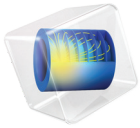


COMSOL Multiphysics

Model Manager Reference Manual

6.3

COMSOL

Model Manager Reference Manual

Version: COMSOL 6.3

Contact Information

Visit the Contact COMSOL page at www.comsol.com/contact to submit general inquiries or search for an address and phone number. You can also visit the Worldwide Sales Offices page at www.comsol.com/contact/offices for address and contact information.

If you need to contact Support, an online request form is located on the COMSOL Access page at www.comsol.com/support/case. Useful links:

- Support Center: www.comsol.com/support
- Product Download: www.comsol.com/product-download
- Product Updates: www.comsol.com/product-update
- COMSOL Blog: www.comsol.com/blogs
- Discussion Forum: www.comsol.com/forum
- Events: www.comsol.com/events
- COMSOL Video Gallery: www.comsol.com/videos
- Support Knowledge Base: www.comsol.com/support/knowledgebase
- Learning Center: https://www.comsol.com/support/learning-center

Part number: CM020017

Contents

Chapter 1: Introduction

Chapter 2: Model Manager Tools

Chapter 3: Searching and Filtering

Chapter 4: Advanced Version Control

Chapter 5: Working with Models in Databases

Chapter 6: Model Manager API

Chapter 7: Glossary

1

Introduction

Read this guide to learn how to use the *Model Manager*, a set of tools for helping you with version control of models and data files. The Model Manager is available directly from the COMSOL Desktop® and includes comprehensive functionality for saving, searching, organizing, and sharing models and data files stored in a Model Manager database. A database can either be created locally on your computer for personal use or you can set up a server database, hosted by a *Model Manager server*, for collaborative use.

In this chapter:

About the Model Manager

In this section:

What Can You Do with the Model Manager?

When working with a simulation model for an extended period of time, you will inevitably have a need to keep a backup of old versions. You may, for example, want to recover an older version in case your current modeling work goes astray, or perhaps you want to use an older version as a template for a completely new model. Your solution to this may vary from something as straightforward as saving files with different filenames on your local hard drive, saving to a file-based version control system, or by uploading files to a product lifecycle management (PLM) system provided by your organization. If you are working on models in a collaborative setting, you are also used to sharing files with your coworkers, for example via email, by placing them on a shared file system, or by allowing your coworkers to download them from a centralized version control system or PLM system.

As the amount of simulation models and data grows, you and your coworkers might find yourselves spending a large part of your time managing these models and data. This may involve working with multiple tools and software — keeping you away from your main modeling and simulation work. Some of the challenges and concerns you might face are:

- Efficient storage of models. Store only relevant data for the purpose of archiving and future retrieval, all while keeping disk space usage manageable.
- Automatic extraction of model metadata. Extract metadata when saving simulation models so that you and others may later find them, without requiring manual data entry for keywords and other search terms.
- Reuse of models. Use previously saved simulation models and data as building blocks when creating new models, perhaps generating an extensive library of reusable parts.
- Track and compare changes to models over time. Get an automatic audit trail for your simulation models to compare and restore older versions, or to reproduce modeling steps.

- Manage relationships between models and data files. Find out which simulation models use a particular data file as input or what simulation model generated a particular data file as output.
- Access control. Control who can find, open, and save simulation models and data.

The Model Manager comes with a set of tools for addressing these points — all while staying within the COMSOL Desktop modeling environment. From the COMSOL Desktop, you can create a new database on your own computer to keep track of your private models and data files, or you and your colleagues may share such models and files by uploading them to a server database accessed via a Model Manager server.

A Model Manager database is tailor-made for the storage needs of a model built in COMSOL Multiphysics®. The Model Manager makes sure to never store duplicates of simulation data when saving multiple versions of the same model. You can also avoid storing built, computed, and plotted data that may instead be reproduced from the model when needed.

The powerful Model Manager search syntax enables you to search deep into models based on their properties, features, settings, and other metadata. You may, for example, perform search queries answering:

- Which models use a Time Dependent study step?
- Which models have a Length parameter between 5 cm and 15 cm?
- Which models were last saved by me?

