form** nodes support the commands **Show**, which sets the form as the main form of the application (that is, the content of the application window will be this form), and **Show as Dialog**, which brings up the form as a separate dialog window.
- The **GUI Commands** branch: The commands under this branch are grouped in three subcategories:
  - **File Commands**: These include **Save Application** (to save the application under its current name); **Save Application As** (to open a file browser dialog allowing the user to save the application in a suitable location); **Save Application on Server**; **Save Application on Server As**; **Open File** (to open an application file resource specified using a valid URI path in the **Arguments** column); **Save File As** (similarly, to allow the user to save the file under a name specified in the **Arguments** column); and **Exit Application** (to close the running application). If the application is run on COMSOL Server, the **Save Application on Server** and **Save Application on Server As** commands save the current state as a new application in the COMSOL Server Application Library.
  - **Graphics Commands**: Here you find the commands **Zoom Extents**, **Reset Current Views**, **Scene Light**, **Transparency**, **Print**, **Select All**, and **Clear Selection**. For all graphics command, add the name of the Graphics object that you want to apply the command to as an argument.
  - **Model Commands**: Here you find the commands **Clear All Solutions** and **Clear All Meshes**.

Double-click or right-click any of the nodes above to add a **Run** command.

- The **Declarations** branch: This branch contains any variable declarations you have added under the **Application Builder** window's **Declarations** branch grouped by type. Like parameters, they support the **Set Value** command.
- The **Form Declarations** branch: This branch contains any variable declarations you have added under a **Declarations** branch under the current **Form** node. Like parameters, they support the **Set Value** command.
- The **Methods** branch: **Method** nodes support the **Run** command.
- The **Form Methods** branch: **Method** nodes under the current **Form** node support the **Run** command.
- The **Libraries** branch: Under **Sounds**, you can choose between sound files to play in a command sequence.

When you click one of the buttons underneath the tree, the currently selected command appears in the **Command** column in the table below. There are also **Icon** and **Arguments** columns, where you can enter any applicable arguments that the command uses.

Click the **Convert to Method** toolbar button () and choose **Convert to Method** or **Convert to Form Method** to convert the entire list of commands in the table to a global or form method that contains the equivalent code. After this operation, the list of commands only contains a single **Run** operation on the created method. When you select a method under **Command**, or there is exactly one method in the list, you can go to the editor window for that method by clicking the **Go to Method** button (). For information about the **Edit Argument** button (), see [Editing Initial Values and Arguments in Declarations and Command Sequences](#). Use the **Move Up** () , **Move Down**

(  ), and **Delete** (  ) toolbar buttons to organize and remove commands from the list (and also remove the local method, if deleted).

## POSITION AND SIZE

This section contains all layout settings for a toggle button in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the toggle button using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode. Then, it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the Application Builder. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

If you want to specify the width of the toggle button, first select **Manual** from the **Width** list. Then enter the width (in points) in the associated field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath.

If you want to specify the height of the toggle button, first select **Manual** from the **Height** list. Then enter the height (in points) in the associated field. If you have chosen **Fill** from the **Vertical alignment** list, you can instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically this means a minimum size of 0) or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify the absolute position of the toggle button using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

## APPEARANCE

In this section, you can control the appearance of the text on the toggle button:

From the **Text color** list, select a color to use for the text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

From the **Background color** list, select a color to use as the background color for the toggle button: Choose **Default** (the platform then controls the background color), **Transparent**, any available color, or **Custom**, to choose the color from a color palette.


The font and the font size for the toggle button label use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **Default size** for the font.

Select the **Bold** check box to use a boldface font and the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the toggle button when users run the application. By default, the toggle button is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the toggle button is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## Check Box

---


The **Check Box** () form object represents a check box, which can be selected or cleared. A selected check box sets the data source value to *on*, and a cleared check box sets the value to *off*.





Enter the name of the check box object in the **Name** field.

Enter the label next to the check box in the **Text** field.

In addition, the **Settings** window contains the following sections:

### SOURCE

In this section you specify the data source for the check box. The section contains a tree with a filtered view of the tree in the **Application Builder** window. The nodes either represent some sort of data or have children that do. For a check box, the data source can be variables in the COMSOL Multiphysics model and scalar strings and Boolean variables that you define under **Declarations**. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button () in the **Source** section header, which takes you to the Model Builder, and then selecting a node in the **Model Builder** branch whose data you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Click to select the check box to include the data as an available source node for check boxes.

When you select a node that represents data, the **Use as Source** toolbar button () below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. You can also click the **Create New Declaration and Use It as Source** button () or the **Create New Form Declaration and Use It as Source** button () in the **Source** section header to create a new global or local (in the form) variable declaration for the check box and use it as the source. A **Create and Use Variable** dialog box opens, so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. Click the **Edit Node** toolbar button () below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.

After selecting a node as the source, the node appears as the selected source under **Selected source**.



If you try to use the same data source in several form objects, you may encounter some strange side effects. The initial value for the source may not be what you expect. You may also experience serious errors if the initial value of one form object is invalid for one of the other form objects.

---

From the **Initial value** list, select **From data source** to link this check box to a variable defined in the **Source** section above and use the value specified by that data source. Select **Custom value** to specify the initial state from the **Initial state** list: **Selected** (the default) or **Cleared**. The value for a selected check box is **on** and for a cleared check box **off**.



If a global parameter is the data source of the check box, it is necessary to add two parameters, **on** and **off**, with values 1 and 0, respectively. Otherwise, you get an error when the COMSOL Multiphysics software looks for the parameters **on** and **off**.

## POSITION AND SIZE

This section contains all layout settings for a check box in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the check box using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode. Then it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the Application Builder. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

The **Width** and **Height** fields are unavailable because the dimensions of the check box are determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the check box using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

## Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

## APPEARANCE

In this section, you can control the appearance of the background and the text on the check box label.

From the **Text color** list, select a color to use for the text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.




From the **Background color** list, select a color to use as the background in the check box: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

The font and the font size for the button label use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **Default size** for the font.

You can also select the **Bold** check box to use a boldface font, the **Italic** check box to use italics (an italic font), or the **Underline** check box to underline the text.


Under **State**, you can control the initial state of the check box when users run the application. By default, the check box is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the check box is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## EVENTS


You can add a code method that the application runs when the user changes the value of the check box. The event is triggered after the new data value is stored in the data source. The **On data change** list contains **None** (the default) and any available methods. To add a local method for this event, click the **Create Local Method** button (  ). The selected method in the **On data change** list then changes to **Local method**. An empty **onDataChange** editor window opens, where you can define the local method. You can also Ctrl+Alt-click on the check box or right-click it to create a local method or (by choosing **Edit Method** or **Edit Local Method**) to open the method associated with the command. To open the selected method, click the **Go to Source** button (  ). Click the **Remove Local Method** button (  ) to delete the local method.





## Combo Box

---

The **Combo Box** (  ) form object represents a *combo box*, which can either work as a combination of a *drop-down list box* and an editable text field or as an uneditable drop-down list box. The **Settings** window contains the following sections.

### SOURCE

In this section, you define the data source for a combo box. The section contains a tree with a filtered view of the tree in the **Application Builder** window. The nodes either represent some sort of data or have children that do. For a combo box, you can use variables from the model or those defined under **Declarations**, including **Unit Set** nodes. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button (  ) in the **Source** section header, which takes you to the Model Builder, and then selecting a node in the **Model Builder** branch whose data you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Click to select the check box to include that data as an available source node for combo boxes.

When you select a node that represents data, the **Use as Source** toolbar button (  ) below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. You can also click the **Create New Declaration and Use It as Source** button (  ) or the **Create New Form Declaration and Use It as Source** button (  ) in the **Source** section header to create a new global or local (in the form) variable declaration for the combo box and use it as the source. A **Create and Use Variable** dialog box opens, so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. Click the **Edit Node** toolbar button (  ) below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.

After selecting a node as the source, the node appears as the selected source under **Selected source**.



If you try to use the same data source in several form objects, you may encounter some strange side effects. The initial value for the source may not be what you expect. You may also experience serious errors if the initial value of one form object is invalid for one of the other form objects.

In the **Initial value** list, choose a method to define an initial value for the combo box. The options are **First allowed value**; **From data source** (the default, to use the value specified by the selected data source); and **Custom value**. For the **Custom value** option, a **Value** list shows the allowed values currently present for the form object and depends on the selected available choice lists and their values. If the data source is a setting from the embedded model that has a list of allowed values, those values are also included in the **Value** list. If a selected initial value becomes invalid because it has been removed from the choice list, for instance, it is kept as an initial value with the text **Invalid initial value** followed by the value.

### CHOICE LIST

In the **Selected** list, add **Choice List** nodes that contribute allowed values to the combo box. If the selected data source is a list with a set of allowed values, only a subset of those values can appear in the allowed values of the combo box. All other values in the selected choice lists are ignored. Available **Choice List** nodes appear under **Available**. Click the **Add Selected** → button to add the selected **Choice List** node to the list under **Selected**. Click the **Remove Selected** ← button to remove a selected **Choice List** node from the list under **Selected**. You can also double-click a **Choice List** node to move it from **Available** to **Selected** and the other way around. Click the **Add New Choice List** button ( + ) to open a **Choice List** window where you can define a new choice list. Add the allowed values in the **Value** column and their corresponding names in the **Display name** column. Click **OK** to add the new choice list as a **Choice List** node ( ◊ ) under **Declarations** and directly under **Selected** in the list under **Choice List**.

If you select a property that has a list of allowed values as the data source in the **Source** section, that property becomes a node initially placed in the **Selected** list. You can move it to the **Available** list, thereby clearing the list of allowed values. You can move it back again or add a custom choice list with values that also belong to the list of values for the property. If the property list and a choice list node are both in the **Selected** list, they will be merged. Identical values pick the description from the first item in the list under **Selected**, so in this way, you can rename one of the items in the property list. If you decide to switch the source to another property in the embedded model that also has a list of allowed values, the previous property list node is removed from both the **Available** and the **Selected** lists, and the new node is added to the **Selected** list.

Select the **Allow other values** check box to get a combo box where you can type arbitrary values. Such combo boxes can accept any value and are not restricted to the values defined by the choice lists.

### POSITION AND SIZE

This section contains all of the layout settings for a combo box in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the combo box using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the Application

Builder. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width of the combo box. Enter a width (in points) in the **Width** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. The **Height** field is unavailable because the height of the combo box is determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the combo box using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, to use no margin around the form object.
- **From parent form** (the default), to use the margins set for the parent form.
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields.

### **APPEARANCE**

In this section, you can control the appearance of the combo box.


From the **Text color** list, select a color to use for the text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

The font and the font size for the text use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **Default size** for the font.

You can also select the **Bold** check box to use a boldface font and the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the combo box when users run the application. By default, the combo box is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the combo box is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.


### **EVENTS**

You can add a code method that the application runs when the data in the combo box changes. The event is triggered after the new data value is stored in the data source. The **On data change** list contains **None** (the default) and any available methods. To add a local method for this event, click the **Create Local Method** button ( **+** ) or right-click the combo box object. The selected method in the **On data change** list then changes to **Local method** and an editor window for the local method opens, where you can define its contents. You can also Ctrl+Alt-click on the combo box or right-click it to create a local method or (by choosing **Edit Method** or **Edit Local Method**) to open the method associated with the command. Click the **Go to Source** button (  ) to open an editor window for the selected method. Click the **Remove Local Method** button ( **X** ) to delete the local method.



## Text Label

---

The **Text Label** () form object represents static text that you can use to display some information in a form window.

Enter the name of the text label object in the **Name** field.

If you want to display multiline text, select the **Multiline text** check box. By default, the label displays a single line of text.

Type the text to display into the **Text** field.

In addition, the **Settings** window contains the following sections.

### POSITION AND SIZE

This section contains all layout settings for a text label in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the text label using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the Application Builder. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

The **Width** and **Height** fields are unavailable because the dimensions of the text are determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the text label using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, to use no margin around the form object.
- **From parent form** (the default), to use the margins set for the parent form.
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields.

### APPEARANCE

In this section, you can control the appearance of the background and the text.

From the **Text color** list, select a color to use for the text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.



From the **Background color** list, select a color to use as the background in the text label: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.


The font and the font size for the text use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **Default size** for the font.

You can also select the **Bold** check box to use a boldface font, the **Italic** check box to use italics (an italic font), or the **Underline** check box to underline the text.

Under **State**, you can control the initial state of the text label when users run the application. By default, the text label is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the text label is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## *Unit*

---

The **Unit** (  ) form object is a predefined label for displaying a unit. When linking to this node from an **Input Field**, it also acts as a unit check for that field (see [Input Field](#)).

Enter the name of the unit object in the **Name** field.

From the **Label** list, select an option for how to display the unit: **From reference**, to display the unit as defined in the referenced source, such as input field form object. You select the source from the tree in the **Source for Label** section. Alternatively, select **Fixed** and type a unit expression into the **Unit expression** field that appears.

By default, the unit display uses Unicode rendering. If the unit display does not look as expected, consider using LaTeX rendering instead by selecting the **LaTeX markup** check box. Then, the unit display does not rely on the selected font. It can also be a matter of preference. The look and alignment of the unit display changes slightly depending on whether this check box is selected or not.

In addition, the **Settings** window contains the following sections.

### **POSITION AND SIZE**

This section contains all of the layout settings for a unit in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the unit using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the Application Builder. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

The **Width** and **Height** fields are unavailable because the dimensions of the unit label are determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the unit label using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, to use no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### APPEARANCE

In this section, you can control the appearance of the unit label's text and background.

From the **Text color** list, select a color to use for the unit label's text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette that opens.

From the **Background color** list, select a color to use as the background in the unit display: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

The font and the font size for the text use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **Default size** for the font.

You can also select the **Bold** check box to use a boldface font and the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the unit when users run the application. By default, the unit is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the unit is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

### Equation


---

The **Equation** ( $\Delta u$ ) form object can display LaTeX-rendered symbols, such as equations.



Enter the name of the equation object in the **Name** field.

The **Settings** window contains the following sections.

#### EQUATION

Enter the equation or mathematical expression to show in the **Enter equation in LaTeX syntax** field. The program displays a preview of the rendered LaTeX syntax after leaving the text field. Otherwise, preview it manually by clicking the **Refresh Equation Preview** button (  ). The default expression is  $-\nabla \cdot (k \nabla u)$ , which displays as:

$$-\nabla \cdot (k \nabla u)$$

In the **Equation** section header, use the **Insert Expression** (  ) and **Replace Expression** (  ) buttons to select predefined LaTeX expressions for vector operations, Greek letters, big operators, sets, logic, and text formatting that you can insert into the equation or add as the start of a new equation.

## POSITION AND SIZE

This section contains all layout settings for an equation in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the equation using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the Application Builder. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

The **Width** and **Height** fields are unavailable because the dimensions of the equation are determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the equation using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

## Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object. This setting can be useful when displaying an equation inside of a sentence to avoid excessive whitespace margins.
- **From parent form** (the default), to use the margins set for the parent form.
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields.

## APPEARANCE

In this section, you can control the appearance of the equation.

From the **Text color** list, select **Inherit** (the default) to use the setting from the form it is located in. You can also select one of the predefined colors or select **Custom** to choose a custom text color from the color palette.

The default font size for the equation text is **Default size**, which uses the font size from the **Forms** node. If needed, choose or enter a font size (in points) in the **Font size** combo box.

Under **State**, you can control the initial state of the equation when users run the application. By default, the equation is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state of the equation hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## Line

---

The **Line** ( — ) form object places a horizontal or vertical line in a form to separate groups of form objects.

Enter the name of the line object in the **Name** field.

Choose the orientation of the divider line in the **Orientation** list: **Horizontal** (the default) or **Vertical**. For the **Horizontal** divider, you can also add text that appears within the horizontal line. To add such text, select the **Include divider text** check box and enter text in the associated **Text** field.

In addition, the **Settings** window contains the following sections.

### POSITION AND SIZE

This section contains all layout settings for a line in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the line using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the Application Builder. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width of a horizontal line, unless you have chosen **Fill** from the **Horizontal alignment** list. Enter a width (in points) in the **Width** field. The **Height** field is unavailable because the height of the horizontal line is determined by the software. For a vertical line, you can instead specify the height, unless you have chosen **Fill** from the **Vertical alignment** list. Enter a height (in points) in the **Height** field. The **Width** field is then unavailable because the width of the vertical line is determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the line using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, to use no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### APPEARANCE

In this section, you can control the appearance of the line's divider text.

From the **Text color** list, select a color to use for the divider text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

The font and the font size for the divider text use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **Default size** for the font.

You can also select the **Bold** check box to use a boldface font or the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the line when users run the application. By default, the line is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the line is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## *Data Display*

---

The **Data Display** () is a form object that represents a display of some property as numerical data.


Enter the name of the data display object in the **Name** field.

By default, the data display uses Unicode rendering. If the data display does not look as expected or the expression to display contains LaTeX syntax, consider using LaTeX rendering instead by selecting the **LaTeX markup** check box. Then the data display does not rely on the selected font. It can also be a matter of preference. The look and alignment of the data display changes slightly depending on whether this check box is selected or not.





In the **Tooltip** field, enter text that will appear as a tooltip when the user hovers the pointer over the data display.

In addition, the **Settings** window contains the following sections.





### **SOURCE**

In this section, you specify the source for the data to display. The section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either represent some sort of data or have children that do. For a data display, the data source can be variables in the COMSOL Multiphysics model, scalar strings, array and matrix settings from model data such as material properties, and numerical (double and integer) variables and arrays or matrices that you define under **Declarations**. The data display can show arrays or matrices with LaTeX syntax. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button () in the **Source** section header, which takes you to the Model Builder, and then selecting a node in the **Model Builder** branch whose data you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Click to select the check box to include that data as an available source node for data display objects.

In addition, you can choose to use one of the following information nodes, which you find under the main **Model** node and under each **Study** node, as the source:

- The **Expected Computation Time** node () under **Model>Information** (): The expected computation time is a value that the application developer can enter in the **Expected** field in the **Root** node's **Settings** window.
- The **Last Computation Time** node () under **Model>Information**: This node shows the last measured computation time for the last computed study.
- The **Last Computation Time** node () under each **Model>Study>Information**: This node shows the last measured computation time for the study.

When you start an application for the first time, the last measured times are reset, displaying **Not available yet**.

When you select a node that represents data, the **Use as Source** toolbar button () below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. You can also click the **Create New Declaration and Use It as Source** button () or the **Create New Form Declaration and Use It as Source** button () in the **Source** section header to create a new global or local (in the form) variable declaration for the data display and use it as the source. A **Create and Use Variable** dialog box opens, so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. Click the **Edit Node** toolbar button () below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.

After selecting a node as the source, the node appears as the selected source under **Selected source**.

### NUMBER FORMAT

In the **Precision** field, enter the number of significant digits to display in the output (default: 4). Choose the notation to use for the data from the **Notation** list. The options are **Automatic** (the default), **Scientific**, and **Decimal**. If you use **Automatic**, the program switches to scientific notation when the output is about 3 orders of magnitude larger than or smaller than 1. For the **Automatic** and **Scientific** options, you can also choose the format of the exponent for scientific notation from the **Exponent** list. The choices are **Power of 10** (the default) and **E-notation**. Select the **Display all significant digits** check box (available for the Scientific and Decimal options) to always add trailing zeros to match the specified precision. Clear the **Append unit to number** check box if you do not want a unit appended after the number representing the data displayed.

### POSITION AND SIZE

This section contains all layout settings for a data display in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the check box using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the Application Builder. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

The **Width** and **Height** fields are unavailable because the dimensions of the numerical data display are determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the data display using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object. This setting can be useful when displaying dynamical data inside a sentence to avoid excessive whitespace margins.
- **From parent form** (the default), to use the margins set for the parent form.
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields.

### APPEARANCE

In this section, you can control the appearance of the text and background in the data display.

From the **Text color** list, select a color to use for the text in the data display: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

From the **Background color** list, select a color to use as the background in the data display: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.


The font and the font size for the text use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **Default size** for the font.

You can also select the **Bold** check box to use a boldface font or the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the data display when users run the application. By default, the data display is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the data display is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

### Graphics

---

The **Graphics** () form object can plot the same thing as you can plot in the **Graphics** window in the COMSOL Desktop. Typically, you choose a plot group under the **Results** branch of the model to plot. You can also plot a geometry or mesh, or dynamically let the plot type change. In addition, you can include the plot-while-solving functionality (including probe plots) in runnable applications. To do so, enable it in the study step settings and then ensure that the plot group you are plotting is set as the source to the Graphics form object.

The Graphics object includes a plot toolbar with buttons for zooming, transparency, lighting, printing, and more. You can extend or replace the standard plot toolbar with custom buttons.


Enter the name of the Graphics object in the **Name** field.




The **Zoom to extents on first plot** check box is selected by default. This setting makes the first plot that appears in the graphics canvas zoom to its extents when the applications starts, if the graphics canvas is empty initially or when something is plotted the first time. Clear this check box to disable the zoom to extents action.

Select the **Data picking** check box to activate data picking in the Graphics window. The data picking provides functionality for processing user interaction with the Graphics form object.

The **Settings** window contains the following sections.




## SOURCE FOR INITIAL GRAPHICS CONTENT



In this section, you specify a node that represents a plot that becomes the initial contents of the graphics window. The section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes represent some sort of plot or have children that do. The filtered view also includes Explicit selection nodes. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button () in the **Source for Initial Graphics Content** section header, which takes you to the Model Builder, and then selecting a node in the **Model Builder** branch whose data you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Click to select the check box to include the data as an available source node for graphics objects.

When you select a node that represents data, the **Use as source** button () below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. After selecting a node as the source, the node appears as the selected source under **Selected source**. You can select from all plot groups and player animations under the **Results** branch and all geometry and mesh nodes. You can also select **Explicit** selection nodes, which makes it possible for users to select geometric entities to update that selection directly in the graphics. Alternatively, use a **Selection Input** object in the application for activating selections of geometric entities. Click the **Edit Node** toolbar button () below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder. Click the **Clear Source** toolbar button () to remove a source that you have selected.



You can also select a string under the **Declarations** branch. The value of this string controls the plot to show, where the value represents a path to one of the nodes that you can select. For example, if the value is `pg1`, the plot shows the plot group with this tag. If the value is `/GeomList/geom1`, it plots the geometry with the tag `geom1`, and `/MeshList/mesh1` plots the mesh with the tag `mesh1`. You can also use the value `/Results/ResultFeatureList/pg1` for the plot group, but that syntax is rather cumbersome.