The Model Manager also comes with standard version control tools such as viewing the version history of models and data files, automatically detecting version conflicts when saving, and comparing versions with each other. You can, for example, open an older model version to create a completely new model with its own split-off version history, or see all the changes made to a model from one version to the next.

You can do exploratory work on an existing model in a database by creating a draft of the model. The draft is version controlled in its own right, enabling you to experiment with various simulation ideas without polluting the version history of the original model. Once you have finished your draft work, you may choose to either keep it as a new version of the original model or discard it.

More advanced version control tools such as branching, merging, and reverting are also available. Branching enables you, for example, to work on an entire collection of models and data files in isolation, while at the same time postponing the decision whether or not your changes are worth preserving. Reverting enables you, for

example, to restore models and data files that you have previously deleted, perhaps by accident.

The Model Manager also comes with an application programming interface (API) for use with the Java® programming language. The Model Manager API enables you to easily perform repetitive tasks by writing and running a few lines of code from the COMSOL Desktop. The API also enables you to perform more advanced database operations that would otherwise be difficult to do using built-in Model Manager tools.

When using a server database accessed via a Model Manager server, you can control who can access models and data files by setting permissions. You can, for example, set which users are permitted to open or save a particular model, or set which users are permitted to search and browse a collection of models. You can also use the web-based asset management system included with a Model Manager server installation to link your simulation models and results to various documents, presentations, project notes, slides, and other supplementary files and metadata — all while keeping everything in the same database storing your models. Using the asset management system is also a simple way of collaborating on simulation projects with people in your organization who may not have access to the COMSOL Multiphysics software. Simulation engineers can, for example, share output files from simulation runs by exporting them to the Model Manager server, while other engineers can upload new versions of data files via the Model Manager server web interface — versions which are then immediately available to the simulation engineer from the COMSOL Desktop.

Where Do I Access the Documentation?

A number of online resources have 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 an application or content referenced in a different guide. However, if you are using the Help system in COMSOL Multiphysics, these links work to open other modules, application examples, and documentation sets.

CONTACTING COMSOL BY EMAIL

For general product information, contact COMSOL at info@comsol.com.

COMSOL ACCESS AND TECHNICAL SUPPORT

To receive technical support from COMSOL for the COMSOL products, please contact your local COMSOL representative or send your questions to support@comsol.com. An automatic notification and a case number will be sent to you by email. You can also access technical support, software updates, license information, and other resources by registering for a COMSOL Access account.

COMSOL ONLINE RESOURCES

COMSOL website	www.comsol.com
Contact COMSOL	www.comsol.com/contact
COMSOL Access	www.comsol.com/access
Support Center	www.comsol.com/support
Product Download	www.comsol.com/product-download
Product Updates	www.comsol.com/product-update
COMSOL Blog	www.comsol.com/blogs
Discussion Forum	www.comsol.com/forum
Events	www.comsol.com/events
COMSOL Application Gallery	www.comsol.com/models
COMSOL Video Gallery	www.comsol.com/videos
Learning Center	www.comsol.com/support/learning-center
Support Knowledge Base	www.comsol.com/support/knowledgebase

Overview of the Manual

This *Model Manager Reference Manual* contains information that helps you get started with the Model Manager in the COMSOL Multiphysics product. The information in this guide is specific to this functionality. Instructions on how to use COMSOL in general are included with the *COMSOL Multiphysics Reference Manual*.

As detailed in the section Where Do I Access the Documentation?, this information can also be searched from the COMSOL Multiphysics software **Help** system.

Instructions on how to install, configure, and administer a Model Manager server is found in the *Model Manager Server Manual*. That manual also contains information about the asset management system included with a Model Manager server installation.

TABLE OF CONTENTS AND INDEX

To help you navigate through this guide, see the Contents section and Index.

TOOLS

The Model Manager Tools chapter has an overview of the tools available in the COMSOL Desktop and includes information about Adding Databases, Databases in the COMSOL Modeling Environment, and The Model Manager Workspace.

SEARCH

The Searching and Filtering chapter gives a detailed description of the search and filtering capabilities of a Model Manager database, including an overview of The Model Manager Search Syntax.

VERSION CONTROL

The Advanced Version Control chapter introduces more advanced tools for version control management in the Model Manager, including Branching, Merging, and Reverting.

TUTORIALS

The Working with Models in Databases chapter showcases how the Model Manager tools can be integrated into your modeling workflow by way of a few example tutorials.

API

The Model Manager API chapter contains examples that show how to perform various tasks involving Model Manager databases by writing and running a few lines of Java® code from the COMSOL Desktop.

GLOSSARY

The Glossary chapter gives a summary of various concepts and terms specific to a Model Manager database.

2

Model Manager Tools

This chapter provides an overview of the tools available in the Model Manager that enables you to save, search, organize, and share models and data files stored in a Model Manager database. To quickly get started with models in databases, see the tutorial Example: Modeling Using Version Control.

In this chapter:

- Maintenance
- Database Administration

Introduction

The Model Manager includes an extensive set of tools for working with version-controlled models and data files stored in databases. These tools are integrated into the COMSOL Desktop modeling environment (the Model Builder, the Application Builder, and the Physics Builder), enabling you to seamlessly switch between models and data files stored on the file system and in a Model Manager database.

In this, and in many other chapters, the term *model* will be used as a catch-all that also includes applications and physics. The term *data file* is a catch-all for other types of files that models may depend on as input or generate as output — including, for example, CAD data, interpolation functions, plots, and reports. See the Glossary for more details.

Some of the key Model Manager tools that you will encounter in your everyday modeling workflow are:

- Opening a model from a database from the **Open** window. See Opening Models from Databases to learn how you can find models to open.
- Saving a model to a database from the **Save** window. See Saving Models to Databases to learn how you can save your modeling work as a new version in a database.
- Reusing components, materials, physics and other model contents via the **Select Model** window. See Inserting Parts and Other Model Contents from Databases to learn how you can copy such contents from models stored in databases into the model currently opened in the COMSOL Desktop. See also Geometry Parts Saved in Databases to learn how you can reference geometry parts stored in databases in your models.
- Accessing older versions of a model from the **Versions** window. See The Versions Window for the COMSOL Desktop Model to learn how you can access previously saved versions of your model, including opening a version, restoring a version, or comparing two versions with each other.
- Selecting data files stored in a database as sources for input or targets for output in the **Select File** window. See Selecting Files in Databases as Input Sources to learn how you can import data into a model from a data file stored in a database. See Selecting Files in Databases as Output Targets to learn how you can export data from a model to a data file stored in a database.

You can quickly get started with the Model Manager by creating a new database stored locally on your computer. See Adding Databases and New Local Database to learn how you can create such a database directly from within the COMSOL Desktop.

The Model Manager is supported on the same platforms as COMSOL Multiphysics®: Windows®, Linux®, and macOS.

If you have installed a Model Manager server, you can also connect to its server database from within the COMSOL Desktop — see Connecting to a Server Database. Read more about installing, configuring, and administrating a Model Manager server in the *Model Manager Server Manual*.

The Model Manager comes with a dedicated workspace for database-specific tools such as browsing, organizing, and administrating your databases. See The Model Manager Workspace and further sections in this chapter to learn, for example, how you can browse and administer your configured databases in The Databases Window, search for models and data files in The Model Manager Window, and view settings, features, properties, and other metadata of your saved models in The Settings Window.

The Model Manager tools are default enabled in the COMSOL Desktop environment. You can hide all Model Manager functionality via the **Preferences** window by clearing the **Enable Model Manager** checkbox on the **Model Manager** page. A program restart is required.

Adding Databases

The COMSOL Desktop supports connecting to both a *local database* stored on the same computer that COMSOL Multiphysics is running on, as well as to a *server database* accessed via a Model Manager server. You can connect to as many databases as you like, and COMSOL Multiphysics will remember connected databases between program sessions. Connecting to a local or server database is also supported when Running COMSOL Multiphysics in Client–Server Mode.

A local database is intended for single-user use and is the recommended database when you want to keep your own models and data files under version control, without necessarily intending for others to see and work with them. It can also be useful when you just want to try out the various features that Model Manager offers. A server database is intended for multiple-user use and is the recommended database when you are working on models and data files in a collaborative setting.