## TARGET FOR DATA PICKING

When you have selected the **Data picking** check box above, select a valid data picking target from the list in this section. The target could be a declared double floating point variable, which then is used in a form to display some quantity at the clicked location in the graphics window. The list also contains probes defined in the model and **Graphics Data** nodes added under **Declarations** (see [The Graphics Data Node](#)). For a probe, you can, for example, use a slider to let the user of the app determine the depth along a line in a 3D geometry. Using a **Graphics Data** node, you can output both the clicked location and some evaluated result at that location. You can also click the **Create New Declaration and Use It as Source** button () or the **Create New Form Declaration and Use It as Source** button () in the **Target for Data Picking** section header to create a new global or local (in the form) variable declaration for the data picking and use it as the selected target. A **Create and Use Declaration** dialog box opens so that you can select the data type of the target (if applicable), its name, and its initial value (if applicable). You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button ()

Click the **Use as Target** button () to make the selected node the target for the data picking. That node then appears under **Selected target**. Click the **Edit Node** button () to move to the **Settings** window for the selected node in the tree.

## APPEARANCE

To add an icon to the upper-right corner (a logotype, for example), specify an image file to use from the **Icon** list, which includes all images in the **Images** library, or click the **Add Image to Library and Use Here** button () to locate an image to use on the file system. This image becomes a part of the **Images** library and is selected as the icon in this **Graphics** object. If you do not want to use an icon, select **None** from the **Icon** list. Click the **Export** button () to export the image to the file system (for use in another application, for example).



This section also contains settings for the background color. Under **Background for 2D plots**, use the **Color** list to select the background color for 2D plots and 1D graphs. The 3D background supports a gradient from top to bottom. Under **Background for 3D plots**, choose the top and bottom color in the **Top color** and **Bottom color** lists.

For all plot background colors, **Use default** is the default setting. This setting is a white background for 2D plots and 1D graphs, and a light blue gradient for 3D plots. In addition to a set of predefined colors, you can also choose **Transparent** and **RGBA**. For RGBA, enter values between 0 and 255 for the red; green; blue; and alpha (opacity, where 0 means full transparency, and 255 is fully opaque) components of the color as comma-separated values in the **RGBA** field.



The graphics canvas shows a placeholder in the form windows and when you run a **Preview Form**, so any changes to these settings are not reflected. You have to run a **Test Application** to see the effect of changes to the graphics settings.

Under **State**, you can control the initial state of the graphics object when users run the application. By default, the graphics object is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the graphics object is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

#### TOOLBAR




In this section, you can add items to a plot toolbar and activate the following table options.





From the **Position** list, choose where you want to position the toolbar relative to the table: **Below**, **Above** (the default), **Left**, or **Right**.

Clear the **Include standard toolbar buttons** check box if you do not want to display the standard plot toolbar buttons for zooming, changing the view, showing legends, adding scene light and transparency, and for creating image snapshots and printing the plot. By default, the toolbar items that you add are placed before the standard toolbar. Clear the **Place standard toolbar before custom items** check box to place the custom items after the standard toolbar items.

You can choose from two icon sizes: From the **Icon size** list, choose **Small** (the default) or **Large**.

In the table below, you can add one or more buttons to form a plot toolbar:

- Click the **Add Item** button () to open the **Edit Custom Toolbar Item** dialog box (see [The Edit Custom Toolbar Item Dialog Box](#)) and create and add a custom toolbar button to the plot toolbar.
- Click the **Add Toggle Item** button () to open the **Edit Custom Toolbar Item** dialog box (see [The Edit Custom Toolbar Item Dialog Box](#)) and create and add a custom toolbar toggle button to the plot toolbar.
- Click the **Add Separator** button () to add a separator between groups of buttons in the toolbar.

Select a button in the table and click the **Edit** button () if you want to change the appearance or behavior of a custom toolbar button in the **Edit Custom Toolbar Item** dialog box. Click the **Move Up** and **Move Down** buttons ( and ) to move and rearrange the toolbar button order. Click the **Delete** button () to delete the selected button.

The table contains a row for each added item, showing its name, icon, text, and tooltip in the **Name**, **Icon**, **Text**, and **Tooltip** columns, respectively.

#### POSITION AND SIZE

This section contains all layout settings for a graphics object in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the check box using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width and height of the graphics object. Enter a width (in points) in the **Width** field and a height (in points) in the **Height** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically this means a minimum size of 0), or choose **Manual** to specify a minimum width in the text field underneath. If you have chosen **Fill** from the **Vertical alignment** list, you can instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically this means a minimum size of 0), or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify the form object's absolute position using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.


---

### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### *Web Page*

The **Web Page** () form object can display the contents of a web page as part of the user interface. You can, for example, use it to provide information about the application. Enter the name of the web page object in the **Name** field.

The **Settings** window contains the following sections.

#### **SOURCE**

You can specify the page source in three different ways from the **Source** list:

- Use the default option, **Page**, to enter HTML code in a text area below the list, enclosed by the `<html>` and `</html>` start and stop tags.

- Use the **URL** option to point to an online web page, which you specify in the **Page URL** field.
- Use the **File** option to point to a local file resource containing HTML code. Type the name of the file in the **File** field or click **Browse** to locate the file on the local file system.



The file reference points to a file resource on the local system that may not be present when running the application. When saving, all file references are embedded into the MPH-file, so you do not have to distribute them along with the application. A running application always looks for embedded resources first.

- Under **Browser preview**, you can see a preview of what the display of the web page will look like.



To display web pages, COMSOL Multiphysics and COMSOL Client use the Internet Explorer version installed on your computer.

## APPEARANCE

Under **Appearance**, you can control the initial state of the web page object when users run the application. By default, the web page object is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the web page object is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## POSITION AND SIZE

This section contains all layout settings for a web page in the grid of the parent form.

You can control the horizontal and vertical alignment of the web page object using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, or **Right**.
- From the **Vertical alignment** list, choose **Middle**, **Top**, or **Bottom**.

In the grid mode, you can also choose **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object is useful in the sketch mode too. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width and height of the web page object. Enter a width (in points) in the **Width** field and a height (in points) in the **Height** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0), or choose **Manual** to specify a minimum width in the text field underneath. If you have chosen **Fill** from the **Vertical alignment** list, you can instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically this means a minimum size of 0), or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify the absolute position of the web page using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---


### *Cell Margin*



Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### *Image*

---

The **Image** () form object makes it possible to add an image inside of a form. You can also add an image form object by pasting an image that you have copied to the clipboard (from an image program or from another form, for example) into a form using Ctrl+V.

Specify a name for the image in the **Name** field. The name must be unique among the form objects in the form. Specify an image file to use from the **Image** list, which includes all images in the **Images** library, or click the **Add Image to Library and Use Here** button () to locate an image to use on the file system. The image then becomes a part of the **Images** library and selected as the image in this **Image** object. If you do not want to display an image, select **None** from the **Image** list. Click the **Export** button () to export the image to the file system (for use in another application, for example).



The file reference points to a file resource on the local system that may not be present when running the application. When saving, all file references are embedded into the MPH-file, so you do not have to distribute them along with the application. A running application always looks for embedded resources first.

---

In addition, the **Settings** window contains the following sections.

#### **POSITION AND SIZE**

This section contains all layout settings for an image in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the image using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form

windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

The **Width** and **Height** fields are unavailable because the dimensions of the image are determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the image using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:


- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields



### APPEARANCE

Under **Appearance**, you can control the initial state of the image when users run the application. By default, the image is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the image is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

### Video

---

The **Video** () form object makes it possible to add a video inside of a form. The Video form object embeds a video file in a form. The video is a web page created in HTML5. The supported video file formats are .mp4 (MP4 format), .ogv (OGV format), and .webm (WebM format) files. However, not all video file formats are supported on all platforms. When running an application by connecting to COMSOL Server from a web browser, which formats are supported depend on the web browser and may vary with different versions of the same web browser. Of these formats, MP4 works with most web browsers, whereas OGV and WebM work in Firefox and Google Chrome (as web clients).

Specify a name for the image in the **Name** field. The name must be unique among the form objects in the form. Specify a video file to use from the **Video** list, or click the **Add File to Library and Use Here** button () to locate a video file to use on the file system. That video then becomes a part of the **Files** library and selected as the video in the **Video** object. To not show a video, select **None** from the **Video** list. Click the **Export** button () to export the video file to the file system (for use in another application, for example).



The file reference points to a file resource on the local system that may not be present when running the application. When saving, all file references are embedded into the MPH-file, so you do not have to distribute them along with the application. A running application always looks for embedded resources first.

---

In the form editor, a placeholder image appears where the video will appear when running the application.

The **Show video controls** check box is selected by default. The user of the application can then control the video using controls to pause and play the video and to mute and unmute the sound. Clear this check box to show the video without any controls.

Select the **Start automatically** check box if you want the video to start directly when the user runs the application.

Select the **Repeat** check box if you want the video to keep running repeatedly.

Select the **Initially muted** check box if you want the video to be initially muted (no sound).



To show a video when running an application with COMSOL Multiphysics and with the COMSOL Client, the Internet Explorer version installed on your computer is used as a software component for displaying the video object.

---

In addition, the **Settings** window contains the following sections.

#### **POSITION AND SIZE**

This section contains all layout settings for a video in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the video using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

The **Width** and **Height** fields are unavailable because the dimensions of the image are determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the video using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

#### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:


- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

## APPEARANCE

Under **Appearance**, you can control the initial state of the video when users run the application. By default, the image is visible. Clear the **Visible** check box if you want to make the initial state of the video hidden. You can then make it visible using a method. Objects that are hidden become visible when selected in the form editor.

### *Progress Bar*

---

The **Progress Bar** () form object adds a progress bar to a form in an application. The progress bar can show progress based on a value that describes some progress in a running application using a method that updates the progress bar. You control the update and display of the progress in the progress bar by referring to the name that you enter in the **Name** field. For example, add the following two lines of code in a method editor window:

```
startProgress("progressbar1");  
setProgress("progressbar1", 12);
```

The first line starts progress for the progress bar object with the name `progressbar1`. The second line updates its progress to 12%. You can find a full listing of the available commands for controlling the progress under **User interface>Progress** in the **Language Elements** window, accessible in the **Code** section of the **Method** ribbon toolbar. You can also base the progress on the built-in model progress (the main progress in COMSOL Multiphysics) by selecting the **Include model progress** check box. Optionally, it is then possible to use two levels of progress. Select **Two** from the **Progress levels** list (default: **One**) to use two levels of progress (represented using two progress bars).

Select the **Cancel button** check box to add a **Cancel** button underneath the progress bars to make it possible to cancel some progress in the application.

If the **Cancel button** check box is selected, you can select or clear the **Close dialog when canceled** check box. If that check box is selected, dialog forms are closed when a user clicks the **Cancel** button.



You can also add a progress bar for the progress information from built-in actions, such as the solvers, in the **Settings** window for the **Main Window** node.

A third option is to add a progress dialog for the application's own progress using available methods such as `startProgress(progressBarname)`, `setProgress(int workDone)`, and `closeProgress()`. See the full list of Application Builder methods in the *Introduction to the Application Builder* document.

---

The **Settings** window contains the following sections.

## APPEARANCE

Under **Appearance**, you can control the initial state of the progress bar when users run the application. By default, the progress bar is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state of the progress bar hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## POSITION AND SIZE

This section contains all layout settings for a progress bar in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the progress bar using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width of the progress bar. Enter a width (in points) in the **Width** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. The **Height** field is unavailable because the height of the progress bar is determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the progress bar using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---


### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### *Log*

---

The **Log** () form object adds a log window that displays messages from the COMSOL Multiphysics software, such as from the solver operations. Enter the name of the log object in the **Name** field.

The **Include standard log toolbar** check box is selected by default, which includes the toolbar in the **Log** window that you see on the COMSOL Desktop. Clear the check box to remove it.

In addition, the **Settings** window has the following section.

#### **POSITION AND SIZE**

This section contains all layout settings for a log in the grid of the parent form.



In the grid mode, you can control the horizontal and vertical alignment of the log using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width and height of the log window. To adjust the width, enter a width (in points) in the **Width** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. To adjust the height, enter a height (in points) in the **Height** field. If you have chosen **Fill** from the **Vertical alignment** list, you can instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify the absolute position of the log window using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows updates these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### **APPEARANCE**

In this section, you can control the appearance of the text and background in the log.

From the **Text color** list, select a color to use for the text in the log: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

From the **Background color** list, select a color to use as the background in the log: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

The font and the font size for the text use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **Default size** for the font.

You can also select the **Bold** check box to use a boldface font or the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the log when users run the application. By default, the log is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the log is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## Message Log

---

The **Message Log** (☒) form object adds a **Messages** window where you can add messages to inform the user about operations that the application carries out using the built-in `message(String message)` method. Enter the name of the message log object in the **Name** field.

The **Include standard message log toolbar** check box is selected by default, which includes the toolbar in the **Messages** window that you see on the COMSOL Desktop. Clear the check box to remove it. The **Show COMSOL messages** check box is selected by default to include messages from the COMSOL Multiphysics software in the **Messages** window. Clear the check box to only include messages from the application itself. Select the **Add timestamps to messages** check box to start each message with a timestamp that provides the current date and time.

In addition, the **Settings** window has the following section.

### POSITION AND SIZE

This section contains all layout settings for a message log in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the message log using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width and height of the **Messages** window. To adjust the width, enter a width (in points) in the **Width** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. To adjust the height, enter a height (in points) in the **Height** field. If you have chosen **Fill** from the **Vertical alignment** list, you can instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify the form object's absolute position using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### APPEARANCE

In this section, you can control the appearance of the text and background in the message log.

From the **Text color** list, select a color to use for the text in the message log: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

From the **Background color** list, select a color to use as the background in the message log: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.


The font and the font size for the text use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **Default size** for the font.

You can also select the **Bold** check box to use a boldface font or the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the message log when users run the application. By default, the message log is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the message log is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

### Results Table


---


The **Results Table** () form object adds a results table that can display numerical results in a table. You typically specify the source of the results data as a Global Evaluation in a COMSOL Multiphysics model. When you add an Evaluate Global Evaluation command for a button, for example, you can provide the name of the results table object that you want to update with the new results. You can also specify a Boolean value to specify if the results table should be cleared (true; the default) or not cleared (false). Enter the result table object's name in the **Name** field.

The **Include standard results table toolbar** check box is selected by default, which includes the toolbar in a **Table** window that you see on the COMSOL Desktop. Clear the check box to remove it. The toolbar includes buttons for precision and notation of the table data, a button for copying the table and headers to the clipboard, and a button for exporting the table data to a file. The file types that you can save table data to include text files (.txt); CSV files (.csv); data files (.dat); and, if the license includes LiveLink™ for Excel®, Microsoft Excel® files (.xlsx).

In addition, the **Settings** window has the following sections.

### SOURCE

In this section, you specify the source of the results data. The section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either represent some sort of output or have children that do. When you select a node that represents an output, the **Use as source** toolbar button () below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. After selecting a node as the source, the node appears as the selected source underneath **Selected source**. All **Evaluation Group** nodes as well as all nodes in the **Derived Values** and

**Tables** branches under **Model>Results** and are valid output nodes. Click the **Edit Node** toolbar button (  ) below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.

## POSITION AND SIZE

This section contains all layout settings for a results table in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the results table using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width and height of the results table. To adjust the width, enter a width (in points) in the **Width** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. To adjust the height, enter a height (in points) in the **Height** field. If you have chosen **Fill** from the **Vertical alignment** list, you can instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify the absolute position of the results table using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form window update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

## Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

## APPEARANCE

In this section, you can control the appearance of the text in the results table:

From the **Text color** list, select a color to use for the text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.


The font and the font size for the results table use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default font size is **Default size**.

You can also select the **Bold** check box to use a boldface font or the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the results table when users run the application. By default, the results table is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the results table is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## Form

---

The **Form** () object is a subform that includes a link to a **Form** so that you can reuse a user interface in several places. Enter the name of the form object in the **Name** field.

The **Form** list holds the reference to the form that this subform links to. Choose an existing form object other than the form that you add the form object to, or choose **None** if you do not want to link to any form.

Select the **Add border** check box to surround the subform with a border. If cleared, the subform's border is not visible. If you add a border, it includes the title of the form that the subform links to.

In addition, the **Settings** window has the following section.

### POSITION AND SIZE

This section contains all layout settings for a form object in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the form object using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width and height of the form. By default, they are determined automatically. To adjust the width, select **Manual** from the **Width** list and then enter a width (in points) in the associated field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. To adjust the height, select **Manual** from the **Height** list and then enter a height (in points) in the associated field. If you have chosen **Fill** from the **Vertical alignment** list, you can instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify the absolute position of the form object using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.


---

### Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### Form Collection

The **Form Collection** () form object is a collection of forms that displays as panes in a tabbed pane form object, where each pane shows the form content. Each form can use a different set of layout options. You can also display them with a list for selecting the panes or as separate sections. A tabbed pane needs a data source to hold the currently selected pane and to enable pane switching from methods. Therefore, you also need to choose a data source and initial value similar to how you do so for combo box form objects. Enter the name of the form collection object in the **Name** field.

From the **Type** list, choose the type of layout for the form collection:

- The **Tabs** layout (the default) displays the forms using tabbed panes.
- The **List** layout displays a list to the left of the form panes, where you can select the form to display.
- The **Sections** layout displays each form in a separate section.
- The **Tiled or tabbed** layout has two different looks with the same forms and a Boolean source controlling which one of the modes (the tiled mode or the tabbed mode) is shown. The tabbed mode is identical to a form collection with the type set to **Tabs**. The tiled mode displays all of the forms simultaneously in a grid.




The **onLoad** and **onClose** event methods are active when opening and closing a form and when you switch tabs in a form collection. The **onClose** method runs when switching to a tab using the form, and the **onLoad** method runs when switching from a tag using the form. The events only run for form collections with the layout types **Tabs** or **List**. For **Sections**, no events run.

In addition, the **Settings** window contains the following sections.

#### TILED OR TABBED






This section is only available if you have chosen **Tiled or tabbed** from the **Type** list above.


From the tree view, choose a Boolean variable to use as the source for the switch between a tiled look and a tabbed look. Click the **Use as Source** button () or right-click and choose **Use as Source** to make a selected Boolean variable the source for switching the look of the form collection.

You can specify some settings for the tiled mode under by **Tiled mode settings**. By default, the **Add borders in tiled mode** check box is selected to add borders around the form objects. From the **Tiling strategy** list, choose **Columns first** (the default) or **Rows first** to control the order of the tiled form objects. You can specify the number of columns for the tiled mode in the **Number of columns** field (default: 2 columns). The form objects in the two modes are synchronized.


#### ACTIVE PANE SELECTOR

In this section, you define the active pane selector that controls the visible pane for the form collection when the **Type** list is set to **Tabs** or **List**. The section contains a tree with a filtered view of the tree in the **Application Builder** window. The nodes either represent some sort of data or have children that do. For a form collection, string variables that you define under **Declarations** are available as the active pane selector. When you select a node that

represents the selector, the **Use as Source** toolbar button () below the tree becomes enabled. You can also right-click the node and choose **Use as Source**. In addition, you can click the **Create New Declaration and Use It as Source** button () or the **Create New Form Declaration and Use It as Source** button () in the **Active Pane Selector** section header to create a new global or local (in the form) variable declaration for the form collection and use it as the source. A **Create and Use Declaration** dialog box opens so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button (). Click the **Edit Node** toolbar button () below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.




After selecting a node as the source, the node appears as the selected source under **Selected source**. A data source as the active pane selector is not necessary, so you can click the **Clear Source** toolbar button () under the source tree. You only need to select a source if you want to control the visible pane in a way other than clicking on the tab to show it.

---

	The data source is only applicable to form collections of the tabs and list types. For form collections that appear as sections, the data source setting has no effect and the source is not applicable.
---	--

---

## PANES

In the **Use selected forms as panes** list, add form objects from the list under **Available** to the list under **Selected** using the  button, where each form represents a pane in the order they appear in the list. From the **Default pane** list, select one of the selected form objects to make it the default pane. When you have selected a data source, the **Default pane** setting also initializes the data source, overriding any default specified in the data declaration. The allowed values for a data source connected to a form collection are the names of the forms, such as `form1` and `form2`. To change the order in which the forms under **Selected** are displayed, use the **Move Up** () and **Move Down** () buttons.

## POSITION AND SIZE

This section contains all layout settings for a form collection in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the form collection using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width and height of the form collection. By default, they are determined automatically. To adjust the width, select **Manual** from the **Width** list and then enter a width (in points) in the associated field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. To adjust the height, select **Manual** from the **Height** list and then enter a height (in points) in the associated field. If you have chosen **Fill** from the **Vertical alignment** list, you can



instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify the absolute position of the form collection using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### **APPEARANCE**

In this section, you can control the appearance of the text in the form collection.

From the **Text color** list, select a color to use for the text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.


The font and the font size for the results table use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default font size is **Default size**.

You can also select the **Bold** check box to use a boldface font or the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the form collection when users run the application. By default, the form collection is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the form collection is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

### *Card Stack*

---

The **Card Stack** () is another special type of form object that only contains cards. A card stack can flip between cards in a stack to show one card at a time. For example, you can display a different image or text depending on an event or results in the application. You associate a card stack with a data source that controls which card to show. Each card specifies a value that it compares against the data source of the card stack. The card stack shows the first card with the matching value. If no cards match, nothing shows. There are two types of cards: local cards and cards that are references to an existing form object.


Enter the name of the card stack object in the **Name** field.






The **Settings** window contains the following sections.

#### **ACTIVE CARD SELECTOR**

In this section, you specify the data source for the active card selector. The section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either represent some sort of data




or have children that do. For a card stack, you typically only see the available parameters under **Parameters**, variables under **Variables**, and the data nodes defined under the **Declarations** branch. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button () in the **Active Card Selector** section header, which takes you to the Model Builder, and then selecting a node in the **Model Builder** branch whose data you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Click to select the check box to include the data as an available source node for card stacks.

When you select a node that represents the source for the active card selector, the **Use as Source** toolbar button () below the tree becomes enabled. You can also right-click the node and choose **Use as Source**. You can also click the **Create New Declaration and Use It as Source** button () or the **Create New Form Declaration and Use It as Source** button () in the **Active Card Selector** section header to create a new global or local (in the form) variable declaration for the card stack and use it as the source. A **Create and Use Declaration** dialog box opens so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button (). Click the **Edit Node** toolbar button () below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.

After selecting a node as the source, the node appears as the selected source under **Selected source**.

In the form, the card stack object displays the card with an activating value that matches the default value of the data source of the card stack.


---

	If you try to use the same data source in several form objects, you may encounter some strange side effects. The initial value for the source may not be what you expect. You may also experience serious errors if the initial value of one form object is invalid for one of the other form objects.
--	--


---


## CARDS


The table in this section contains the cards (in the **Card** column) and their associated activating values (in the **Activating value** column). The stack decides which card to display through the activating values, which you type into this section. The values are checked against the value of the source. For all cards, you can enter their activating values in the **Activating value** column. For local cards, you can also edit the name of the card in the **Card** column.