Support for creating local databases is included with the COMSOL Multiphysics installation and requires no additional modules, software, or running processes. A server database requires a running Model Manager server, which can either be started on the same computer as COMSOL Multiphysics or on another computer that COMSOL Multiphysics has network access to.

To add a new database, do one of the following:

- In the **Open** window, the **Save** window, the **Select File** window, the **Select Model** window, or the **Export** window, choose **Add Database** ( ) from the list of options.
- In the Model Manager workspace, click the **Add Database** button ( ) in the **Database** section in the **Home** toolbar.

In this section:

The Add Database Window

From the **Add Database** window:

- Click the **New Local Database** button () to create a new Model Manager database that will be stored locally on your computer.
- Click the **Open Local Database** button () to add an existing Model Manager database stored locally on your computer.
- Click the **Connect to Server Database** button () to connect to a server database via a Model Manager server.

- New Local Database
- Opening a Local Database
- Connecting to a Server Database

New Local Database

To create a new local database on your computer:

1. Under **Database name**, write the name of the new database as stored on the file system.

   The database name can only contain characters that are valid for a directory name and filename. The name will also be used as the initial label of the database in the COMSOL Desktop, although this label can later be changed — see Database Configurations.
2. Under **Databases directory**, write the path to the parent directory that will contain the new database. You can also keep the suggested, prefilled directory path.

   The database itself will be created as a subdirectory inside this directory.
3. Click the **Add Database** button () to create the new database.

Creating a database on a network or cloud drive is not supported and COMSOL Multiphysics will report an error if it detects such a file system path. This includes placing the database in a local folder on your computer that is mapped to an external cloud drive via two-way synchronization. For more information, see https://www.comsol.com/support/knowledgebase/1295.

The field under **Databases directory** contains a prefilled default directory path appropriate for the underlying operating system. You can change this default from the **Preferences** window in the **Directory for local databases** field on the **Model Manager** page.

A LOCAL DATABASE ON THE FILE SYSTEM

The created database consists of a directory with the name specified by the **Database name** field, located within the parent directory specified by the field under **Databases directory** in the **New Local Database** window. This database directory contains:

- An SQLite® database file with the same name as the directory. Its file extension is `mphdb`.
- A `resources` directory containing files whose file sizes are deemed too large to store directly inside the SQLite® database file. These files can, for example, be built, computed, and plotted data generated by models, or standalone data files such as CAD data, interpolation functions, and reports.
- An `indexes` directory containing files used by Model Manager for searching and filtering. The file contents in this directory is automatically created by Model Manager using the data stored in the SQLite® database file and, as a result, may be safely skipped in backups.

The database directory also contains an `index` directory used for backward compatibility when the database is accessed from a COMSOL Multiphysics 6.0 installation. This directory and its file contents are created automatically as needed and may be safely skipped in backups.

Backup for Local Databases

Opening a Local Database

You can add an already created database to the COMSOL Desktop. This is useful, for example, if you want to move a local database created on another computer to your current computer or if you want to restore a database whose database directory has previously been backed up by external backup software.

From the **Open** dialog, browse to and choose the SQLite® database file for the database you want to add. Click the **Open** button to add the database.

Model Manager will check that it can connect to the SQLite® database file, as well as check that the `resources` directory and the `indexes` directory for the database, as described in A Local Database on the File System, are found next to the SQLite® database file. If the `indexes` directory cannot be found, Model Manager will recreate the directory and its file contents using the data stored in the SQLite® database file. If all checks succeed, the Model Manager database is added to the COMSOL Desktop. Otherwise, the **Open Local Database** window is shown.

1. Write the path to the SQLite® database file under **Database file**.
2. Write the path to the `resources` directory under **Resources directory**.
3. Write the path to the `indexes` directory under **Search indexes directory**.
4. Write a label for the database under **Label**.
5. Click the **Add Database** button (☑) to add the database to the COMSOL Desktop.

Adding a database stored on a network or cloud drive is not supported and COMSOL Multiphysics will report an error if it detects such a file system path. This includes placing the database in a local folder on your computer that is mapped to an external cloud drive via two-way synchronization. For more information, see https://www.comsol.com/support/knowledgebase/1295.