Click the **Add Card** button () to add a card to the table. An **Add Card** dialog box appears, where you can specify the new card.

From the **Card type** list, choose **Local** (the default) to create a local card for the card stack, or choose **Existing form** to use an existing form as a card. For a local card, enter a card name in the **Name** field. For a form, choose one of the existing forms from the **Form** list. If desired, enter a unique value to act as an activating value in the **Activating value** field.

Click the **Delete** button () (or right-click the card entry in the table) to remove a selected card.

Click the **Edit** button () to edit the individual card. You can also right-click a card entry in the table and select **Edit** or right-click the card stack object in the form window and select **Edit card 1**, for example.

Click the **Duplicate** button () to duplicate a card in the card stack. It is also possible to right-click in the Card Stack object in the form window and select **Duplicate card 2**, for example, to duplicate a card in the stack.

## APPEARANCE

Under **Appearance**, you can control the initial state of the card stack when users run the application. By default, the card stack is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the card stack is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor,

the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## POSITION AND SIZE

This section contains all layout settings for a card stack in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the card stack using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width and height of the card stack. By default, they are determined automatically. To adjust the width, select **Manual** from the **Width** list and then enter a width (in points) in the associated field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. To adjust the height, select **Manual** from the **Height** list and then enter a height (in points) in the associated field. If you have chosen **Fill** from the **Vertical alignment** list, you can instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify the absolute position of the card stack using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---



### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### *Card*

---

The **Card** () is a special form object that is a child to a card stack. When you edit a local card object, it appears in a new form window where you add form objects (such as images or text) that show up on the card. If you edit a card that uses an existing form, that form window opens. The card has a condition. When the condition is true, the card stack shows that card. A true condition stops the search for cards to show, so all other cards with a true condition do not show. For a local card, enter the name of the card object in the **Name** field. To move to the card stack that the card belongs to, click the **Go to Card Stack** button (.

The settings window for a local card contains the following sections.

### CARD ACTIVATION

Enter the value to activate the card in the **Activating value** field. The card becomes visible when this value matches the value of the data source selected in the parent **Card Stack** node (where you can also edit the activating value; this value becomes the default in the **Activating value** field).

### MARGINS

In this section, you can adjust the card's **Horizontal** and **Vertical** margins if desired (default: 20 pixels).

### SKETCH GRID



The **Sketch Grid** section is only available when you have selected the sketch mode for the form.

---

In this section, you find settings for the grid that you can display in the sketch mode (see [Showing Grid Lines and Snapping to the Grid](#)) and for the snapping of form objects to that grid.

Specify the grid size by entering values in the **Column width** (default: 100 pixels) and **Row height** (default: 20 pixels) fields.

Select the **Align grid to margin** check box to make the grid lines align with the left and top margins.

The **Snap zone** slider controls how exact you need to be when resizing a form object to make it snap to the grid. By default, the snap zone is set to its maximum value so that the object quickly resizes to snap to the grid. Move the slider from **Large** to **Small** to make the snap zone smaller, if desired.

Select the **Snap only to grid** check box to make the resizing of form objects snap only to the grid and not to the borders of other form objects, for example.

### GRID LAYOUT FOR CONTAINED FORM OBJECTS



The **Grid Layout for Contained Form Objects** section is only available when you have selected the grid mode for the form.

---

There are two tables in this section: one for the columns and one for the rows in the grid. In the **Column** and **Row** columns, you find the column and row numbers, respectively, each starting at 1 from the left top. You can control how each row and column fills up the space in the form. Each table has a **Width** (columns) or **Height** (rows) column with lists that contain the following options: **Fit** (the default), **Grow**, and **Fixed**.

The **Grow** option makes it possible for the column or row in the grid to expand, taking up space in the form when the user increases the size of the form. No columns or rows with the settings **Fit** or **Fixed** grow. The **Fixed** option specifies that the grid layout has a certain width or height for its column or row, specified in the tables' third **Size** column. For the other options, the third column is ignored. The added width or height in pixels appears in the column or row header.



From the **Inherit columns** list, select a form object from which to inherit column settings. The default is **None**; that is, the column settings are not inherited.

### APPEARANCE

In this section, you can control the color of the text and the background color and image for the card.

From the **Text color** list, select a color to use for the text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.


From the **Background color** list, select a color to use as the background in the card: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

From the **Background image** list, choose a background image if you want to use such an image in the card. The default is **None**; that is, no background image is used. To add an image to the image library and use it as a background image, click the **Add Image to Library and Use Here** button (  ). Click the **Export** button (  ) to save the background image to a PNG file.

Under **State**, you can control the initial state of the card when users run the application. By default, the card is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the card is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.


## *File Import*




---

The **File Import** (  ) is a special form object for browsing and choosing files to import that the application can use for some purpose (providing data input, for example). You can also add files to a file library in the Application Builder (see [The Libraries Branch](#) for more information). Enter the name of the file import object in the **Name** field.

Enter a text to appear on the button for browsing in the **Button text** field.

Enter a title for the file import dialog in the **Dialog title** field. The dialog title is also the tooltip of the button.




In the **File types** list, add the types of files that the file browser should display. Click the **Add** button (  ) to open a dialog box with a list of file types and their associated extensions, which you can add to filter the list of files to display in the browser.

Use the **Move Up** (  ), **Move Down** (  ), and **Delete** (  ) toolbar buttons to organize and remove commands from the list.

By default, the **Allow entering file name** check box is selected so that users can type in a filename in the browser in addition to selecting a file from the list of matching files.

In addition, the **Settings** window contains the following sections.

### **FILE DESTINATION**

This section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either represent some sort of file destination or have children that do. The list contains, under **Declaration**, **File** nodes that you declare. It also contains settings under **Model** that support browsing for files. Such a setting is typically a text field with a **Browse** button that you find in the **Interpolation** function and geometry **Import** nodes, for example. In the tree, it appears as a **Filename** subnode (for example, under an **Interpolation** node). When you select a node that represents a file destination, the **Use as source** toolbar button (  ) below the tree becomes enabled. You can also right-click the node and choose **Use as source**. After selecting a node as the source, the node appears as the selected source underneath **Selected source**. The file scheme for accessing the file appears next to **Access using**. For an **Interpolation** node with the tag `int1`, for example, the scheme is `upload:///int1/filename`. You can also click the **Create New File and Use It as Source** button (  ) in the **File Declaration** section header to create a new variable declaration and use it as the file destination. A **Create and Use File** dialog box opens, so that you can specify the filename. The name cannot be in conflict with any existing filename. Click the **Edit Node** toolbar button (  ) below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.

## POSITION AND SIZE

This section contains all layout settings for a file import object in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the file import object using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width of the file browser. Enter a width (in points) in the **Width** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. The **Height** value is determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the file browser using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

## APPEARANCE

In this section, you can control the appearance of the background and the text in the file import object:

From the **Text color** list, select a color to use for the file import object's text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.




From the **Background color** list, select a color to use as the background in the file import: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

The font and the font size for the text use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default font size is **Default size**.

You can also select the **Bold** check box to use a boldface font or the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the file import object when users run the application. By default, the file import object is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the file import object is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## EVENTS

For certain types of form objects, you can specify a method to run when an event such as data entry occurs. The **On data change** list contains **None** (the default) and any available methods. To add a local method for the event, click the **Create Local Method** button (  ) to the right of the **On data change** list, or right-click the file import object. The selected method in the **On data change** list then changes to **Local method**. To open the local method in a method editor window, click the **Go to Source** button (  ). An empty **onDataChange** editor window then opens, where you can define the local method. (This window opens automatically when you first create a local method this way.) Click the **Remove Local Method** button (  ) to delete the local method. Methods called from the file import object support a string argument with the new client file name selected by the user. You can also right-click the file import object to create a local method or (by choosing **Edit Method** or **Edit Local Method**) to open the method associated with the command.


When you Ctrl+Alt-click the file import object:

- If the **On data change** list is set to a method, the method's editor window opens.
- If the **On data change** list is set to **None**, it creates a local method (if needed), sets the list to **Local method**, and opens the local method's editor window.

For events triggered by data change, the event is triggered after the new data value is stored in the data source.


## *Information Card Stack*




---



The **Information Card Stack** (  ) is a form object that displays a built-in card stack with cards that display information about the application, such as the computation time for the last run (if a solution is not yet available) or some other solution status information. The card stack can flip between cards in a card stack to show one at a time. You associate a card stack with a data source that controls which card to show. Each card specifies a value that it compares against the data source of the card stack. The card stack shows the first card with the matching value. If no cards match, nothing shows. Enter the name of the information card stack object in the **Name** field.

The **Settings** window contains the following sections.

### ACTIVE INFORMATION CARD SELECTOR

In this section, you specify the data source for the active card selector. The section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either represent some sort of data or have children that do. For a card stack, you typically only see the available parameters under **Parameters**, variables under **Variables**, and the data nodes defined under the **Declarations** branch. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button (  ) in the **Active Card Selector** section header, which takes you to the Model Builder, and then selecting a node in the **Model Builder** branch whose data you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Click to select the check box to include that data as an available source node for card stacks.

When you select a node that represents the source for the active card selector, the **Use as Source** toolbar button (  ) below the tree becomes enabled. You can also right-click the node and choose **Use as Source**. In addition, you can click the **Create New Declaration and Use It as Source** button (  ) or the **Create New Form Declaration and Use It as Source** button (  ) in the **Active Card Selector** section header to create a new global or local (in the form) variable

declaration for the information card stack and use it as the source. A **Create and Use Declaration** dialog box opens so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button (). Click the **Edit Node** toolbar button () below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.


After selecting a node as the source, the node appears as the selected source under **Selected source**.





If you try to use the same data source in several form objects, you may encounter some strange side effects. The initial value for the source may not be what you expect. You may also experience serious errors if the initial value of one form object is invalid for one of the other form objects.

## INFORMATION CARDS

The table in this section contains the information cards' associated activating values (in the **Activating value** column), icon (in the **Icon** column), and the text to display on the card (in the **Text** column). The stack decides which card to display through their activating values, which you type in this section. The values are then checked against the value of the source.

Click the **Add Information Card** button () to add a card to the table. The **Information Cards** dialog box then opens, where you can add some predefined cards. Click the **Add>>** and **<<Remove** buttons to add or remove information cards, respectively. Click the **Custom Card** button to open the **Edit Information Card** dialog box, where you can define the activating values, select an icon, define the text to display on the card, and select a text color (choose **Inherit** to use the form's text color) for the new card. Click **OK** to save the card's settings and add it to the cards in the information card stack.

Click the **Delete** button () or right-click the card entry in the table to remove a selected card.

Click the **Edit Information Card** button () to edit the individual card in the **Edit Information Card** dialog box that opens. In that dialog box, you can define the activating values, select an icon, define the text to display on the card, and select a text color (choose **Inherit** to use the form's text color). Click **OK** to save the card's settings.

## POSITION AND SIZE

This section contains all layout settings for an information card stack in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the information card stack using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width and height of the card stack. By default, they are determined automatically. To adjust the width, select **Manual** from the **Width** list and then enter a width (in points) in the associated field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify



a minimum width in the text field underneath. To adjust the height, select **Manual** from the **Height** list and then enter a height (in points) in the associated field. If you have chosen **Fill** from the **Vertical alignment** list, you can instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify the absolute position of the card stack using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### **APPEARANCE**

In this section, you can control the appearance of the background and the text in the information card stack.

From the **Background color** list, select a color to use as the background in the information card stack: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.


The font and the font size for the text use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default font size is **Default size**.

You can also select the **Bold** check box to use a boldface font, the **Italic** check box to use italics (an italic font), and the **Underline** check box to use underlined text.

Under **State**, you can control the initial state of the information card stack when users run the application. By default, the information card stack is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the information card stack is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

### *Array Input*

---

The **Array Input** () is a predefined form object with an input table to enter array inputs (vector inputs). The **Array Input** supports arrays as data sources. You can also add an optional label, symbol, and unit. Enter the name of the array input object in the **Name** field.






In the **Length** field, enter the length of the array as a positive integer (default: 3). The maximum length is 1000.

From the **Show vector as** list, choose **Table** (the default) to show the array components as a table, or choose **Components** to show each array component as a separate input field with a label.

In addition, the **Settings** window contains the following sections.



## SOURCE

In this section, you select the source for the array input. The section contains a tree with a filtered view of the tree in the **Application Builder** window. The nodes either represent some sort of data or have children that do. For an array input, the list contains array variables defined under **Declarations**, for example. When you select a node that represents data, the **Use as Source** toolbar button () below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. The initial values from the data source appear in the array input object. You can also click the **Create New Declaration and Use It as Source** button () or the **Create New Form Declaration and Use It as Source** button () in the **Source** section header to create a new global or local (in the form) variable declaration for the card stack and use it as the source. A **Create and Use Declaration** dialog box opens so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button (). Click the **Edit Node** toolbar button () below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.

After selecting a node as the source, the node appears as the selected source under **Selected source**.



If you try to use the same data source in several form objects, you may encounter some strange side effects. The initial value for the source may not be what you expect. You may also experience serious errors if the initial value of one form object is invalid for one of the other form objects.




From the **Initial values** list, select **From data source** (the default) to use the value specified by the selected data source, or select **Custom values**. Then, in the associated table below, enter the initial values for the components in the array.

## LAYOUT OPTIONS

This section provides settings for adding optional labels and units to the array input.

Use the **Label position** list to place a label. The options are **Above** (the default), **Left**, and **No label**. For the two first options, you can enter the desired label in the **Label text** field.

When the label position is above the table, you can include a symbol to the left of the table by selecting the **Include symbol** check box. You cannot see this setting when the label position is to the left of the table, as it overlaps with the symbol position. Enter the symbol using LaTeX syntax in the **Symbol (LaTeX encoded)** field.

As a final option, select the **Include unit** check box to add a unit symbol to the right of the table. To add a unit, click the **Select Quantity** button () to open the **Physical Quantity** dialog box to browse to find a physical quantity to use. You can also type a search string in the text field at the top of the dialog box and then click the **Filter** button () to filter the list of physical quantities. For example, type `potential` and click the **Filter** button to only list physical quantities that represent some kind of potential. Alternatively, click the **Custom Unit** button () to enter a unit (for example,  $m/s^2$ ) in the text field (the physical quantity then becomes a **Custom unit**).



If you choose to display the array input as components, the **Label** text field changes to a **Component labels** table with one row for each of the component's labels. Likewise, the **Symbol (LaTeX encoded)** field changes to a **Component symbols** table with one row for each of the component's symbols.

## POSITION AND SIZE

This section contains all layout settings for an array input in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the array input using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width of the array input. Enter a width (in points) in the **Width** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. The **Height** field is unavailable because the height of the input field is determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the array input using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### **APPEARANCE**

In this section, you can control the appearance of the background and the text in the array input.

From the **Text color** list, select **Inherit** (the default) to inherit the text color from the setting in the **Form** node, or select one of the predefined colors, such as **Black**. Select **Custom** to choose a custom text color from the color palette.

From the **Background color** list, select a color to use as the background in the array input: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.




The font and the font size for the text in the array input fields use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **Default size** for the font.

You can also select the **Bold** check box to use a boldface font or the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the array input when users run the application. By default, the array input is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the array input is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor,

the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## EVENTS

For certain types of form objects, you can specify a method to run when an event such as data entry occurs. The **On data change** list contains **None** (the default) and any available methods. To add a local method for the event, click the **Create Local Method** button (  ) to the right of the **On data change** list, or right-click the file import object. The selected method in the **On data change** list then changes to **Local method**. To open the local method in a method editor window, click the **Go to Source** button (  ). An empty **onDataChange** editor window then opens, where you can define the local method. (This window opens automatically when you first create a local method this way.) Click the **Remove Local Method** button (  ) to delete the local method. Methods called from the array input object support a string argument with the new client file name selected by the user. You can also right-click the array input object to create a local method or (by choosing **Edit Method** or **Edit Local Method**) to open the method associated with the command.


When you Ctrl+Alt-click the array input object:

- If the **On data change** list is set to a method, the method's editor window opens.
- If the **On data change** list is set to **None**, it creates a local method (if needed), sets the list to **Local method**, and opens the local method's editor window.

After a data change, an event is triggered after the new data value is stored in the data source.

## *Radio Button*


---




The **Radio Button** (  ) form object represents a group of radio buttons (option buttons) that provide a fixed number of options, from which you can choose one. It is typically useful when you have just a few options (with many options, consider using a list box or combo box instead). Enter the name of the radio button object in the **Name** field.


From the **Orientation** list, choose **Vertical** (the default) or **Horizontal** to have the radio buttons lined up vertically or horizontally.

The **Settings** window contains the following sections.

### SOURCE

In this section, you select the data source for the radio button. The section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either represent some sort of data or have children that do. For radio buttons, you can select from variables in the model and variables under **Declarations** in the Application Builder, including **Unit Set** nodes. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button (  ) in the **Source** section header, which takes you to the Model Builder, and then selecting a node in the **Model Builder** branch whose data you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Click to select the check box to include the data as an available source node for radio buttons.

When you select a node that represents data, the **Use as Source** toolbar button (  ) below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. You can also click the **Create New Declaration and Use It as Source** button (  ) or the **Create New Form Declaration and Use It as Source** button (  ) in the **Source** section header to create a new global or local (in the form) variable declaration for the radio button and use it as the source. A **Create and Use Declaration** dialog box opens so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. Click the **Edit Node** toolbar

button (  ) below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.





After selecting a node as the source, it will appear as the selected source under **Selected source**.



If you try to use the same data source in several form objects, you may encounter some strange side effects. The initial value for the source may not be what you expect. You may also experience serious errors if the initial value of one form object is invalid for one of the other form objects.

In the **Initial value** list, choose a method to define an initial value for the combo box. The options are **First allowed value**; **From data source** (the default, to use the value specified by the selected data source); and **Custom value**. For the **Custom value** option, a **Value** list shows the allowed values currently present for the form object and depends on the selected available choice lists and their values. If the data source is a setting from the embedded model that has a list of allowed values, those values are also included in the **Value** list. If a selected initial value becomes invalid because it has been removed from the choice list, for instance, it is kept as an initial value with the text **Invalid initial value** followed by the value.

### CHOICE LIST

In the **Selected** list, add **Choice List** nodes that contribute allowed values to the radio buttons, where each valid value represents one radio button. If the selected data source is a list with a set of allowed values, only a subset of those values may appear as a radio button. All other values in the selected choice lists are ignored. Available **Choice List** nodes appear under **Available**. Click the **Add Selected**  button to add the selected **Choice List** node to the list under **Selected**. Click the **Remove Selected**  button to remove a selected **Choice List** node from the list under **Selected**. You can also double-click a **Choice List** node to move it from **Available** to **Selected** and the other way around. Click the **Add New Choice List** button (  ) to open a **Choice List** window, where you can define a new choice list. Add the allowed values in the **Value** column and their corresponding names in the **Display name** column. Click **OK** to add the new choice list as a **Choice List** node (  ) under the **Declarations** node in the **Application Builder** tree and directly under **Selected**.

If you select a property that has a list of allowed values as the data source in the **Source** section, that property becomes a node initially placed in the **Selected** list. You can move it to the **Available** list, thereby clearing the list of allowed values. You can move it back again or add a custom choice list with values that also belong to the list of values for the property. If the property list and a **Choice List** node are both in the **Selected** list, they will be merged. Identical values pick the description from the first item in the list under **Selected**. In this way, you can rename one of the items in the property list. If you decide to switch the source to another property in the embedded model that also has a list of allowed values, the previous property list node is removed from both the **Available** and **Selected** lists, and the new node is added to the **Selected** list.

### POSITION AND SIZE

This section contains all layout settings for a radio button in the grid of the parent form.

You can control the horizontal and vertical alignment of the radio buttons using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, or **Right**.
- From the **Vertical alignment** list, choose **Middle**, **Top**, or **Bottom**.

In the grid mode, you can also choose **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object is useful in the sketch mode too. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in

the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

The **Width** and **Height** fields are unavailable because the dimensions of the radio buttons are determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the radio button using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### **APPEARANCE**

In this section, you can control the appearance of the background and the text in the radio button.

From the **Text color** list, select **Inherit** (the default) to inherit the text color from the setting in the **Form** node, or select one of the predefined colors, such as **Black**. Select **Custom** to choose a custom text color from the color palette.


From the **Background color** list, select a color to use as the background in the radio button: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

The font and the font size for the text in the array input fields use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **Default size** for the font.

You can also select the **Bold** check box to use a boldface font, the **Italic** check box to use italics (an italic font), and the **Underline** check box to use underlined text.

Under **State**, you can control the initial state of the radio button when users run the application. By default, the radio button is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the radio button is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.


### **EVENTS**

You can add a code method that the application runs when the data in the radio button changes. The event is triggered after the new data value is stored in the data source. The **On data change** list contains **None** (the default) and any available methods. To add a local method for this event, click the **Create Local Method** button ( **+** ). The selected method in the **On data change** list then changes to **Local method** and an editor window for the local method opens, where you can define its contents. You can also click the **Go to Source** button (  ) to open the editor window for the selected method. Click the **Remove Local Method** button ( **X** ) to delete the local method. You can also right-click the radio button to create a local method or (by choosing **Edit Method** or **Edit Local Method**) to open the method associated with the command.

When you Ctrl+Alt-click the radio button:

- If the **On data change** list is set to a method, the method's editor window opens.
- If the **On data change** list is set to **None**, it creates a local method (if needed), sets the list to **Local method**, and opens the local method's editor window.

## Selection Input

The **Selection Input** () is a form object for a selection input of some geometric entities in an application. The selection input object is similar to the selection settings in COMSOL Multiphysics models. Users can activate selections, and you can copy and paste selections into the list of selected entities, for example.





You can also add a selection as the source for a Graphics object so that users can select geometric entities directly in that Graphics object without a selection input component. In that case, the Graphics object should be used for input only, and users do not need to activate the selection.

Enter the name of the selection input object in the **Name** field.

The **Settings** window contains the following sections.

### SOURCE




In this section, you define the selection to use as the source. The section contains a tree with a filtered view of the tree in the **Model Builder** window. The nodes either represent some sort of data or have children that do. For a selection input, the tree contains **Explicit** selection nodes from the model, which you can choose as the source for the selection. The selection is then available for the application to use and contains updated selections made by the user. When you select a node that represents data, the **Use as Source** toolbar button () below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. Click the **Edit Node** toolbar button () below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.

After selecting a node as the source, it will appear as the selected source under **Selected source**.



If you try to use the same data source in several form objects, you may encounter some strange side effects. The initial value for the source may not be what you expect. You may also experience serious errors if the initial value of one form object is invalid for one of the other form objects.

### GRAPHICS TO USE WHEN ACTIVE

When users of the application set the activation switch to ON, you can connect the selections to a Graphics object where the selected geometric entities are highlighted. Select a Graphics object from the tree and click the **Use Graphics** button to add it under **Selected graphics**. Users can also make selections directly in the Graphics object. If the Graphics object is set up to include the standard plot toolbar, that toolbar also includes the **Zoom to Selection** () , **Select Box** ( in 3D), and **Deselect Box** () buttons. If multiple selection input objects are connected to the same Graphics object, only one of the selections can be active at a time.



Users cannot set the activation switch to OFF. The switch becomes switched off if another selection input is activated or if the associated Graphics object gets a source other than the selection.

## POSITION AND SIZE

This section contains all layout settings for a selection input object in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the selection input form object using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too. You can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width of the selection input. Enter a width (in points) in the **Width** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. The **Height** field is unavailable because the height of the input field is determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the selection input using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

### Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields



Hover selection is not supported in the web client. Instead, a preselection appears, which shows what will be selected when you click the mouse. This preselection disappears when you move the mouse. To select underlying objects, use the mouse wheel. In contrast to hover selection, the top layer is also included as the first layer. To make it easier to choose the closest object, the top layer object under the mouse pointer is selected if there is no preselection, which means that an object other than the preselected one might be selected if you move the mouse before you click.

## APPEARANCE

In this section, you can control the appearance of the background and the text in the selection input.

From the **Text color** list, select **Inherit** (the default) to inherit the text color from the setting in the **Form** node, or select one of the predefined colors, such as **Black**. Select **Custom** to choose a custom text color from the color palette.



From the **Background color** list, select a color to use as the background in the selection input: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.


The font and the font size for the text in the array input fields use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default is to use the **Default size** for the font.

You can also select the **Bold** check box to use a boldface font or the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the selection input when users run the application. By default, the selection input is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the selection input is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## *Text*

---


The **Text** () form object is a predefined form for a text field with default text. It can either be static text that provides some information or editable text so that users can add notes or comments, for example. Enter the name of the text object in the **Name** field.

Select the **Editable** check box to make it possible for users to edit and add text. By default, the text is static.





The **Wrap text** check box is selected by default. Click it to disable wrapping of the text. A scroll bar appears if the text does not fit in its defined dimensions.

The **Settings** window contains the following sections.




### **SOURCE**

In this section, you define a source for the text object. The section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either represent some sort of data or have children that do. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button () in the **Source** section header, which takes you to the Model Builder, and then selecting a node in the **Model Builder** branch whose data you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Click to select the check box to include that data as an available source node for text objects.


If the **Editable** check box is cleared so that the text is read-only, you can choose to use one of the following information nodes, which you find under the main **Model** node and under each **Study** node, as the source.

- The **Expected Computation Time** node () under **Model>Information** (): The expected computation time is a value that the application developer can enter in the **Expected** field in the **Root** node's **Settings** window.
- The **Last Computation Time** node () under **Model>Information**: This node shows the last measured computation time for the last computed study.
- The **Last Computation Time** node () under each **Model>Study>Information**: This node shows the last measured computation time for the study.

When you start an application for the first time, the last measured times are reset, displaying **Not available yet**.

When you select a node that represents data, the **Use as Source** toolbar button () below the tree is enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. You can also click the **Create New Declaration and Use It as Source** button () or the **Create New Form Declaration and Use It as Source** button () in the **Source** section header to create a new global or local (in the form) variable declaration for the text and use it as the source. A **Create and Use Declaration** dialog box opens



so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. Click the **Edit Node** toolbar button (  ) below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.

After selecting a node as the source, the node appears as the selected source under **Selected source**.



If you try to use the same data source in several form objects, you may encounter some strange side effects. The initial value for the source may not be what you expect. You may also experience serious errors if the initial value of one form object is invalid for one of the other form objects.

From the **Initial value** list, choose **From data source** (the default) to use the text from the source. Alternatively, choose **Custom value** to add text to display in the **Value** text field below.

### POSITION AND SIZE

This section contains all layout settings for a text object in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the text object using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width and height of the text input. Enter a width (in points) in the **Width** field and a height (in points) in the **Height** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically this means a minimum size of 0), or choose **Manual** to specify a minimum width in the text field underneath. If you have chosen **Fill** from the **Vertical alignment** list, you can instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify the absolute position of the text input using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

### Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object

- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

## APPEARANCE

In this section, you can control the appearance of the background and the text.

From the **Text color** list, select a color to use for the text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

From the **Background color** list, select a color to use as the background for the text: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.




From the **Text alignment** list, select an alignment for the text: **Left** (the default), **Center**, or **Right**.

The font and the font size for the text use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default font size is **Default size**.

You can also select the **Bold** check box to use a boldface font or the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the text object when users run the application. By default, the text object is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the text object is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## EVENTS

You can add a code method that the application runs when the text changes. The event is triggered after the new data value is stored in the data source. The **On data change** list contains **None** (the default) and any available methods. To add a local method for this event, click the **Create Local Method** button (  ). The selected method in the **On data change** list then changes to **Local method**. (The editor window opens automatically when you first create a method this way.) To open the selected method in an editor window, click the **Go to Source** button (  ). An empty **onDataChange** editor window then opens, where you can define the local method. Click the **Remove Local Method** button (  ) to delete the local method.


You can also right-click the text object to create a local method or (by choosing **Edit Method** or **Edit Local Method**) to open the method associated with the command.

When you Ctrl+Alt-click the text object:

- If the **On data change** list is set to a method, the method's editor window opens.
- If the **On data change** list is set to **None**, it creates a local method (if needed), sets the list to **Local method**, and opens the local method's editor window.


## List Box





---

The **List Box** (  ) form object represents a list box. If you use a list as the source, you can select more than one item in the list using Shift-click or Ctrl-click. For other sources, you can only select one value from the list. Enter the name of the list box object in the **Name** field.

The **Settings** window contains the following sections.

## SOURCE

In this section, you define the data source for the list box. The section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either represent some sort of data or have children that do. For a list box, the tree contains variables and parameters in the model and variables, such as a 1D array that you have added under **Declarations**, including **Unit Set** nodes. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button () in the **Source** section header, which takes you to the Model Builder, and then selecting a node in the **Model Builder** branch whose data you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Click to select the check box to include the data as an available source node for list boxes.

When you select a node that represents data, the **Use as Source** toolbar button () below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. You can also click the **Create New Declaration and Use It as Source** button () or the **Create New Form Declaration and Use It as Source** button () in the **Source** section header to create a new global or local (in the form) variable declaration for the list box and use it as the source. A **Create and Use Declaration** dialog box opens so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. Click the **Edit Node** toolbar button () below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.

After selecting a node as the source, the node appears as the selected source under **Selected source**.







If you try to use the same data source in several form objects, you may encounter some strange side effects. The initial value for the source may not be what you expect. You may also experience serious errors if the initial value of one form object is invalid for one of the other form objects.

In the **Initial value** list, choose a method to define an initial value for the list box. The options are **First allowed value**; **From data source** (the default, to use the value specified by the selected data source); and **Custom value**. For the **Custom value** option, a **Value** list shows the allowed values currently present for the form object, which depends on the selected available choice lists and their values. If the data source is a setting from the embedded model that has a list of allowed values, those values are also included in the **Value** list. If a selected initial value becomes invalid because it has been removed from the choice list, for instance, it is kept as an initial value with the text **Invalid initial value** followed by the value.

When the data source is a string array, the **Initial value** list is there along with the **Empty array** option that sets an empty array as the default for the selected source. There are also additional choices for how to select values. From the **Select values in** list, select **Dialog** to use a dialog to display the list box and enter a **Dialog title** in the field below, or select **List box** to use a multiselect list box. The **Dialog** option uses a list with buttons underneath, which users can use to add, delete, and move items in the list of selected items. This is suitable for a list with many items where you want to better control which items are selected (compared to a standard multiselect list box).

## CHOICE LIST

In the **Selected** list, add **Choice List** nodes that contribute allowed values to the list box. If the selected data source is a list with a set of allowed values, only a subset of the values can appear in the allowed values of the list box. All other values in the selected choice lists are ignored. Available **Choice List** nodes appear under **Available**. Click the **Add Selected**  button to add the selected **Choice List** node to the list under **Selected** or click the **Remove Selected**  button to remove a selected **Choice List** node from the list under **Selected**. You can also double-click a **Choice List** node to move it from **Available** to **Selected** and the other way around. Click the **Add New Choice List** button () to open a **Choice List** window where you can define a new choice list. Add the allowed values in the **Value** column and

their corresponding names in the **Display name** column. Click **OK** to add the new choice list as a **Choice List** node (  ) under **Declarations** and directly under **Selected**.

If you select a property that has a list of allowed values as the data source in the **Source** section, that property becomes a node initially placed in the **Selected** list. You can move it to the **Available** list, thereby clearing the list of allowed values. You can move it back again or add a custom choice list with values that also belong to the list of values for the property. If the property list and a choice list node are both in the **Selected** list, they will be merged. Identical values pick the description from the first item in the list under **Selected**. In this way, you can rename one of the items in the property list. If you decide to switch the source to another property in the embedded model that also has a list of allowed values, the previous property list node is removed from both the **Available** and **Selected** lists, and the new node is added to the **Selected** list.

## POSITION AND SIZE

This section contains all layout settings for a list box in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the list box using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too. You can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width and height of the list box. Enter a width (in points) in the **Width** field and a height (in points) in the **Height** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. If you have chosen **Fill** from the **Vertical alignment** list, you can instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify the absolute position of the list box using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

## APPEARANCE

In this section, you can control the appearance of the text in the list box.




From the **Text color** list, select a color to use for the text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

The font and the font size for the text use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default font size is **Default size**.

You can also select the **Bold** check box to use a boldface font or the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the list box when users run the application. By default, the list box is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the list box is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.


## EVENTS

You can add a code method that the application runs when the data in the list box changes. The event is triggered after the new data value is stored in the data source. The **On data change** list contains **None** (the default) and any available methods. To add a local method for this event, click the **Create Local Method** button (  ). The selected method in the **On data change** list then changes to **Local method**. To open the selected method in a method editor window, click the **Go to Source** button (  ). An empty **onDataChange** editor window then opens, where you can define the local method. Click the **Remove Local Method** button (  ) to delete the local method. You can also right-click the list box to create a local method or (by choosing **Edit Method** or **Edit Local Method**) to open the method associated with the command.

When you Ctrl+Alt-click the list box:

- If the **On data change** list is set to a method, the method's editor window opens.
- If the **On data change** list is set to **None**, it creates a local method (if needed), sets the list to **Local method**, and opens the local method's editor window.

## Table


The **Table** (  ) form object represents a table with rows and columns. Enter the name of the table object in the **Name** field. The table columns include headers. Clear the **Show headers** check box to remove the headers.





Select the **Automatically add new rows** check box to make the table add a new row when the user enters new values in the same way as for tables containing global parameters and variables in the COMSOL Desktop.

To make the table rows sortable, select the **Sortable** check box. This way, users can sort the rows by clicking the table column headers. The rows are sorted in the following sequence: ascending, descending, and then unsorted (the original row order in the table).




In addition, the **Settings** window contains the following sections.

## SOURCES

In this section, you add the sources for the table data. The section contains a tree with a filtered view of the tree in the **Application Builder** window. The nodes either represent some sort of data or have children that do. For a table, arrays that you have added under **Declarations** are available as the data source for a table. When you select a node that represents data, the **Add to Table** toolbar button (  ) below the tree becomes enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Add to Table**. The data source you select

becomes the source of the data displayed in the table, and the initial values from the data source appear in the table input object. You can also click the **Create New Declaration and Use It as Source** button (  ) or the **Create New Form Declaration and Use It as Source** button (  ) in the **Sources** section header to create a new global or local (in the form) variable declaration for the table and use it as the source. A **Create and Use Declaration** dialog box opens so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. You can extend the list of available data nodes by clicking the **Switch to Model Builder and Activate Data Access** button (  ). Click the **Edit Node** toolbar button (  ) below the tree to move to the corresponding node.

In the table below, there is a row for each column, where you can edit the header text under **Header**, the width (in pixels) under **Width**, and whether or not the table data is editable (the default) under **Editable**. The table columns where you select the **Grow** check box can grow horizontally. To make it possible for table columns to grow, the form in which you add the table must use the Grid mode and you must select **Fill** from the **Horizontal alignment** list under **Position and Size** (see below). You can adjust the alignment of the column's data presentation using the lists in the **Alignment** column. Choose **Left** (the default), **Center**, or **Right**. The rightmost column, **Data source**, lists the sources for the data in each column.

Click the **Move Up** and **Move Down** buttons (  and  ) to move and rearrange the column order. Click the **Delete** button (  ) to delete the selected column.

#### Working with Data Sources

When you work with data sources for tables, the following rules apply when changing sources to a table from methods, for example, but also if you perform a `set` command on a specific data source that is part of a table. In the rules below, the *first data source* of a table is the source that controls the number of rows in the table. All other sources used by the table will be padded with default element values or cropped to match the length of the first data source. The first data source has a lock symbol on its tree node.

- Clearing data only requires the first data source to be cleared.
- Adding one row only requires that the first data source gets a new row.
- Removing the last row only requires removing the last row of the first data source.
- Inserting a row must be done by inserting the row in the first data source, then inserting it for the other properties.
- Removing a row must be done by removing the row for all other sources first, and remove it for the first data source last.


#### TOOLBAR


In this section, you can add items to a table toolbar and activate the following table options:

From the **Position** list, choose where you want to position the toolbar relative to the table: **Below** (the default), **Above**, **Left**, or **Right**.

You can choose from two icon sizes: From the **Icon size** list, choose **Small** (the default) or **Large**.



In the table below, you can add one or more buttons to form a table toolbar.





Click the **Add Toolbar Commands** button (  ) to open the **Toolbar Buttons** dialog box, where you can select and add one or more of the following table commands as toolbar buttons.

Under **Move** (  ), double-click the **Move Up** (  ) and **Move Down** (  ) buttons to add corresponding buttons for moving rows.

Under **Modify** (  ):


- Double-click the **Add** button (  ) to add a button for adding rows.





- Double-click the **Delete** button (  ) to add a button for deleting selected rows.
- Double-click the **Clear Table** button (  ) to add a button for clearing the entire table of all its contents.

Under **File** (  ), double-click the **Load from File** (  ), **Clear Table and Load from File** (  ), and **Save to File** (  ) buttons to add corresponding buttons for loading table data from a file, clearing the table first and then loading new content from a file, and saving table data to a file, respectively. The file types that users can load and save table data to and from include text files (.txt); CSV files (.csv); data files (.dat); and, if the license includes LiveLink™ for Excel®, Microsoft Excel® files (.xlsx). Allowed data separators are comma, semicolon, and tab for CSV files, and space and tab for other non-Excel® files.

Double-click the **Move**, **Modify**, and **File** buttons to add all buttons in those groups. Alternatively, click the **Add** and **Remove** buttons to add and remove the selected buttons, respectively. Click the **Custom Button** button to open the **Edit Custom Toolbar Item** dialog box (see [The Edit Custom Toolbar Item Dialog Box](#)).

Click **OK** to close the **Toolbar Buttons** dialog box and add the selected commands as buttons in the table's toolbar. If you open the **Toolbar Buttons** dialog box again, it's left side contains only the table commands that you have not added yet.

Click the **Add Separator** button (  ) to add a separator between groups of buttons in the toolbar.

Select a button in the table and click the **Edit** button (  ) if you want to change the appearance or behavior of a custom toolbar button in the **Edit Custom Toolbar Item** dialog box. Click the **Move Up** and **Move Down** buttons (  and  ) to move and rearrange the toolbar button order. Click the **Delete** button (  ) to delete the selected button.

The table contains a row for each added item, showing its name, icon, text, and tooltip in the **Name**, **Icon**, **Text**, and **Tooltip** columns, respectively.

## POSITION AND SIZE

This section contains all of the layout settings for a table in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the table using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the height of the table. Enter a height (in points) in the **Height** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. The **Width** field is unavailable because the width of the table is determined by the software.



Additionally, in the sketch mode, you can specify the absolute position of the table using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### Cell Margin

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### APPEARANCE

In this section, you can control the appearance of the text in the table.


From the **Text color** list, select a color to use for the text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

The font and the font size for the text in the table use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default font size is **Default size**.

You can also select the **Bold** check box to use a boldface font or the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the table when users run the application. By default, the table is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the table is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

### EVENTS

For certain types of form objects, you can specify a method to run when an event such as data entry occurs. The **On data change** list contains **None** (the default) and any available methods. To add a local method for an event, click the **Create Local Method** button ( **+** ) to the right of the **On data change** list. The selected method in the **On data change** list then changes to **Local method** and opens the editor window for the new method. To open the selected method in a method editor window, click the **Go to Source** button (  ). An empty **onDataChange** editor window opens, where you can define the local method. Click the **Remove Local Method** button ( **X** ) to delete the local method. You can also right-click the table object to create a local method or (by choosing **Edit Method** or **Edit Local Method**) to open the method associated with the command,

When you Ctrl+Alt-click the table object:

- If the **On data change** list is set to a method, the method's editor window opens.
- If the **On data change** list is set to **None**, it creates a local method (if needed), sets the list to **Local method**, and opens the local method's editor window.








After a data change, an event is triggered after the new data value is stored in the data source.



## The Edit Custom Toolbar Item Dialog Box

---

There are several ways to add a custom button or toggle button to a toolbar object, for a table, or for a graphics object. To create a button item or toggle button item and open the **Edit Custom Toolbar Item** dialog box to define it, use one of the steps below. The **Edit Custom Toolbar Item** dialog box provides settings and tools for creating a custom button or toggle button with an associated command or method to run.



- For a **Toolbar** object, click the **Add Item** () or **Add Toggle** button () underneath the table of items in the **Toolbar Item** section, or right-click the toolbar in the Form window and choose **Add Item** () or **Add Toggle** button (). You can also open it by right-clicking a custom toolbar button or toggle button in the **Toolbar** object's **Settings** window and choosing **Edit**.
- For a **Table** object, click the **Add Toolbar Item** button () in the **Toolbar** section of the **Settings** window. Then, in the **Toolbar Items** dialog box, click the **Custom Item** button or the **Custom Toggle Item** button to open the **Edit Custom Toolbar Item** dialog box. You can also open it by right-clicking a custom toolbar button or toggle button in the **Table** object's **Settings** window and choosing **Edit**.
- For a **Graphics** object, click the **Add Item** () or **Add Toggle** button () at the bottom of the **Toolbar** section in the **Settings** window. You can also open it by right-clicking a custom toolbar button or toggle button in the **Graphics** object's **Settings** window and choosing **Edit**.

The **Edit Custom Toolbar Item** dialog box includes the following pages:

### GENERAL

In the **Name** field, type the name of the toolbar button that you use to refer to it.

In the **Text** field, type the text that appears as a label on the button.

From the **Icon** list, choose **None** for no icon or choose an icon from existing image files. Click the **Add Image to Library and Use Here** button () to browse and select an image to use as the icon. Click the **Export** button () to save the icon as an image file. If you use an icon and a text label, the toolbar item includes both the icon and the label.

In the **Tooltip** field, enter an explanatory text that will appear as the tooltip for the button or toggle button. In the **Toolbar Items** dialog box, the added button appears with its tooltip, if the text label is empty.

To add a keyboard shortcut, make the **Keyboard shortcut** field active, and then type a keyboard shortcut on the keyboard:

You must use a modifier in the keyboard shortcut, not just a plain letter (for example, CTRL+SHIFT+D). The shortcut can include the Ctrl key (CTRL), Alt key (ALT), and Shift key (SHIFT). Note that the Ctrl key is interpreted as Command on OS X. Avoid using the following keys in your shortcut:

- Backspace, as it can be used to clear a shortcut
- Delete, as it can be used to clear a shortcut
- Escape
- Alt on its own (to avoid conflicts with File menu shortcuts)






It is possible to override other keyboard shortcuts, so take care when choosing the shortcut key combinations to use.

---

## SOURCE



The **Source** page is only available for toolbar toggle buttons.

Here you specify the source for the state of a toggle button in a toolbar. The source can be string or Boolean variable that is created under **Declarations**, which you select from the tree and then click the **Use as Source** button (). Alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. You can also click the **Create New Declaration and Use It as Source** button () or the **Create New Form Declaration and Use It as Source** button () below the source list to create a new global or local (in the form) variable declaration for the toolbar item and use it as the source. A **Create and Use Declaration** dialog box opens so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. When you have specified a source, use these settings to define its initial state:

From the **Initial value** list, choose **From data source** (the default) to use the initial value from the data source, or choose **Custom value** to define an initial state using the **Initial state** list: Choose **Selected** (the default) or **Cleared** (the value for selected is on; for cleared, it is off).

### CHOOSE COMMANDS TO RUN

This section contains a tree with a filtered view of the tree in the **Application Builder** window. The nodes either support a command or have children that do. When you select a node that supports one or more commands, the corresponding command toolbar buttons are enabled in the toolbar below the tree. You can also right-click a node to get a list of available commands for that particular node. Once you click on a command with a node selected (or press Enter or double-click to add a command with its default command such as **Run**, **Plot**, or **Set Value**), the command and node appear in the last row of the table below the tree. This table contains all nodes that run, and you can delete and move commands using the toolbar below the table.

In the **Model** branch, all nodes that represent some sort of data value, such as a parameter under the **Parameters** node, support the **Set Value** command. When adding a **Set Value** command to the table, the third column, **Arguments**, becomes enabled. In this column you type the value to set. For data that represents arrays, use curly braces and commas to enter the array elements. For example, enter `{1, 2, 3}` to set a three-element array with the values 1, 2, and 3. See [The Array 1D String Node](#) for more details on how to enter arrays and matrices. For nodes that represent a file import, such as a **Filename** node under an **Interpolation** function node, an **Import File** command is available.

The tree includes a number of branches from the application tree in addition to the **Model** branch:

- The **Forms** branch: **Form** nodes support the commands **Show**, which sets the form as the main form of the application (that is, the content of the application window will be this form), and **Show as Dialog**, which brings up the form as a separate dialog window.
- The **GUI Commands** branch: The commands under this branch are grouped in three subcategories:
  - **File Commands**: These include **Save Application** (to save the application under its current name); **Save Application As** (to open a file browser dialog allowing the user to save the application in a suitable location); **Save Application on Server**; **Save Application on Server As**; **Open File** (to open an application file resource specified using a valid URI path in the **Arguments** column); **Save File As** (similarly, to allow the user to save the file under a name specified in the **Arguments** column); and **Exit Application** (to close the running application). If the







application is run on COMSOL Server, the **Save Application on Server** and **Save Application on Server As** commands save the current state as a new application in the COMSOL Server Application Library.

- **Graphics Commands:** Here you find the commands **Zoom Extents**, **Reset Current View**, **Scene Light**, **Transparency**, **Print**, **Select All**, and **Clear Selection**. For all graphics commands, add the name of the Graphics object that you want to apply the command to as an argument.
- **Model Commands:** Here you find the commands **Clear All Solutions** and **Clear All Meshes**.

Double-click or right-click any of the nodes above to add a **Run** command.

- The **Declarations** branch: This branch contains any variable declarations you have added under the **Application Builder** window's **Declarations** branch grouped by type. Like parameters, they support the **Set Value** command.
- The **Form Declarations** branch: This branch contains any variable declarations you have added under a **Declarations** branch under the current **Form** node. Like parameters, they support the **Set Value** command.
- The **Methods** branch: **Method** nodes support the **Run** command.
- The **Form Methods** branch: **Method** nodes under the current **Form** node support the **Run** command.
- The **Libraries** branch: Under **Sounds**, you can choose between sound files to play in a command sequence.


When you click one of the buttons underneath the tree, the currently selected command appears in the **Command** column in the table below. There are also **Icon** and **Arguments** columns, where you can enter any applicable arguments that the command uses.

Click the **Convert to Method** toolbar button () and choose **Convert to Method** or **Convert to Form Method** to convert the entire list of commands in the table to a global or form method that contains the equivalent code. After this operation, the list of commands only contains a single **Run** operation on the created method. When you select a method under **Command**, or there is exactly one method in the list, you can go to the editor window for that method by clicking the **Go to Method** button (). For information about the **Edit Argument** button () see [Editing Initial Values and Arguments in Declarations and Command Sequences](#). Use the **Move Up** () , **Move Down** () , and **Delete** () toolbar buttons to organize and remove commands from the list (and to remove the local method, if deleted).

Click **OK** to close the **Edit Custom Toolbar Item** dialog box and add the button to the toolbar.

## Slider

---

The **Slider** () is a form object for choosing a numerical input using a slider control. Enter the name of the slider object in the **Name** field.

From the **Value type** list, choose **Integer** or **Real** (the default), depending on the type of data in the data source for the slider.

In the **Minimum value** (default: 0) and **Maximum value** (default: 1) fields, enter the minimum and maximum values that define the range of the data covered by the slider.

Enter the number of steps (resolution) for the slider in the **Number of steps** field (default: 5).


From the **Orientation** list, choose **Horizontal** (the default) or **Vertical** to change the slider orientation from horizontal to vertical or vice versa.





In the **Tooltip** field, enter text that will appear as a tooltip when the user hovers the pointer over the slider.

You can append a unit to the number for the slider's value by selecting the **Append unit to number** check box and typing a unit in the associated text field. Units are only applicable to sources that are string declarations and parameters.

In addition, the **Settings** window contains the following sections.

## SOURCE

In this section, you define the variable to use as the data source for the slider. The section contains a tree with a filtered view of the trees in the **Application Builder** and **Model Builder** windows. The nodes either represent some sort of data or have children that do. For a slider, you can use a scalar value from a variable in the model or a variable defined under **Declarations**. To extend the list of available data nodes, click the **Switch to Model Builder and Activate Data Access** button () in the **Source** section header, which takes you to the Model Builder, and then select a node in the **Model Builder** branch whose data you want to access. With this button active, the **Settings** window of the selected node displays a **Select Data Source** check box (a green square) next to the settings that you can include. Click to select the check box to include that data as an available source node for sliders.

When you select a node that represents data, the **Use as Source** toolbar button () below the tree is enabled. You can click it or, alternatively, press Enter, double-click, or right-click the node and choose **Use as Source** to add it as the selected source. You can also click the **Create New Declaration and Use It as Source** button () or the **Create New Form Declaration and Use It as Source** button () in the **Source** section header to create a new global or local (in the form) variable declaration for the slider and use it as the source. A **Create and Use Declaration** dialog box opens so that you can select the data type of the source (if applicable), its name, and its initial value (if applicable). The name cannot be in conflict with any existing variable declaration. Click the **Edit Node** toolbar button () below the tree to move to the corresponding node. If necessary, the program switches to the Model Builder.

After selecting a node as the source, the node appears as the selected source under **Selected source**.



If you try to use the same data source in several form objects, you may encounter some strange side effects. The initial value for the source may not be what you expect. You may also experience serious errors if the initial value of one form object is invalid for one of the other form objects.

From the **Initial value** list, select **From data source** to use the value specified by the selected data source, or select **Custom value** to enter the initial value (starting value) for the slider (default: 0) in the **Value** field below. Typically, the initial value and the minimum value are the same.

## UNIT

From the **Method** list, select one of the following methods to specify the unit for the slider:

- **No unit**, the default option.
- **Append unit to number**: The unit that you type in the **Unit expression** field is appended to the number corresponding to the slider's position.
- **Append unit from unit set**: The slider appends a unit from a **Unit Set** node added under **Declarations** (see [The Unit Set Node](#)). You specify the unit set to use from the **Unit set** list and the unit to use from **Unit list**, which lists all defined properties and their units from the select unit tests. There is also a **No unit** option.

## APPEARANCE

Under **Appearance**, you can control the initial state of the slider when users run the application. By default, the slider is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the slider is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## POSITION AND SIZE

This section contains all of the layout settings for a slider in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the slider using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width of the slider. Enter a width (in points) in the **Width** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. The **Height** field is unavailable because the height of the slider is determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the slider using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.




---

### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the slider. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the slider
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the slider in the **Horizontal** and **Vertical** text fields

### **EVENTS**

For certain types of form objects, you can specify a method to run when an event such as data entry occurs. The **On data change** list contains **None** (the default) and any available methods. To add a local method for an event, click the **Create Local Method** button (  ) to the right of the **On data change** list, or right-click the slider object. The selected method in the **On data change** list then changes to **Local method** and an empty **onDataChange** editor window for the new method opens automatically. To open the local method in a method editor window later, click the **Go to Source** button (  ) and click the **Remove Local Method** button (  ) to delete the local method. Methods called from the slider support a double argument holding the new value of the data source. You can also right-click the slider object to create a local method or (by choosing **Edit Method** or **Edit Local Method**) to open the method associated with the command.

When you Ctrl+Alt-click the slider object:


- If the **On data change** list is set to a method, the method's editor window opens.
- If the **On data change** list is set to **None**, it creates a local method (if needed), sets the list to **Local method**, and opens the local method's editor window.

After a data change, an event is triggered after the new data value is stored in the data source.

By default, the **Trigger while dragging** check box is selected. The event is then triggered while dragging the slider. If the method that is connected to the event takes some time to run, the slider can appear to be sluggish. If you clear the **Trigger while dragging** check box, the event is instead only triggered when you stop dragging and release the slider.

## Hyperlink

---

The **Hyperlink** () form object provides the possibility to add a hyperlink to a web page with additional or related information.

Enter the name of the hyperlink object in the **Name** field.

In the **Text** field, enter the text that appears in the form as a clickable hyperlink.

In the **URL** field, enter either a valid web address or an email address.

A valid web address, such as `www.comsol.com`, does not normally need the `http://` or `https://` prefix. The web page will open in the user's default browser.

For email addresses, use the `mailto` format, such as `mailto:info@comsol.com`, to provide a hyperlink for sending an email. It can also include a specified subject and message body. The hyperlink will launch the user's default email application program and prepare a new message, where the **To** field is set to the specified address. Note that this way of interactively sending an email from a COMSOL application differs from using the built-in email method.

The **Settings** window contains the following sections.

### POSITION AND SIZE

This section contains all layout settings for a hyperlink object in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the hyperlink object using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width and height of the hyperlink object. Enter a width (in points) in the **Width** field and a height (in points) in the **Height** field. If you chose **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. If you have chosen **Fill** from the **Vertical alignment** list, you can instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify the absolute position of the hyperlink object using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the hyperlink object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

### **APPEARANCE**

In this section, you can control the appearance of the background and the hyperlink text.

From the **Background color** list, select a color to use as the background for the text: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.


The font and the font size for the text use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default font size is **Default size**.

You can also select the **Bold** check box to use a boldface font or the **Italic** check box to use italics (an italic font).

Under **State**, you can control the initial state of the hyperlink object when users run the application. By default, the hyperlink object is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the hyperlink object is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

### *Toolbar*







---

The **Toolbar** () contains the specifications of a toolbar placed as any other form object in a form. It shows a bar of menu buttons below the title of the window. Clicking a menu button runs a sequence of commands. It works like any other menu, except that it does not support submenus. Right-click the toolbar object in the form window and choose **Item** to add an action item to the toolbar (see [The Item Node](#)) or choose **Separator** to add a separator between groups of toolbar buttons (see [The Separator Node](#)). You can also add and edit the toolbar's contents in the **Toolbar Items** section. Enter the name of the toolbar object in the **Name** field.

You can choose from two icon sizes: From the **Icon size** list, choose **Small** (the default) or **Large**.

The **Settings** window contains the following sections.

### **TOOLBAR ITEMS**

The table in this section lists the current toolbar items and separators with their names, icons, and text. You can edit the text directly in the **Text** column. Right-click an item to move it; delete it; or, for custom buttons, edit it in the **Edit Custom Toolbar Item** window that opens. You can also use the **Move Up** () , **Move Down** () , **Delete** () , and **Edit** () buttons underneath the table. Click the **Item** button () to add an item to the toolbar. Click the **Separator** button () to add a separator.

The table contains a row for each added item, showing its name, icon, text, and tooltip in the **Name**, **Icon**, **Text**, and **Tooltip** columns, respectively.

## POSITION AND SIZE

This section contains all layout settings for a toolbar in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the toolbar using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

The **Width** and **Height** fields are unavailable because the dimensions of the toolbar are determined by the software.

Additionally, in the sketch mode, you can specify the absolute position of the toolbar using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---

### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

## APPEARANCE

In this section, you can control the appearance of the background and the text in the toolbar.

From the **Text color** list, select a color to use for the text: **Inherit** (the default; the form object then uses the setting from the Form it is located in), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

From the **Background color** list, select a color to use as the background for the toolbar text: **Transparent** (the default), any of the predefined basic colors, or **Custom**, which makes it possible to select a custom color from a color palette.

The font and the font size for the text use the font settings from the **Forms** node by default. Select a font from the **Font** list: **Default font** or any of the available fonts. If needed, choose or enter a font size (in points) in the **Font size** combo box. The default font size is **Default size**.


You can also select the **Bold** check box to use a boldface font or the **Italic** check box to use italics (an italic font).



Under **State**, you can control the initial state of the toolbar when users run the application. By default, the toolbar is visible and enabled. Clear the **Visible** or **Enabled** check box if you want to make the initial state so that the toolbar is hidden or unavailable. You can then make it visible or enable it using a method. In the form editor, the state of the form object is indicated by a change in its appearance. Objects that are hidden become visible when selected in the form editor.

## Spacer

---

The **Spacer** () form object is invisible in the user interface. It defines a space of absolute size that you can use to ensure that neighboring form objects have enough space to show their contents. Typically, you use a spacer next to tables or plots to ensure that they display properly. If the user resizes the window so it becomes smaller than the size of the spacer, the effective size of the window is maintained by displaying scroll bars. Enter the name of the spacer object in the **Name** field.

The **Settings** window contains the following section.

### POSITION AND SIZE

This section contains all layout settings for a spacer in the grid of the parent form.

In the grid mode, you can control the horizontal and vertical alignment of the spacer using the following lists:

- From the **Horizontal alignment** list, choose **Left**, **Center**, **Right**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).
- From the **Vertical alignment** list, choose **Middle**, **Top**, **Bottom**, or **Fill**, which automatically expands the form object to fill the cell in the horizontal or vertical direction (where applicable).

The need to specify the alignment is most obvious when working in the grid mode, as it controls how the form object is aligned in its grid cell. Aligning the form object can be useful in the sketch mode too, and you can then use the alignment tools on the **Arrange** menu in the **Form** toolbar's **Sketch** section. When running the application in any client other than the Windows client, the form objects may not be positioned exactly as seen in the form windows. This is because the form objects may have a different size in other clients, giving them a slightly different positioning. Specifying the alignment ensures that the form objects are aligned as you want them to be in all clients.

You can also specify the width and height of the spacer. Enter a width (in points) in the **Width** field and a height (in points) in the **Height** field. If you have chosen **Fill** from the **Horizontal alignment** list, you can instead specify a **Minimum width**. Choose **Automatic** to compute the minimum width automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum width in the text field underneath. If you have chosen **Fill** from the **Vertical alignment** list, you can instead specify a **Minimum height**. Choose **Automatic** to compute the minimum height automatically (typically, this means a minimum size of 0) or choose **Manual** to specify a minimum height in the text field underneath.

Additionally, in the sketch mode, you can specify absolute position of the spacer using the **Position x** and **Position y** fields. In the grid mode, you can position the object in the grid and see the grid position as the **Row**, **Column**, **Row span**, and **Column span** values.



The form windows update these size and positioning settings through the draw operations performed there, such as resizing and moving the object.

---



The **Width** and **Height** settings specify the area the spacer occupies. Usually, you set either the width or height to a very small value and the other to the desired size in one direction. Use two fillers to enforce a size in two directions.

---

### *Cell Margin*

Under **Cell margin** (in grid mode only), you can control the margins around the form object. By default, the margins are taken from the parent form. From the **Cell margin** list, choose:

- **None**, for no margin around the form object
- **From parent form** (the default), to use the margins set for the parent form
- **Custom**, to specify the margins for the form object in the **Horizontal** and **Vertical** text fields

## Working with Methods

In this chapter, learn about the tools in the Application Builder that you can use to create methods with code for your application or model. These tools provide language elements, automatic code recording, syntax checking, search tools, debugging tools, and other functionality for creating, editing, testing, and debugging methods and utility classes. Also see the

In this chapter:



- [Overview](#)
- [Creating Methods](#)
- [Debugging Methods for Applications](#)

# Overview

The Application Builder provides tools and editor windows designed for writing code that the application uses to perform tasks that extend its built-in functionality.



## *Opening a Method Editor Window*

---

You activate the method editor tools from the **Method** ribbon toolbar. To open a method editor window, click the tab, create a new method, double-click a **Method** node, or right-click a **Method** node and choose **Edit**. You can have several method editor windows open. The tabs for method editor windows show  (for application methods) or  (for model methods) and the name of the method.


## *Coding and Methods Overview*

---

In the Application Builder, custom code is represented by a **Method** node. To write code, you must first create a **Method** node. To create a **Method** node, click the **New Method** button () in a ribbon toolbar or right-click the **Methods** node () in the **Application Builder** window and select **New Method**.

The new **Method** node appears under the **Methods** node in the Application Builder tree and contains an *application method*. Such methods are available globally for use in all application methods and form objects. It is also possible to create *local methods* in some of the form objects in a form. Local methods are not accessible or visible outside of the objects where they are defined. These local methods connect to events in the form objects, such as when the setting (data) changes for a check box. The **Method** node is used for referring to the method from a command sequence or a form object's event. For use with a model in the Model Builder, you can also create **Model Method** nodes for *model methods*, which work the same as application methods but are used to run in the Model Builder to extend or customize some part of a COMSOL Multiphysics model.

A **Method** node contains COMSOL Multiphysics® code and Java® code, which you can inspect and edit by double-clicking it in the Application Builder window or by right-clicking and selecting **Edit**. When a method is opened, it appears in a method editor window.

The changes made to the code in an editor tab are stored in the Application Builder model when you close the tab or after compiling the code. To compile and check the syntax of the code, click the **Check Syntax** button () in the ribbon toolbar.

The code defines an `ApplicationMethod` Java class. This class has a method called `execute` that the command sequence calls when the application runs it. A typical use case is that you create a **Method** node, write some code in the `execute` method, and link this code to a command sequence that a form object in the application's user interface can trigger. Methods that affect the state of form objects apply immediately.

The default setting is to only display the code contained inside the `execute` method. To display all code, enable the **View all code** check box, which is a preference setting in the **Methods** section of the **Preferences** dialog box.

When the `execute` method is triggered while running the application, `model` is the *model object* of the application. From this method, you can access features of the physics and change their parameters.

## The Application Builder Window

---

The **Application Builder** window is a window that, by default, is placed at the left end of the desktop when you are using the Application Builder. The tree contains nodes for all components (forms, methods, files, and so on) in the application.

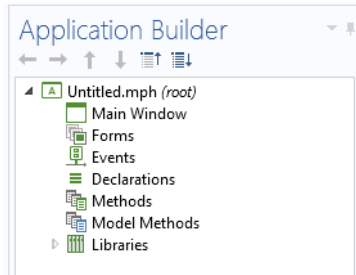


Figure 4-1: The Application Builder window.

Double-clicking a method node in the tree opens its code in an editor tab. If the editor tab associated with the node is already open, the tab is given focus. Right-clicking a node in the tree opens a context menu. This menu contains an **Edit** option, which also opens the editor. The context menu also contains the standard options for a Model Builder tree node, such as **Move Up**, **Move Down**, **Copy**, and **Duplicate**.

Selecting a method node in the tree gives focus to the editor tab that is associated with the node, if the editor tab is open. In the same way, if you click an editor tab to give it focus, the corresponding node is selected in the **Application Builder** window.

## The Method Windows

---

For each method, a method window contains the code for the method. You can enter the code manually, record code from modeling steps, or insert code from the Model Code, Model Expressions, and Language Elements windows. The method editor window uses syntax highlighting, which you can configure in the **Preferences** dialog box (see [Syntax Highlighting and Comments](#)). In the method editor window, you can use the standard keyboard shortcuts and a context menu to select, copy, cut, paste, and delete parts of the code. Triple-click in a line of code or click the line number at the left of the method editor window to select the entire line of code.




## The Method Toolbar







---

The **Method** contextual ribbon toolbar provides access to most of the functionality for creating, editing, and debugging methods. It is available when a method editor window is active. This section briefly describes the buttons in the **Method** toolbar.

### THE MAIN SECTION




This section contains the following buttons for moving to various windows and creating new forms and methods:

- The **Model Builder** button (  ), to switch from the Application Builder to the Model Builder windows and the standard COMSOL Desktop.
- The **New Form** button (  ), to create a new form using the New Form wizard. See [Working with a Form and Using the New Form Wizard](#).
- The **New Method** button (  ), to create a new **Method** node in the model and open its code in a new editor tab. See [The Method Nodes and Method Editor Windows](#).

- The **Data Access** button (  ), to add model-dependent data and properties, as well as application-specific properties that can be modified from a running application through the data access functionality. See [Data Access](#).
- The **Record Method** button (  ), to create a new method by starting a recording session of operations on the embedded model that you can later use as code in that method. When the recording starts, the button changes to the **Stop Recording** button (  ), which you click to end the recording. See [Recording Code](#).
- The **Settings** button (  ), to open or close the **Settings** window.
- The **Preview** button (  ), to show or hide the **Preview** window for a live preview of the forms and methods in the application. In the **Preview** window, you can scroll to get a preview of all forms and methods in the application, which can be useful, for example, if you are working on a method that interacts with a form. To show a preview of a form in the **Preview** window, select a form or method node in the **Application Builder** window.
- The **Editor Tools** button (  ), to show or hide the **Editor Tools** window, where you can choose common COMSOL Multiphysics model operations and insert them into a method or generate form objects based on them. See [Adding Model Code and Form Objects](#).


### THE LIBRARIES SECTION

This section contains the following buttons for including external code and utility classes:

- The **Utility Class** button (  ), to create a new **Utility Class** node in the model and open its code in a new editor tab. See [The Utility Class Node](#).
- The **External Java Library** button (  ), to create a new **External Java Library** node in the model. See [The External Java Library Node](#).
- The **External C Library** button (  ), to create a new **External C Library** node in the model. See [The External C Library Node](#).









### THE EDIT SECTION

The **Edit** section contains the following button:

- The **Revert to Saved** button (  ), to discard the changes made since the method was last saved and revert to the saved version.






### THE CODE SECTION

This section contains the following buttons, which deal with the code more directly:

- The **Language Elements** button (  ), to show or hide the **Language Elements** window, where you can choose and insert language elements into a method. See [Adding Language Elements](#).
- The **Model Expressions** button (  ), to show or hide the **Model Expression** window, where you can choose and insert COMSOL Multiphysics model expressions into a method. See [Adding Model Expressions](#).
- The **Check Syntax** button (  ), to check the syntax for all of the methods that you have created. Syntax errors and warnings in the methods appear in the **Errors and Warnings** window. See [The Errors and Warnings Window](#).
- The **Go to Node** button (  ), to move to the node in the embedded model that corresponds to a model entity in the selected source code. See [Going to Node the Source Code is Mapped To](#).
- The **Record Code** button (  ), to start a recording session of operations on the embedded model that you can later use as code in a method. When the recording starts, the button changes to the **Stop Recording** button (  ), which you click to end the recording. See [Recording Code](#).
- The **Use Shortcut** button (  ), to create a local member field variable to use as a shortcut in an expression of a certain form. See [Using Shortcuts](#).
- The **Create Local Variable** button (  ), to add a local variable and its type declaration in the method editor. See [Creating Local Variables and Their Type Declarations](#).



## THE DEBUG SECTION

This section contains buttons for debugging methods. See [Debugging Methods for Applications](#). The buttons include:

- The **Continue** button (  ), to continue debugging a method after stopping at a breakpoint
- The **Step** button (  ), to step forward in a method
- The **Step Into** button (  ), to step into another method or utility method
- The **Stop** button (  ), to force the current method to stop
- The **Debug Log** button (  ), to open the **Debug Log** window




## THE BREAKPOINTS SECTION

This section contains the following buttons for removing and disabling all breakpoints:

- The **Remove All** button (  ), to remove all breakpoints in all methods
- The **Disable All** button (  ), to disable or enable all breakpoints in all methods








## THE TEST SECTION

This section contains the following tools for testing the application:

- The **Test Application** button (  ), to launch the application in a separate window so that you can test it. See [Testing the Application](#).
- The **Apply Changes** button (  ), to compile and apply code changes to the running application (so-called hot code swap). See [Applying Changes to a Running Application](#).
- The **Test in Web Browser** button (  ), to test the application in a web browser. See [Testing the Application](#).


## THE VIEW SECTION

The **View** section contains the following buttons for rearranging the views in the Application Builder desktop window:

- The **Tile** (  ) and **Move To** (  ) buttons, to rearrange the windows in the Application Builder. Select **Tile Vertically** (  ), **Tile Horizontally** (  ), or **Stack** (  ) from the **Tile** list to arrange the windows. You can also choose a window (  ) from the **Move To** list that is not in focus and move the current window to the same pane (when you have several tiled window panes).
- The **Reset Desktop** button (  ), to reset the desktop layout to the default state.

## *The Method Nodes and Method Editor Windows*

---

The **Method** nodes (  ) contain user-defined methods. There are three types of methods:


- Global methods, which are available globally within an app and, by default, also as methods in the Model Builder.
- Form methods, which are available within a form only and cannot be used in the Model Builder.
- Local methods for events, for example. Such local methods cannot be used in the Model Builder and their inputs and outputs cannot be changed.

To add a new **Method** node, right-click the main **Methods** node, for a global method, or the **Methods** node under a **Form** node, for a form method, and choose **New Method**. You can also choose **Global Method** or **Form Method** from the **New Method** button in the **Method** toolbar (**Form Method** is only active if there is an active form editor). In the **New Method** dialog box that opens, type a label for the method in the **Name** field.

Also, commands to be run, such as for buttons, can be converted to a global method, form method, or, in most cases, local method.

Methods within an application provide functionality connected to buttons, windows, and other components created using the Application Builder.

Methods used in the Model Builder can directly modify the model object represented by the Model Builder in the current session. Such methods can be used, for example, to automate modeling tasks that consist of several manual steps, possibly in connection with settings forms. Methods used in the Model Builder are global methods.

To edit the method, double-click the method node, or right-click it and choose **Edit** () . An editor window opens, where you can edit the code for the method. Depending on the **View all code** preference setting, you see just the method declarations or the full class. In the editor, you can highlight part of the code and right-click to cut, copy, paste, and delete it. You can also use the standard keyboard shortcuts such as Ctrl+C to copy, Ctrl+A to select all code in the editor window, and the Delete button. The code extends a Java class called `ApplicationMethod` that only requires one method with the following signature:



```
public void execute() {
    model.physics("es").feature("ccn1").set("ConstitutiveRelationD", 1,
    model.modelData().getString("dielectricModel"));
}
```


This method can perform any operations available to the COMSOL Multiphysics API. There are a number of methods and members available through the extended class, the most important of which is the model object accessible as the member `model`. The code example above reads a data field named `dielectricModel` from the application's own data (in an Electrostatics interface) and sets it to a parameter of a physics feature in the embedded model.

See [The Method Node](#) for information about the inputs and outputs that you can add in the **Settings** window for method nodes (not local methods).

### *The Utility Class Node*

---

The **Utility Class** node () contains a utility class with methods that you can call from other methods. To add a new **Utility Class** node, right-click the **Libraries** node () and choose **Utility Class** or click the **Utility Class** button in the ribbon toolbar. Enter the name of the utility class in the **Name** field.

A utility class makes it possible to share Java code between methods in your applications and to copy implementations between applications. You can call methods declared in your utility class from any other method in your application. To edit the utility class code, double-click the **Utility Class** node, or right-click it and choose **Edit** () . An editor window opens, where you can edit the code for the utility class. Depending on the **View all code** preference setting, you see just method declarations or the full class.

As an example, consider the following method, which builds all geometries in the application:

```
public static void runAllGeom() {
    model.geom().run();
}
```

The name of the utility class must be the same as the tag of the utility class node. Suppose that the **Utility Class** node is named *util1*. You would call the method above by typing `util1.runAllGeom()`; in any other method.



If you change the name of the utility class, you need to update the source code so that the name of the utility class matches the tag. Otherwise, the code does not compile.

---



If you enable the **View all code** preference setting, the code editor shows that the utility class extends an abstract base class called `ApplicationLanguageBase`. This inheritance makes it possible to access the model object (as exemplified above) and use other convenience methods. You can also remove this inheritance to make classes that



you can instantiate and even extend another utility class or other accessible class. The package of the utility class is `builder`, and you are advised to keep this name.

### *The External Java Library Node*

---

Use the **External Java Library** node (  ) to import compiled Java libraries that you want to include in a method. The Java libraries (JAR files) can be created with any third-party Java development tool. To add a new **External Java Library** node, right-click the **Libraries** node (  ) and choose **External Java Library** or click the **External Java Library** button in the ribbon toolbar. Enter a label to display in the **Label** field.

The **Settings** window for an **External Java Library** node contains the following section:

#### **IMPORT LIBRARY**

In the **Filename** field, enter the name of the Java library file that you want to import, or click **Browse** to locate and choose a Java library file (JAR file). Then click **Import** to import the library into the application. During the import, the JAR file is copied into the application. When you save the application, it contains a copy of the entire JAR file. Also, import statements for packages declared in the JAR file are added to the source code of all existing and future methods and utility classes in the application, and all Java code in the application is recompiled, with the added JAR file on the class path.

When compiling code in the future, the JAR file is on the class path.





To use an updated version of the library, click the **Import** button again. Because the application contains a copy of the library, it is not sufficient to update only the original JAR file.

---

If you delete the **External Java Library** node, the program updates all Java source code to exclude imports defined by the JAR file and recompiles all Java code in the application.

### *The External C Library Node*

---

Use the **External C Library** node (  ) to import a compiled native code library based on, for example, C code that you want to include in a method. To add a new **External C Library** node, right-click the **Libraries** node (  ) and choose **External C Library** or click the **External C Library** button in the ribbon toolbar. Enter a label to display in the **Label** field and a name for the external C library in the **Name** field.

You can import native libraries that are written in any programming language and compiled for the target platform of your application. This can be useful, for example, to access specific hardware (such as measurement devices) attached to the computer or to speed up heavy calculations. The methods you want to call must follow the C language calling convention for the specific platform. Many programming languages can produce methods that fulfill this requirement.

In the **Settings** window, expand the sections corresponding to the platforms you intend to support, and then browse and select the shared libraries you want to import. On the Windows® operating system, the file extension of a shared library is typically `DLL`, on Linux® the extension is typically `so`, and on macOS it is typically `dylib`. Click the **Reload** button if you have changed the file in the file system and want to import the new version into the application. Click the **Edit** button to select a different file to import into the application.

The **Settings** window for an **External C Library** node contains the following sections.

#### **IMPORT LIBRARY FOR WINDOWS, 64-BIT**

This section is expanded by default. Click **Browse** to locate and choose a compiled native code library file (DLL file) for 64-bit Windows®.

## IMPORT LIBRARY FOR LINUX, 64-BIT

Click **Browse** to locate and choose a compiled native code library file (.so file) for 64-bit Linux®.

## IMPORT LIBRARY FOR MACOS, 64-BIT

Click **Browse** to locate and choose a compiled native code library file (dylib file) for 64-bit macOS.

## LOADING

By default, the **Allow loading on unsupported platform** check box is selected so that it is possible to load the shared libraries on platforms that they do not support. On such a platform, an error occurs when you try to access the library from a method. If you clear the check box, you get an error message directly if you try to start the application on an unsupported platform, and the application does not start.

## *Using External C Libraries*

To use a library in your application, you need to call it from a method or utility class. Use the `External external(String libraryTag)` method available in the `ApplicationMethod` class or the `ApplicationLanguageBase` class to return an object of type `External` with the following methods.

TABLE 4-1: METHODS FOR THE EXTERNAL OBJECT

METHOD	DESCRIPTION
<code>long invoke(String method, Object... arguments)</code>	Invokes the named native method in the library with the supplied arguments. Strings are converted to <code>char *</code> . Returns the value returned by the method.
<code>long invokeWideString(String method, Object... arguments)</code>	Invokes the named native method in the library with the supplied arguments. Strings are converted to <code>wchar_t *</code> . Returns the value returned by the method.
<code>void close()</code>	Releases the library and frees resources. If you do not call this method, it is automatically invoked when the external library is not needed any longer.

The syntax could be:

```
length = external("native1").invoke("stringLength", "Foo");
```

which assumes a C function declared as `int stringLength(char *str)`. Alternatively, you could use the following code:

```
External library = external("native");
seven = library.invoke("sum", 3, 4);
two = library.invoke("sum", 3, -1);
library.close();
```

which assumes a C function declared as `int sum(int a, int b)`. The latter syntax is more efficient if you need to make several calls to the same library.

## SUPPORTED ARGUMENT TYPES

Methods can have up to six arguments and may (optionally) return an integer (or native pointer) value, which is a 64-bit value on 64-bit platforms. In Java, the return value is always a long.

Not all argument types are supported, and some restrictions apply. In particular, float arguments are converted to double arguments. The native function must not be declared with float arguments. If you need to transfer data of the float type, you can instead use a float array, which is supported.

TABLE 4-2: SUPPORTED ARGUMENT TYPES

JAVA TYPE	C/C++ TYPE	REMARKS
boolean	bool	
byte	char	
char	wchar_t	System dependent. 32 bits on Linux and macOS, 16 bits on Windows.
short	short	
int	int	
long	long long	
String	const char * or const wchar_t *	The version you get on the C side depends on which version of the invoke interface you use. The string is null terminated and must not be modified. Compare the remark for wchar_t above.
float	double	The argument is automatically extended to double precision.
double	double	
boolean[]	bool *	
byte[]	char *	
short[]	short *	
int[]	int *	
int[][]	int **	
long[]	long long *	No restriction on any platform.
String[]	char ** or wchar_t **	String[] arguments can be used to transfer strings back to Java. Compare the remarks for String above.
float[]	float *	
double[]	double *	
double[][]	double **	

For array types, the external function can modify the values in the supplied array. The updated values are transferred back to the Java program. Note that it is not possible for the callee to change the size of a transferred array. The caller and callee are responsible for communicating the size, by using an extra argument or by a convention. If the callee writes or reads memory outside the allocated buffer, the program behavior is undefined but may include an abrupt program exit. Note that for the double array types, only values may be modified, not any pointers.

For arguments of type `String[]`, the program can modify a pointer in the array of pointers to point to a new null-terminated string, which is transferred back to Java. The maximal allowed length of a string is 65,535 characters. Use a byte array to transfer larger amounts of data. It is possible to return a null value to Java by setting the corresponding element in the string array to 0.

More complex types need to be serialized on the Java side to a byte array, which can be deserialized by the callee.

The external function can return an integer (`int`) or a pointer to some internal object (`void *`). In Java, this value is returned as a long value, which is sufficiently wide to hold a 64-bit memory address. If you need a `String` or a

`double` as the return value, you must instead pass an array of length one and the right type, which can be filled in by the external function.



As the app creator, you need to ensure that the right type of arguments are given in the right order. There are no checks and making a mistake can easily crash the program or cause undefined behavior. In particular, when working with integer and floating-point arguments, you need to take extra care. A function declared as `sum(int a, int b)` must be invoked by `invoke("sum", 1, 2)`, whereas a function declared as `sum(double a, double b)` must be invoked by `invoke("sum", 1.0, 2.0)`.

### Debugging

It may be possible to use a native code debugger (like Microsoft® Visual Studio®) when working with applications. First, you need to build the library with debug symbols. Then, import this library into the application, and attach the debugger to the COMSOL Multiphysics process. Put a breakpoint in the native function that you want to debug. Run or test run the application in COMSOL Multiphysics. When the software loads the external library, the debugger should be able to match the source code where you put your breakpoint with the loaded library and to break when the function is called. Refer to the documentation of your native debugging system for further details.

### Notes and Tips For Using External Libraries

- Be very careful when accessing memory from your native code. You can easily corrupt the memory of the entire application, which causes undefined behavior or crashes.
- You need to provide separate libraries for each platform you intend to support with the application, but it is not necessary for a specific application to support all platforms supported by COMSOL Server. Currently, the supported platforms are Windows 64-bit, Linux 64-bit, and macOS 64-bit, in all cases using the AMD64 architectures.
- Most compilers create shared libraries that depend on a runtime environment provided by other shared libraries that are distributed with the compiler. You must make sure that the appropriate runtime environment is installed on the intended target computers.
- If the native code library you want to use is too complex to fit inside a single shared library (DLL, so, or dylib), you need to deploy the library separately on the target computer (using a custom installer). Then, the application will contain a small wrapper library that calls the complex library. Technically, it may be possible to include the entire installer of the complex library inside a large but simple library with a method that installs the complex library, provided no system administrator rights are required.

### EXAMPLE OF AN EXTERNAL FUNCTION

The following steps show an example of how to create and import a library and then call it in methods.

#### I Create the library.

The source code below defines a trivial external function written in C++, which you want to call. The `#ifdef` statement is intended to make the source code cross-platform compatible, but writing a shared library typically involves compiler-specific settings, so you may need to consult the documentation of your compiler. The header file, `test.h`, is as follows:

```
// test.h : Declares the exported function
#ifdef _MSC_VER
    #define TESTDLL_API __declspec(dllexport)
#else
    #define TESTDLL_API __attribute__((__visibility__("default")))
#endif
extern "C" {
    TESTDLL_API int testSum(int a, int b);
}
```

```
}
```

The function definition, `test.cpp`, looks like:

```
// test.cpp : Defines the exported function for the library
#include "test.h"

// This is an example of an exported function.
TESTDLL_API int testSum(int a, int b) {
    return a + b;
}
```

Using the gcc compiler, type something like this in the command shell:

```
gcc -shared -o test.so -fPIC test.cpp
```

If you are using a graphical C++ build environment, like Microsoft Visual Studio, it is probably easiest to insert the code above into a DLL project.

## 2 Import the library.

Add an **External C Library** node, with tag `native1`, and import the library created in Step 1 for the right platform.

## 3 Calling methods:

Add a **Method** node and insert the following code into the Execute Method body:

```
long sum = external("native1").invoke("testSum", 1, 2);
alert("1 + 2 = " + sum);
```

Add a button to the form, and choose the method added above as the command to execute. If you are building the application on the same platform the library is built for, you can click **Test Application** to test the method.

Otherwise, save the application and run it in a COMSOL Multiphysics session on the correct platform. In any case, when you press the button in the application window, you should see a dialog stating a truism.

## *File Schemes and File Handling*


---

### TRANSFERRING FILES FROM SERVER TO CLIENT

To transfer files from a COMSOL Server to the client application, do not add any **File** node under **Declarations**, and follow these steps:

1 Write the data to `temp:///output.data`, for example.

2 Add `temp:///output.data` as the argument to a `downloadtoclient` action.

For assistance when defining the argument for the `downloadtoclient` action, click the **Edit Argument** button () to open an **Edit Argument** dialog box (see [Editing Initial Values and Arguments in Declarations and Command Sequences](#)), where you can choose one of the file schemes below from the **File scheme** list. You then specify the filename (such as `output.data`) in the **Filename** field. For the `embedded:///` file scheme, choose an existing file in the application from the list under **Choose an application file resource**. Click **OK** to close the dialog box and fill the **Arguments** field with the selected file scheme and filename.

### FILE SCHEMES

Anywhere in the model or application where a file path is given that would normally refer to a file on the client file system, you can instead use a scheme syntax such as `<scheme>:///...` to refer to a file that should be taken from

somewhere else on the server. The following table includes the available schemes and where the files exist when running from the Application Builder and COMSOL Server.

TABLE 4-3: FILE SCHEMES

SCHEME	REFERS TO	DEFAULT PATH IN APPLICATION BUILDER	DEFAULT PATH ON COMSOL SERVER
embedded:///	Files embedded in the model using file libraries (read only).	Inside the MPH-file	Inside the MPH-file
upload:///	Files that are uploaded to file declarations.	user:/// or temp:/// depending on target	user:/// or temp:/// depending on target
temp:///	Files in a random temporary directory, which is unique for each started application instance.	Random subdirectory of %TEMP% or /tmp	Random subdirectory of .comsol/v50server/service/users/[username]
common:///	Files in a directory shared by all users and applications.	In .comsol/v54/applications/files/common	In .comsol/v54server/applications/files/common
user:///	Files in a directory shared by all applications for the current user.	In .comsol/v54/applications/files/user	In .comsol/v54server/applications/files/users/[username]

Files saved to the `temp` scheme do not persist between multiple runs of the same application, even for the same user. Files saved to the `user` scheme persist and can be accessed by the same user, even for other applications. Files saved to the `common` scheme persist and can be accessed by all users for all applications.

You can modify the default locations separately using the following preferences settings.

TABLE 4-4: DEFAULT LOCATION PREFERENCES

SCHEME	APPLICATION BUILDER PREFERENCE	COMSOL SERVER PREFERENCE
temp:///	Files>Folder for temporary files	Files>Folder for temporary files
common:///	Files>Folder for common files	Files>Folder for application library files
user:///	Files>Folder for user files	Files>Folder for application user files

## URI SYNTAX

File scheme strings are uniform resource identifiers (URIs) as defined by RFC 2396 (<http://www.ietf.org/rfc/rfc2396.txt>). In addition to what is allowed by RFC 2396, the file schemes allow any nonreserved Unicode characters in the path segments. This means that spaces do not need to be escaped in file and directory names. Sequences of escaped octets (for example, %20) are decoded as UTF-8. Currently, the file schemes do not use the authority component; that is, they are always on the form `<scheme>:///<path>` with three initial slashes.

### *Getting Files to and from the Client File System*

The Application Builder provides some ways of getting files to and from the client file system when running an application. The following ways of getting files work both when running the web client and when running the native client.

- Use the file import form object (see [File Import](#)) to ask the user for a file. The user browses to a file on the client file system that is then uploaded to the server and available to the application and its methods. This method can be used, for example, to provide a CAD file or experimental data from the user at run time.

- You can call the `fileOpen` function from any application method. It picks any file from the server produced by a method, the model, or embedded with the application and opens it using the associated application on the client. This method can be used, for example, to open a PDF document on the client or show a text file or an image exported from the model on the client.
- It is also possible to call the `fileSaveAs` function from any application method in a way that is similar to `fileOpen`. It takes any file from the server and presents a **Save As** dialog box where the user can browse to a client location to save the file. This method is similar to downloading files from a link within a web browser.
- The Save Application As and Save Application on Server As commands are available in the command sequence tree for buttons and menu items in a form. These commands present a **Save As** dialog box where the user can specify a client path where the entire application is saved. If the application is run on COMSOL Server, these commands save the current state as a new application in the COMSOL Server Application Library.

# Creating Methods

The following sections describe the tools for writing code and creating methods:

- [Syntax Highlighting and Comments](#)
- [Code Completion and Tooltip Help](#)
- [Code Folding](#)
- [Adding Language Elements](#)
- [Adding Model Expressions](#)
- [Adding Model Code and Form Objects](#)
- [Going to Node the Source Code is Mapped To](#)
- [Recording Code](#)
- [Using Shortcuts](#)
- [Calling Other Methods Directly](#)
- [Using Properties Defined in Declarations as Variables](#)
- [Searching and Finding Text](#)
- [Indentation and Whitespace Formatting](#)
- [Brace Matching](#)

See also the *COMSOL Multiphysics Programming Reference Manual* for information about the built-in methods for working with the model object available in COMSOL Multiphysics (listed there except for physics interfaces).

## *Syntax Highlighting and Comments*

---

Different language elements in the methods' code are displayed using different styles. As an example, consider the following code snippet.

```
Code3 ×  
1 for (int i = 0; i < 10; i++) {  
2     model.result().numerical("eval").set("expr", "var" + i);  
3     model.result().numerical("eval").appendResult();  
4 }
```

The code snippet include three styles:

- Keywords (`for`, `int`, and so on) appear using a blue boldface font.
- String literals appear using a red font.
- The remainder of the code appears using a black font.

You can configure the syntax highlighting theme in the **Preferences** dialog box. Choose **File>Preferences>Methods** to see the preferences that are specific for methods. Under **Syntax highlighting**, the **Theme** list contains two predefined themes: **Modern** (the default) and **Classic**. Choose **User defined** to define a syntax highlighting mode where the colors can be assigned to individual parts of the code, such as keywords, numbers, strings, and language elements. Click the color chooser for each part of the code to choose a color from the color palette that opens. Click the **Define Custom Colors** button to add additional colors that are defined using RGB values.



## ADDING COMMENTS

You can add comments to the code by preceding the comment by `//`. You can toggle comments on and off for a selected line using the shortcut `Ctrl+7` or by right-clicking in the method window and choosing  **Toggle Comment**.

## Code Completion and Tooltip Help

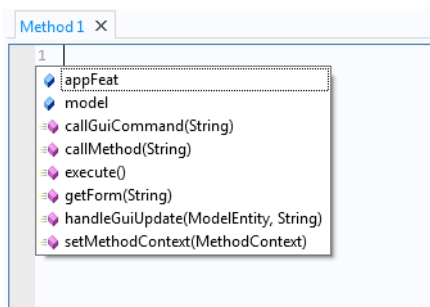
---

### CODE COMPLETION

When typing Java code in a method editor window, it is possible to request suggestions for completions of partial code. The list of possible completions are shown in a separate completion list that opens. In some situations, detailed information appears in a separate window when an entry is selected in the list. Code completion can always be requested with the keyboard shortcut `Ctrl+Space` (or `Ctrl+/>`). It appears automatically when you type a period because you typically want to choose between the available methods when accessing a field.

The Application Builder supports the following types of completion suggestions.

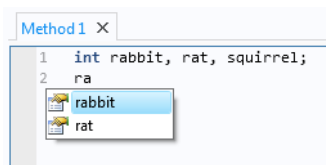
*New statements:* A simple example is hitting `Ctrl+Space` without having entered anything, as shown below.



The completion list contains the member fields (for example, `appFeat` or `model`) and member methods (for example, `callGuiCommand`) that can be called. You can select a completion in two ways:

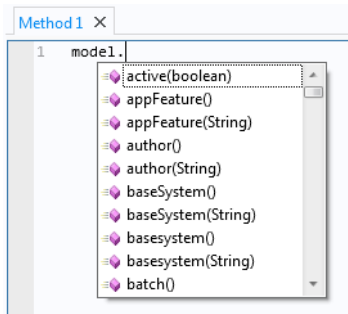
- Using the arrow keys to select an entry in the list and pressing the `Tab` key to confirm the selection.
- Typing some text. The list is automatically updated so that only the completions that begin with the text are kept. For example, if you type `appF`, then only `appFeat` remains. You can then press `Tab` to confirm the selection.

*Partial statements:* If you enter the beginning of a variable, field, or method name and press `Ctrl+Space`, the suggested completions are shown.

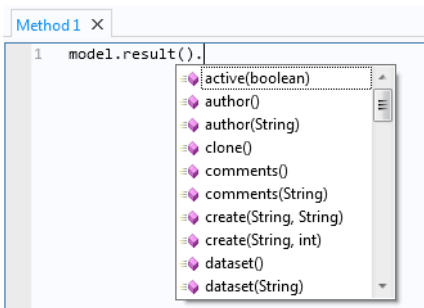


Only variables that match the prefix `ra` are shown. This example shows that local variables also appear in the completion suggestions.

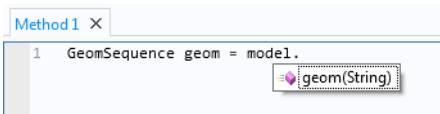
*Class members:* If you enter a variable or expression that resolves to a known Java class for the model object, then you can get completion suggestions for the members in the class, as shown below.



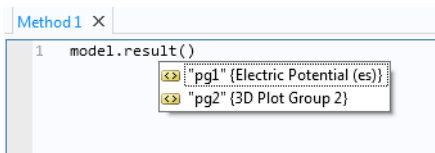
This completion also works for chains of calls, as shown below.



*Class members in assignments and declarations:* In assignments and variable declarations, the expected type can be used to filter the list of completions. Only completions that have the same type declared on the left side of the assignment statement are shown.

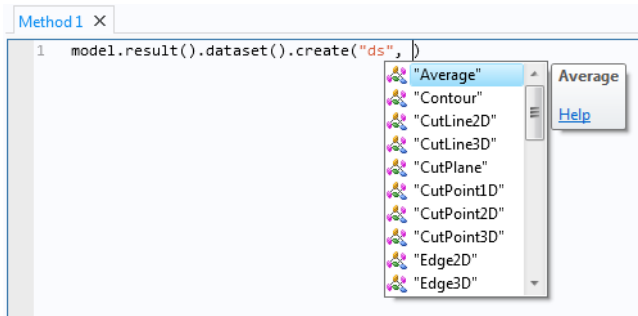


*Tags in the model object:* When the expression to complete can be resolved to a concrete entity in the embedded model, completions are available for the methods in the model object API that take a first argument that is a tag in a list, as shown.

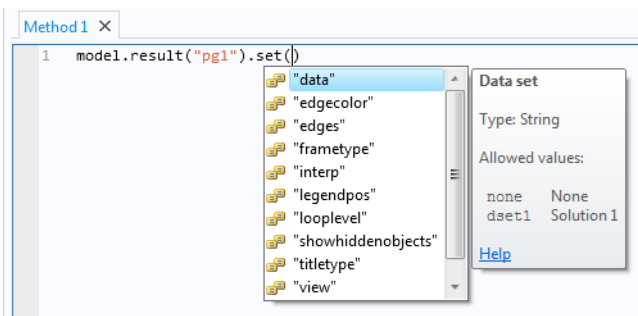


The list above comes from an application whose embedded model contains two plot groups (pg1 and pg2). The completion list contains both the tags and the names of the corresponding plot groups. To transfer only the tag to the code, choose a completion in the list and then press Tab.

*Types in the model object when creating new entities:* When the expression to complete can be resolved to a concrete entity in the embedded model, completions are available for operation types in the model object when creating a new node.

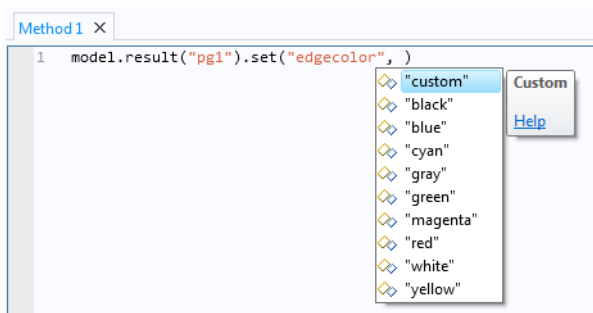


*Property names:* When the expression to complete can be resolved to a concrete entity in the embedded model, completions are available for property names used as first arguments to the `get` and `set` operations. These are available for most features in the model object.



The example above shows the properties that are available for the plot group `pg1` in the embedded model. If you select an entry in the list, you see the property's description and its data type. When available, you may also see a link to the documentation and a set of allowed values. The allowed values appear if the set of allowed values is a finite array of strings.

*Property values:* When the expression to complete can be resolved to a concrete entity in the embedded model, completions are available for property values used as second arguments in the `set` operations that are available for most features in the model object.



If you select an entry in the list, you see the property value's description and a link to the documentation, if available.

## TOOLTIPS FOR CODE IN THE METHOD EDITOR

When hovering over different parts of the code for a method, tooltips appear to provide information about that part of the code. The following tooltips appear for different parts of the code:

### Property Names

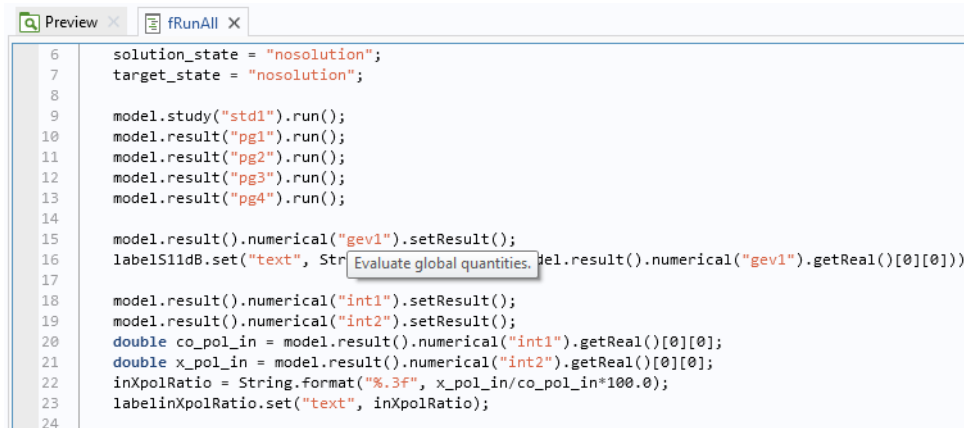
When you hover over a property name for a model or application object property, you get information about the property, similar to what you see for the property when code completion is used. For example, if you hover over `numerical` in `model.result().numerical("gev1").setResult();`, you see the following tooltip.



```
6 solution_state = "nosolution";
7 target_state = "nosolution";
8
9 model.study("std1").run();
10 model.result("pg1").run();
11 model.result("pg2").run();
12 model.result("pg3").run();
13 model.result("pg4").run();
14
15 model.result().numerical("gev1").setResult();
16 labelS11dB.set("numerical(String tag) model.result().numerical("gev1").getReal()[0][0]);
17
18 model.result().
19 model.result().
20 double co_pol_in = model.result().numerical("int1").getReal()[0][0];
21 double x_pol_in = model.result().numerical("int2").getReal()[0][0];
22 inXpolRatio = String.format("%.3f", x_pol_in/co_pol_in*100.0);
23 labelinXpolRatio.set("text", inXpolRatio);
24
25 model.result().numerical("int3").setResult();
26 model.result().numerical("int4").setResult();
27 double co_pol_out = model.result().numerical("int3").getReal()[0][0];
28 double x_pol_out = model.result().numerical("int4").getReal()[0][0];
29 outXpolRatio = String.format("%.3f", x_pol_out/co_pol_out*100.0);
30 labeloutXpolRatio.set("text", outXpolRatio);
31
```

### Model Entities

If you hover over the same property's model entity name, "gev1" in this case, you see the following tooltip with information about the purpose of the model entity.



```
6 solution_state = "nosolution";
7 target_state = "nosolution";
8
9 model.study("std1").run();
10 model.result("pg1").run();
11 model.result("pg2").run();
12 model.result("pg3").run();
13 model.result("pg4").run();
14
15 model.result().numerical("gev1").setResult();
16 labelS11dB.set("text", String.format("%.3f", model.result().numerical("gev1").getReal()[0][0]));
17
18 model.result().numerical("int1").setResult();
19 model.result().numerical("int2").setResult();
20 double co_pol_in = model.result().numerical("int1").getReal()[0][0];
21 double x_pol_in = model.result().numerical("int2").getReal()[0][0];
22 inXpolRatio = String.format("%.3f", x_pol_in/co_pol_in*100.0);
23 labelinXpolRatio.set("text", inXpolRatio);
24
```

## Declarations

For variables defined under **Declarations** in the Application Builder tree, the variable's description appears as the tooltip when you hover above them. In this case, it is the string variable `inXpolRatio`.



```
6 solution_state = "nosolution";
7 target_state = "nosolution";
8
9 model.study("std1").run();
10 model.result("pg1").run();
11 model.result("pg2").run();
12 model.result("pg3").run();
13 model.result("pg4").run();
14
15 model.result().numerical("gev1").setResult();
16 labelS11dB.set("text", String.format("%.3f", model.result().numerical("gev1").getReal()[0][0]));
17
18 model.result().numerical("int1").setResult();
19 model.result().numerical("int2").setResult();
20 double co_pol_in = model.result().numerical("int1").getReal()[0][0];
21 double x_pol_in = model.result().numerical("int2").getReal()[0][0];
22 inXpolRatio = String.format("%.3f", x_pol_in/co_pol_in*100.0);
23 Input waveguide feed cross polarization ratio percentage
24
25 model.result().numerical("int3").setResult();
26 model.result().numerical("int4").setResult();
27 double co_pol_out = model.result().numerical("int3").getReal()[0][0];
28 double x_pol_out = model.result().numerical("int4").getReal()[0][0];
29 outXpolRatio = String.format("%.3f", x_pol_out/co_pol_out*100.0);
30 labeloutXpolRatio.set("text", outXpolRatio);
```

## Shortcuts

For shortcuts to form objects, also defined under **Declarations**, the description and the corresponding Java code appear in the tooltip. In this case, it is for the shortcut with the name `labelinXpolRatio`.



```
6 solution_state = "nosolution";
7 target_state = "nosolution";
8
9 model.study("std1").run();
10 model.result("pg1").run();
11 model.result("pg2").run();
12 model.result("pg3").run();
13 model.result("pg4").run();
14
15 model.result().numerical("gev1").setResult();
16 labelS11dB.set("text", String.format("%.3f", model.result().numerical("gev1").getReal()[0][0]));
17
18 model.result().numerical("int1").setResult();
19 model.result().numerical("int2").setResult();
20 double co_pol_in = model.result().numerical("int1").getReal()[0][0];
21 double x_pol_in = model.result().numerical("int2").getReal()[0][0];
22 inXpolRatio = String.format("%.3f", x_pol_in/co_pol_in*100.0);
23 labelinXpolRatio.set("text", inXpolRatio);
24 Text label
25 Shortcut to Text label
26 app.form("mainForm").formObject("labelinXpolRatio");
27 double co_pol_out = model.result().numerical("int3").getReal()[0][0];
28 double x_pol_out = model.result().numerical("int4").getReal()[0][0];
29 outXpolRatio = String.format("%.3f", x_pol_out/co_pol_out*100.0);
30 labeloutXpolRatio.set("text", outXpolRatio);
31
```

## PREFERENCES FOR CODE GENERATION AND COMPLETION

In the **Methods** part of the **Preferences** dialog box, you can specify options for code generation and completion.

The **Close brackets automatically** check box is selected by default. The software then automatically inserts a corresponding closing bracket if you type a {, [, or ( character. Clear the check box to disable the addition of closing brackets.

The **Generate compact code using 'with' statements** check box is selected by default. The generated code uses with statements to set a target to use with the coming calls to `set()`, `setIndex()`, `getString()`, and so on, which makes the code more compact. Clear the check box if you prefer to use the full code without the use of with statements.

## Code Folding

By default, the methods' editor windows provide code folding, which makes it possible to selectively hide and display sections of the code in the editor. You can use code folding to manage methods that include a lot of code by viewing only those sections of the code that are relevant at any given time.

Click the – (minus) button to hide (fold) that code segment or the + (plus) button to display the code segment. When folded, put the cursor at the box with an ellipsis (...). A tooltip then displays the hidden code.

```

b_compute X
16 L1=Math.max(L1,minlength);
17 int k=2;
18 while(k<MAXITERATIONS && Math.abs(f1)>fqtol ){
19     f2 = f1;
20     fq = frequency(L1);
21     f1 = fq-targetfq;
22     carry = L1;
23     L1=L1-f1*((L1-L2)/(f1-f2));
24     L2 = carry;
25     L1=Math.max(L1,minlength);
26     k=k+1;
27     setProgress(k*100/MAXITERATIONS);
28 }
29 L1=Math.round(L1*100)/100.00; // we won't get more than 2 decimal accuracy (mesh limits the accuracy rather than the secant method)
30 model.param().set("L1",L1);
31 if(Math.abs(f1)>fqtol) {
32     error("Computation terminated after "+toString(MAXITERATIONS)+" iterations. Frequency diff: "+toString(Math.abs(f1)));
33 }
34 }
35 else { // "Find prong length" check box is cleared in main form
36     startProgress(false, "Computing frequency.");
37     setProgress(25);
38     fq = frequency(model.param().evaluate("L1"));
39     setProgress(75);
40 }
41 }
  
```

Figure 4-2: A method editor window before folding some of the code segments.


```

b_compute X
16 L1=Math.max(L1,minlength);
17 int k=2;
18 while(k<MAXITERATIONS && Math.abs(f1)>fqtol )...
19 L1=Math.round(L1*100)/100.00; // we won't get more than 2 decimal accuracy (mesh limits the accuracy rather than the secant method)
20 model.param().set("L1",L1);
21 if(Math.abs(f1)>fqtol) ...
22 }
23 else ...
41 }
42 with(model.result("pg1"));
43     set("looplevel", new String[]{"7"}); // The first real eigenfrequency always the 7th computed eigenfrequency in solid mechanics
44 endwith();
45 useGraphics(model.result("pg1"), "graphics1");
46 zoomExtents("graphics1");
47 setProgress(100);
48 play_sound();
49 closeProgress();
  
```

Figure 4-3: The same method after folding some of the code segments.

To turn off code folding, open the **Preferences** dialog box. On the **Methods** page, clear the **Enable code folding** check box under **Settings**.

## Adding Language Elements

The **Language Elements** window includes common language constructs and templates for performing common operations, such as creating arrays, or for blocks, which you can insert into a method. You can also add string utilities, variable declarations and conversions, progress information and error messages, file handling, user interface components, code for sending emails, and other useful language elements. Double-click to insert an expression, or right-click and choose **Insert Template** (  ). Use the search field at the top to filter the list of available constructs and templates.

Some actions use the current editor selection in the template — for example, putting the current selection within the **Multiline comment** template. The part of the template that you are most likely to change is selected or the cursor is positioned there.

## Adding Model Expressions

The **Model Expressions** window includes COMSOL Multiphysics expressions that you can insert into a method editor window for use with results or equation features in a method. For example, you can insert `x` (the  $x$ -coordinate) in `model.result().numerical("gev1").set("expr", "x")`. Double-click an expression to insert it, or right-click and choose **Insert Expression**. The program inserts the expression at the cursor in the method editor window.

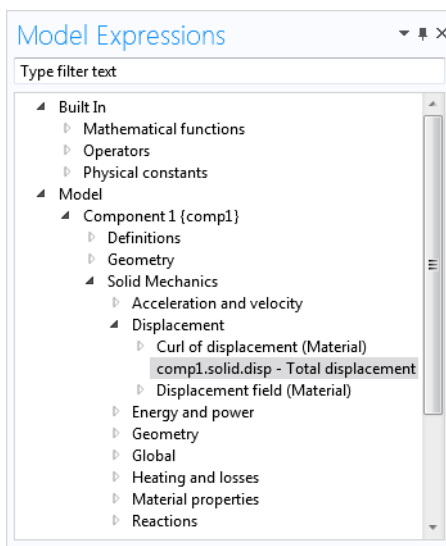


Figure 4-4: The full view of the available model expressions.

Use the search field at the top of the window to filter the list of expressions. For example, the following figure shows the list of expressions where `si` occurs:

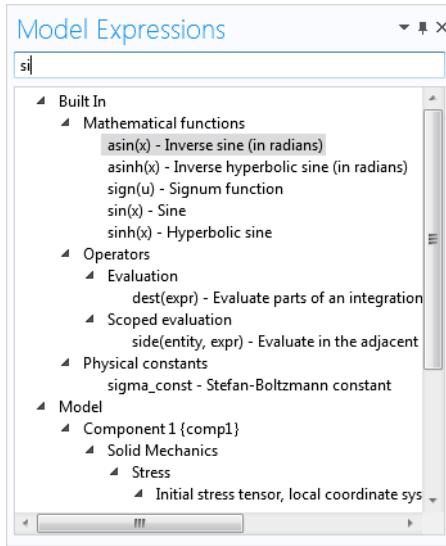


Figure 4-5: A filtered view of the available model expressions.

The inserted expressions and functions are strings enclosed within quotation marks (for example, `"sigma_const"`), unless the insertion point is already within a string. The reason for the string format is to make it clear that the inserted expressions are model expressions and not code.

### *Adding Model Code and Form Objects*

---

The **Editor Tools** window provides code for common model operations (for example, setting all nondefault properties of a feature) and options to create applicable form objects corresponding to a model operation.

#### **EDITOR TOOLS**

When you work in a method editor, you can insert the code at the cursor position in a method editor window. Right-click on a tree node in the **Editor Tools** window, choose from the actions applicable to that node in the model, click one of the corresponding buttons underneath the tree, or double-click or press Enter to insert the top action on the tree node's context menu.



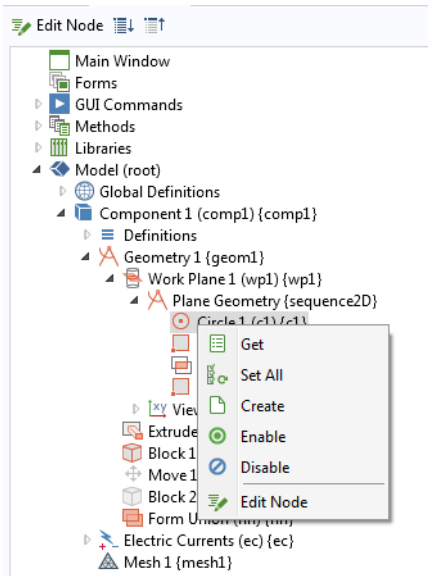











Figure 4-6: Use the context menu to insert code into the current method.


The following actions are available in the method editor (not all actions are applicable to all nodes):




- **Get** (  ), to insert code to get the value of a parameter, property, or filename (for example, `model.param().get("w")` or `model.component("comp1").geom("geom1").feature("r1").getDouble("rot")`), or to insert a reference to the feature (for example, `model.geom("geom1").feature("r1")`).
- **Set** (  ), to insert code to set the value of a parameter, property, or filename (for example, `model.param().set("w", 0)`; or `model.component("comp1").geom("geom1").feature("r1").set("rot", "0");`).
- **Set All** (  ), to insert code to set all nondefault properties of a feature.
- **Create** (  ), to insert code to create the feature (for example, `model.component("comp1").geom("geom1").create("r1", "Rectangle");`).
- **Run** (  ), to insert code to run the feature (for example, `model.sol("sol1").run();`).
- **Enable** (  ) or **Disable** (  ), to insert code to enable or disable the feature (for example, `model.component("comp1").geom("geom1").feature("r1").active(true)`; to enable that feature).
- **Edit Node** (  ), to select that node under the **Model** branch in the main desktop's **Application Builder** window and open its **Settings** window for editing. You can also click the **Edit Node** button (  ) at the top of the **Editor Tools** window.

#### FORM EDITOR TOOLS



When you work in a form editor, you can insert form objects corresponding to an action or display for a selected item in the tree. Right-click on a tree node and choose among the different form objects or collection of form objects, click one of the corresponding buttons underneath the tree, or double-click to insert the top option on the tree node's context menu.

The following actions are available in the form editor (not all actions are applicable to all nodes):

- **Input** (  ), to insert a Text Label, Input Field, and (if applicable) Unit object to display the name and an input field for a model parameter or a variable under Declarations.

- **Output** (  ), to insert a Text Label and Data Display object to display the name and value of a model parameter or variable, or for a variable under Declarations.
- **Graphics** (  ), to insert a Graphics object for a plot group, geometry, or mesh.
- **Button** (  ), to insert a Button object for a file or view command or to create a report, for example.

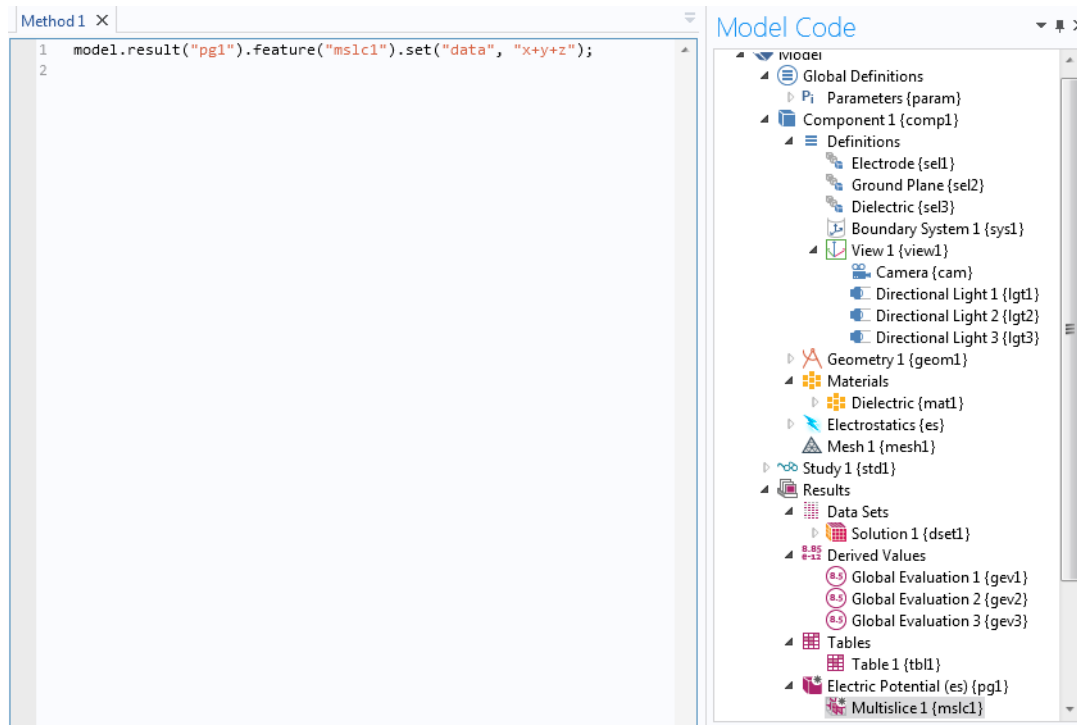
### *Going to Node the Source Code is Mapped To*

If you click the **Go to Node** button (  ) in the **Method** ribbon toolbar, then the source code at the cursor is mapped to an entity in the embedded model. If possible, the corresponding entity is shown in the **Editor Tools** window. This window is opened automatically if it is not already open. You can also right-click on the code and select **Go to Node**  when you are positioned on a part of the code that refers to the embedded model.

As an example, suppose that you enter the following code:



```
Method1 x
1 model.result("pg1").feature("mslc1").set("data", "x+y+z");
2
```

Position the cursor somewhere on `feature` and click **Go to Node**. You get the following effect:

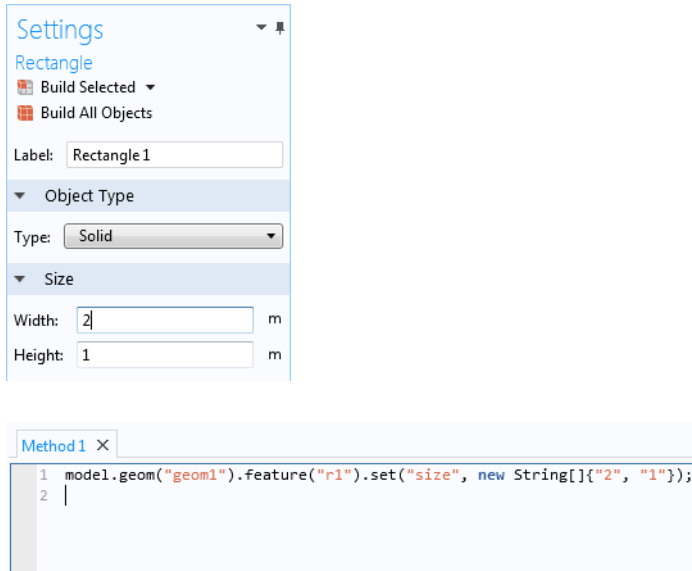


If you move the cursor to somewhere on `result` and click **Go to Node**, then the Electric Potential plot group is selected instead in the **Editor Tools** window.

### *Recording Code*



This action makes it possible to insert code based on modifications of the embedded model. Put the insertion point where you want to insert the actions, click **Record Code** (  ), perform the actions in the **Model Builder** window, and then click **Stop Recording** (  ) in the **Method** toolbar's **Code** section. The recorded code then appears directly in the

method editor window. For example, if you update the size settings for a Rectangle geometry object, as in the following figure the recorded code for that action appears in the method editor window.




When recording code, a red dot on a **Method** node indicates the method where the recorded code appears and a red frame surrounds the method editor window for that method.

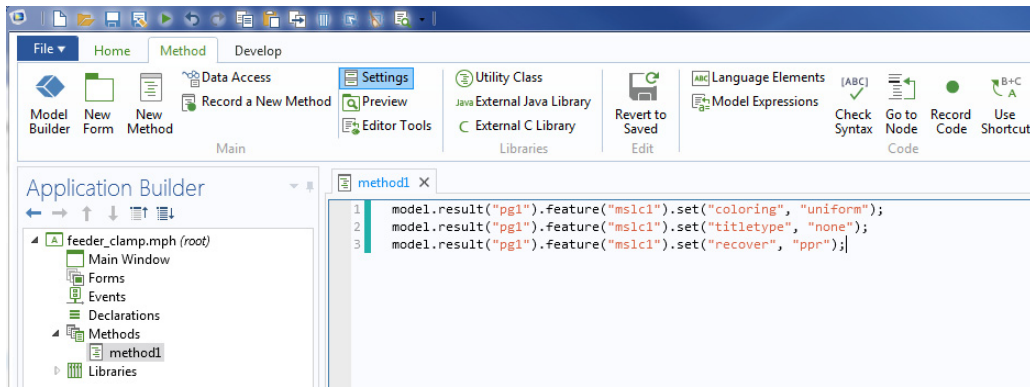
### RECORDING A NEW METHOD

For convenience, there is also an action in the **Home** ribbon toolbar (as well as in the **Method** toolbar), in the **Main** section, for creating a new method and starting the recording. Click **Record Method** (  ); enter a name for the new method in the **Name** field and choose a type of method — **Application method** (the default) or **Model method** — from the **Method type** list of the **Record Method** dialog box; perform the actions that you want to record; and then click **Stop Recording** (  ). In principle, you do not need to see any code to record a method. If a method editor window is open, the recorded code appears directly, just as when using Record Code above. When you record a new method, a red frame surrounds the Model Builder and Application Builder windows in the COMSOL Desktop to indicate that code recording is active.

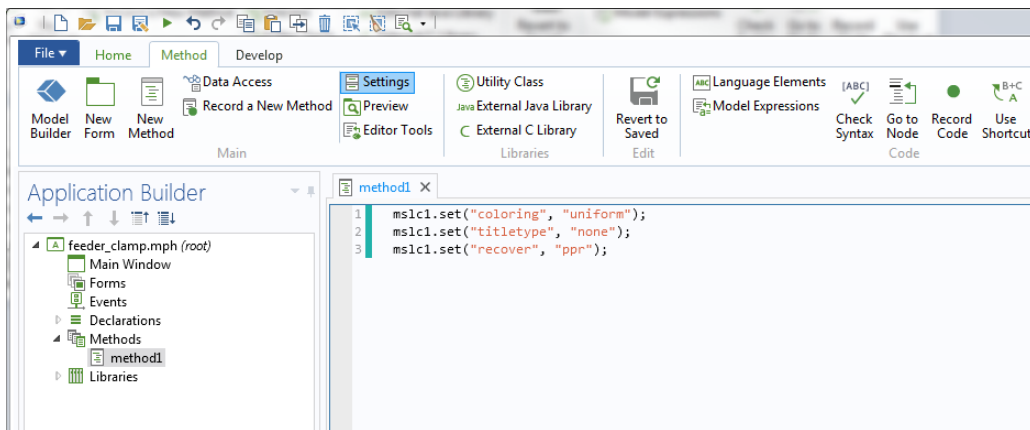
### *Using Shortcuts*

If you click **Use Shortcut** (  ) in the **Method** ribbon toolbar (or press Ctrl+K), the method editor checks the source code at the cursor. If it corresponds to an expression of a certain form, a **Use Shortcut** dialog box appears, where you can enter the name of the shortcut in the **Name** field. Click **OK** to add the shortcut to the list in the **Settings** window of the **Shortcuts** node and to replace the source code that is represents with that shortcut. If there is more than one possible expression in the selected source code, a **Select Expression** dialog box appears, where you can select the part to extract from the **Extract** list. The expression must correspond to a model entity that is currently included in the embedded model.

For example, enter the code in the following figure and position the cursor on the first occurrence of `feature`.



Click the **Use Shortcut** button to transform the source code into what is shown in the following figure.




The expression corresponding to the `feature` part of the source code has been replaced by the shortcut `ms1c1`, which is defined as `Results/pg1/ms1c1`.

If you click the **Use Shortcut** button when the cursor is not at an expression of a supported type, an error message appears. The expression form that is supported typically occurs when the model object API is used; for example, a sequence of method invocations on an object such as

```
<variable name> [ . <method name> (<method arguments (optional)>) ]
```

### *Creating Local Variables and Their Type Declarations*

If you have declared a local variable but does not know its type, you can click the **Create Local Variable** button () (or press `Ctrl+1`) to insert the correct type declaration. If the local variable is also missing, a unique local variable is created along with its type declaration. For example, if you type

```
x = model.component("comp1").geom();
```

and then click the **Create Local Variable** button, the variable's type declaration is added:

```
GeomList x = model.component("comp1").geom();
```

If you only added `model.component("comp1").geom()`, a unique local variable name is also added:

```
GeomList var3 = model.component("comp1").geom();
```

## Calling Other Methods Directly

---

Suppose that the application contains the methods `method1` and `method2`. You can call a method by using its name directly in the Java code. This is equivalent to calling `callMethod()`, which is declared in the superclass `ApplicationMethod` (for this example, it is `method16`):


```
package builder;

import com.comsol.api.*;
import com.comsol.model.*;

public class method16 extends ApplicationMethod {

    public void execute() {
        // The following two lines are equivalent:
        method2();
        callMethod("method2");
    }

}
```

To open the other method's editor window, **Ctrl+Alt+double-click** on the method name in the code (`method2`, for example), or right-click and choose **Go to Method** ()

## Using Properties Defined in Declarations as Variables

---

For each node under **Declarations** that corresponds to a data type (String, Boolean, and so on), corresponding variables are available for use in the Java code. With some exceptions, they behave like ordinary Java variables. As an example, suppose that there is a declaration of the string property `foo` in a node under **Declarations**. The following code assigns it a value (for this example, it is `method16`):

```
package builder;

import com.comsol.api.*;
import com.comsol.model.*;

public class method16 extends ApplicationMethod {

    public void execute() {
        foo = "bar";
    }

}
```

## Searching and Finding Text

---


Press **Ctrl+F** to open a **Find** tool that you can use to search for text, and replace it if needed, in methods. In the **Find** tool, you can click **All** to search the entire application, including user interface components, model entity tags, operation identifiers, and labels. For example, searching for `method` also finds all methods, including all local methods, in addition to finding all occurrences of the text `Label1`. Click **Methods** to find and optionally replace a string in methods.

You can also type a string to replace the search string within the **Replace with** text field.

Under **Find in methods**, click **Current** to search only in the current method, or click **All** to search all methods. Under **Direction**, click **Forward** or **Backward** to control the search direction in the method. Select the **Case sensitive** check box to make the search case sensitive (the search string must match the text exactly, including uppercase and lowercase characters).


Click **Find Next** to find the next occurrence of the search string. Click **Replace** to replace it with the string in the **Replace with** text field. Click **Replace All** to search and replace all occurrences of the search string with the string in the **Replace with** text field.

Click the **Find All** button to launch the search and display the search results in a **Find Results** window, where each occurrence of the search string appears in a row. Double-click the row to highlight the search result in the method where it occurs. The **Method** column lists the method where the search string appears, the **Line** column lists the line where the string appears, and the **Text** column shows the text in which the search string appears.

Click the **Refresh** button (  ) in the top-left corner of the **Find Results** window to perform the search or search-replace action again and refresh the contents of the **Find Results** window.

### *Indentation and Whitespace Formatting*

---

To format the code in a method so that the code uses the correct indentations and whitespace formats, press Tab, or right-click in the method editor and select **Indent and Format** (  ). The formatting and insertion of white spaces applies to the selected part of the method's code or to the current line if no code is selected. By default, indentation and white space formatting also happen automatically when the keyboard focus leaves the method editor window. You can specify if you want indentation and formatting to be applied automatically when leaving the method editor window using the **Indent and format automatically** check box on the **Methods** page in the **Preferences** dialog box. The formatting is the same as that used for the automatically generated code that you can record in the Model Builder. The following list includes the most common whitespace rules:

- There is no space after a left parenthesis or before a right parenthesis.
- There is no space between a function name and the left parenthesis or between **with** and the left parenthesis.
- There is a space after keywords such as **if**, **while**, **for**, and **catch**.
- There is a space before and after operators, such as **=**, **==**, **<=**, **\***, **+**, **%**, **&&**, and **||**.
- There is a space after semicolon characters, but not before.
- There is no trailing whitespace on lines that contain at least one non-whitespace character.

To indent the currently selected lines without whitespace formatting, press Shift+Tab.

To increase or decrease indentation, regardless of the code formatting, use Ctrl+Alt+i to increase indentation and Shift+Ctrl+Alt+i to decrease indentation.

### *Brace Matching*

---

The method editor recognizes matching sets of braces (square brackets, curly braces, or parentheses) and highlights both matching braces when you click to select one of them. Use the following keyboard shortcuts to navigate between matching braces and to select the contents within braces:

- Ctrl+M, to move the cursor between matching braces
- Ctrl+Shift+M, to select the entire range of text between two matching braces

# Debugging Methods for Applications

## Indication of Compilation Errors

---

If there is a compilation error in a method, the method node shows a red cross:

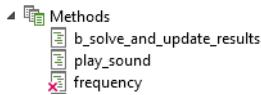


Figure 4-7: The frequency method contains some compilation errors. The other methods have no errors.

If there is a compilation error in a local method for a form object, the local method icon shows a red cross.



Figure 4-8: Indication that the local method for this input field contains code that caused compilation errors.

Open the method in a method error to see which lines of code that show a compilation error. The incorrect code is underlined, and a tooltip indicates the type of problem when you hover over the underlined code.

## Debugging Tools

---

The Application Builder includes some tools for debugging the code in methods used by an application or in a model as model methods. This functionality is available from the **Debug** and **Breakpoints** sections of the **Method** ribbon toolbar.

You can mark lines of code where the test run of an application should pause. This might be useful to, for example, make sure that a method really enters an if-statement or to make sure that a method actually ran. You can mark a line of code in this way by adding a breakpoint to a line by clicking in the margin to the left of the line numbers. The margin is then marked with a red dot that indicates the breakpoint. Note that clicking next to an empty line will not add a breakpoint — the line has to contain a statement for the breakpoint to make sense.

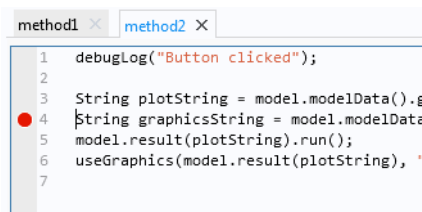


Figure 4-9: A method editor window with a breakpoint at line 4.

To remove a breakpoint, click on it again. All breakpoints can be removed in one action by using the **Remove All** (🗑️) button in the **Breakpoints** section of the **Method** ribbon tab. Sometimes, it is convenient to make the test

run of an application ignore the existing breakpoints. In this case, you can disable all breakpoints using the **Disable All** (🚫) toggle button. Disabled breakpoints are indicated by an empty red circle.

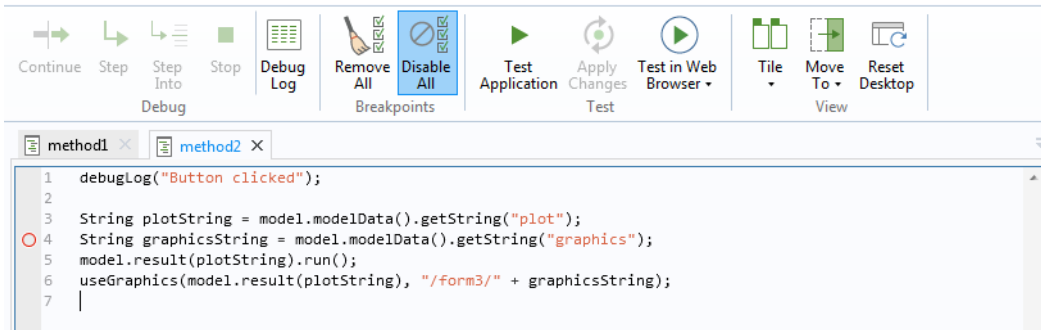


Figure 4-10: A disabled breakpoint on line 4 in the editor for method2.

If you run a method during the test run of an application, the method editor window is open, and the method has an enabled breakpoint, then the method will stop running on the line of the breakpoint. The entire line then becomes highlighted in yellow, and the **Continue** (➡), **Step** (↳), and **Step Into** (↳≡) buttons are enabled in the **Debug** ribbon tab. The **Continue** and **Step** buttons are only available when the debugger is paused.

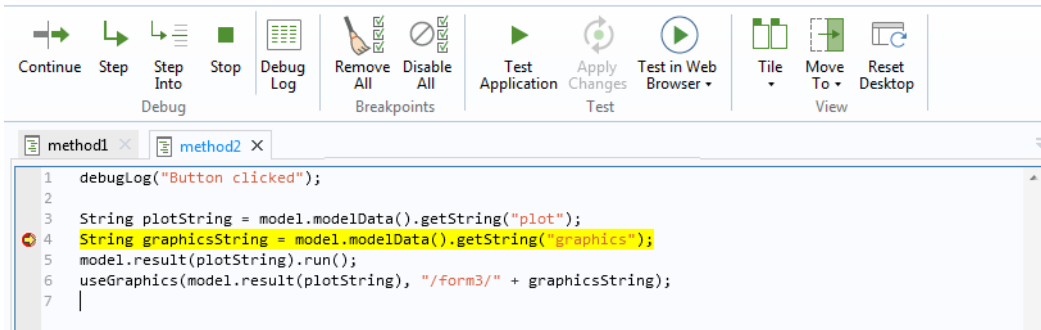


Figure 4-11: The debugger is stopped at a breakpoint, and the Continue and Step buttons are available. The code uses a debugLog method to print debugging information to the Debug Log.

If you click the **Continue** button (or press F5), the method continues to run until it completes or reaches another breakpoint. If you click the **Step** button (or press F6), the method continues to run until the next line, where it stops again, giving you the options to continue or step. The **Step Into** button works in the same way as the **Step** button. However, if the line contains a call to another method or utility method, clicking **Step Into** (or pressing F7) then takes you to the first line in that method instead of continuing to the next line in the current method. There is also a **Stop** button (■), which is available when you start to run a method. Clicking it forces the current method to stop. It can be useful if you want to stop the debugging run immediately or if the method runs into an endless loop, for example.

Often, it is useful to print messages that indicate that a line of code has been reached or to print the value of a certain object. You can do this using the `debugLog` method. There are `debugLog` methods that take a `String`, `double`, `int`, or `boolean`, or 1D or 2D array (matrix) types of these objects as input. These methods print a message to the **Debug Log** window. You can control the visibility of the **Debug Log** view using the **Debug Log** (☰) button in the **Method** ribbon tab.

### The Errors and Warnings Window

Compilation or syntax errors and warnings in methods appear in the **Errors and Warnings** window, which displays the current number of errors and warnings at the top. Click the **Check Syntax** button (ABC) in the Method toolbar



to perform a syntax check and display any errors and warnings. In the table below, errors and warnings appear with the method and line where they occurred under **Method** and **Line**, respectively. Under **Message**, you will find information about the particular error or warning. Double-click on a warning or error to highlight the code where it occurs in the corresponding method's editor window.

### *Handling Runtime Errors in Methods*

---

When you run a method in an application, runtime errors can occur. Such errors might not be discovered during a syntax check or compilation. When such an error occurs, an **Error** window appears and provides information about the type of error. Click **Details** to get additional information, including the name of the class and method, the line number, and a link to the method so that you can debug it in its editor window.

### *Stopping a Running Method*

---

When testing an application, a method may take a very long time to complete or contain an endless loop. You can stop running a method when testing applications by pressing Ctrl+Pause.



# Index

- A**
  - About dialog box 22
  - activation condition 44
  - Application Builder window 60, 149
  - Application Libraries window 14
  - application tree 13
  - Applications folder 14
  - apps
    - compiling 17
  - arranging form objects 66
  - Array Input form objects 120
  - array syntax 37
- B**
  - background images 31
    - for cards 116
  - Boolean values 38
  - brace matching 174
  - breakpoints, adding 175
  - browser, for file import 116
  - Button objects 76
- C**
  - canceling progress 103
  - Card form objects 114
  - Card Stack form objects 112
  - cards
    - adding 113
    - background images for 116
  - Check Box form object 83
  - choice lists 43
  - code completion 161
  - code folding 166
  - code, for model operations 168
  - columns and rows, in grid 70
  - Combo Box form objects 85
  - commands to run 24, 27, 34, 76, 80, 138
  - comments, in code 161
  - comparing applications 16
  - compilation error, indicating 70
  - compiler (node) 17
  - compiling apps 17
  - context menu, for form objects 66
  - continue and step, when debugging 176
  - copy forms, between applications 17
  - copying form objects 65–66
- D**
  - data access 63
  - Data Display form objects 93
  - Debug Log window 176
  - debugging code 175
  - declaration of local variables 172
  - deleting form objects 66
  - desktop layout
    - resetting 151
  - displaying progress 103
  - dragging
    - to copy form object 65
  - duplicating form objects 66
- E**
  - Editor Tools window 168
  - emailing COMSOL 8
  - Equation form objects 90
  - Excel files, saving to 107, 135
  - external c/c++ libraries 153
  - external Java libraries 153
  - extracting a subform 69
- F**
  - file (node) 45
  - File Import form objects 116
  - file schemes 157
  - files
    - getting to and from file system 158
  - filters, for checking user inputs 73
  - Find Results windows 174
  - Find tool 173
  - finding text 173
  - folding, of code 166
  - Form Collection form objects 110
  - form methods 151
  - form name 60
  - Form objects 109
  - form objects
    - arranging 66
    - Array Input 120
    - Card 114
    - Card Stack 112
    - Check Box 83
    - Combo Box 85
    - Data Display 93
    - Equation 90
    - File Import 116
    - Form 109
    - Form Collection 110
    - Graphics 95
    - Hyperlink 142
    - Image 100
    - Information Card Stack 118
    - Input Field 72
    - Line 92
    - List Box 130

- Log 104
- Message Log 106
- overview of 71
- Progress Bar 103
- Radio Buttons 123
- Results Table 107
- Selection Input 126
- Slider 139
- Spacer 145
- Table 133
- Text 128
- Text Label 88
- Toolbar 143
- Unit 89
- Video 101
- Web Page 98
- Form ribbon toolbar 60
- form title 60
- Form toolbar 63
- form windows 60
- formatting, of methods 174
- forms 29
  - preview of 29
- G** global methods 151
- graphics commands 62
- graphics data 46
- Graphics form objects 95
- grid
  - for sketching 30
  - resizing 69
- grid lines
  - showing 66
  - snapping to 66
- grid mode 66
  - rows and columns in 70
- H** Hyperlink form objects 142
- I** Image form objects 100
- images, as background 31
- importing files, command for 34, 77, 81, 138
- indentation 174
- information nodes, as source 72, 93, 128
- Informations Card Stack form objects 118
- Input Field form objects 72
- input fields
  - checking inputs 73
- inputs, to methods 50
- Internet resources 8
- K** knowledge base, COMSOL 9
- L** Line form objects 92
- List Box form objects 130
- local methods 50, 151
- local variables, declaration of 172
- Log form objects 104
- M** main window, of application 21
- matching, of brackets 174
- Message Log form objects 106
- method editor, tooltips in 164
- Method node 151
- Method toolbar 149
- methods
  - as commands 70
  - inputs and output 50
  - local 50
  - running in form objects 70
  - searching and replacing text in 173
  - show in Model Builder 50
  - stopping 177
  - types of 151
- model commands 62
- model progress 103
- O** object shortcuts 46
- on data change, event for 76
- output, from methods 50
- P** parentheses matching 174
- placement, of About link 22
- preview of form layout 60
- preview, of forms 29
- Progress Bar form objects 103
- progress bars 103
- progress, for model 103
- R** Radio Button form objects 123
- refreshing search 174
- regular expressions, for input checking 74
- replacing text, in methods 173
- resizing the grid 69
- Results Table form objects 107
- RFC 2396 file scheme 158
- rows and columns, in grid 70
- S** searches, refreshing 174
- searching for text 173
- searching text 173
- selecting entire lines of code 149
- selecting form objects
  - in sketch mode 65
- Selection Input form objects 126
- selections

- directly in graphics 96
  - using a Selection Input 126
- settings windows 13
- shortcuts
  - adding 46
  - using in methods 171
- size, of grid 70
- sketch mode 65
- Slider form objects 139
- snap zone 30, 115
- Spacer form objects 145
- stopping running applications 177
- string arrays, syntax for 37
- subform, extracting 69
- T** Table form objects 133
- technical support, COMSOL 8
- testing forms 70
- Text form objects 128
- Text Label form objects 88
- tiled or tabbed layout 110
- title, of application 21
- Toggle Button objects 79
- Toolbar form objects 143
- toolbars
  - Form 63
  - Method 149
- tooltips, in method editor 164
- tooltips, in ribbons 24, 26
- transferring files to client 157
- U** uniform resource identifiers 158
- Unit form objects 89
- unit sets 45
- utility classes 152
- V** valid Boolean values 38
- Video form objects 101
- View section 151
- W** web browser
  - testing apps in 55
- Web Page form objects 98
- websites, COMSOL 9
- whitespace formatting 174
- windows
  - rearranging 151
- Z** zoom to extents 95