OPENING A LOCAL DATABASE FROM MULTIPLE COMSOL MULTIPHYSICS PROCESSES

You can access the same local database from multiple COMSOL Multiphysics program sessions as long as these program sessions run on the same computer that stores the database. There are, however, a few caveats to keep in mind:

- Changes saved from one COMSOL Multiphysics program session may not be immediately visible to other program sessions.
- Some advanced search and filter functionality, as described in Searching and Filtering, will only be available to the first program session that connects to the local database. Other program sessions use a simplified search.

Connecting to a local database from multiple COMSOL Multiphysics program sessions running on different computers by, for example, trying to place the local

database on a network drive or a cloud drive, or use some other type of sharing or synchronization tool is not supported.

	Use a server database accessed via a Model Manager server when working in a multiple-user or multiple-computer environment.

Connecting to a Server Database

You can connect to a server database via a Model Manager server running either locally on your computer or on a server computer in your organization's internal network.

	See the *Model Manager Server Manual* for instructions on how to install, update, configure, and otherwise administer a Model Manager server.

	If you are connecting to the Model Manager server for the first time using the administrator account set up when installing the server, you must first change the temporary password given during the installation. Log in to the Model Manager server from a web browser using your temporary password. If the web browser is running on the same computer as Model Manager server, the server address is `http://localhost:<port>` where `<port>` is the port number set during installation — default 8181. Enter your new password in the dialog and click **Save**.

1. Write the server address to the Model Manager server under **Server**.
2. Select the **Require secure connection** checkbox to require that the network connection is made using a secure connection, with transport layer security provided by HTTPS (as opposed to plain HTTP). A warning message is shown if the checkbox is cleared.

   You can leave out the port number if using a secure connection and the server is listening on port 443 or, if using a nonsecure connection, port 80. Otherwise, include a colon followed by the port number in the **Server** field — for example, `modelmanager.my-company.com:8181` for a Model Manager server listening on 8181.
3. Write your user account credentials used to authenticate with the Model Manager server under **User**. You can opt to remember the provided password between

program sessions by selecting the **Remember password** checkbox. The password will be stored in an encrypted form on the local file system.

4 Write a label to be used in the COMSOL Desktop for the server database under **Label**.

5 Click the **Connect** button ( ) to connect to the Model Manager server.

A secure connection attempt is still made even if you have not selected the **Require secure connection** checkbox. Only if that connection attempt fails, and the checkbox is cleared, will COMSOL Multiphysics fall back to a nonsecure connection via plain HTTP.

You are strongly recommended to connect to the Model Manager server using a secure connection, with transport layer security provided by HTTPS. Connecting via plain HTTP will send all data, including your credentials, in an unencrypted cleartext format. See the *Model Manager Server Manual* for configuring a Model Manager server to use secure connections.

To connect to a Model Manager server running on, for example, port `8181` on the same computer as COMSOL Multiphysics, write `localhost:8181` in the **Server** field.

To change your user account credentials or any other settings used to connect to an already added Model Manager server database, see Database Configurations.

CONNECTING TO A NONDEFAULT DATABASE

A Model Manager server can host more than one Model Manager database. An administrator may, for example, have set up multiple databases to be used by different departments in your organization or a secondary database used to store older archival data that can afford having a simplified backup plan.

One of the databases in a Model Manager server may be set as the *default server database* from the Model Manager server web interface by an administrator. This is also the server database that you connect to when following the previous steps. You can also choose to connect to a nondefault database using its so-called *database alias*, if such an alias has been set by an administrator from the web interface. Append `/db/`

<alias> to the server address in the **Server** field — for example, `modelmanager.my-company.com:8181/db/my-secondary-database` for a server database with alias `my-secondary-database`. Repeat the connection steps for every server database you want to add to the COMSOL Desktop.

	See the *Model Manager Server Manual* for how to configure server databases via the Model Manager server web interface including, for example, giving them database aliases.

	The database alias as set via the Model Manager server web interface is not the same as the alias you can set for a database configuration in the COMSOL Desktop — see Database Configurations. The latter is an alternative identifier to be used when connecting to either a local or a server database via method code using the Model Manager API.

CONNECTING VIA A COMSOL MULTIPHYSICS SERVER

When a COMSOL Multiphysics client is connected to a COMSOL Multiphysics server running on another host computer, the communication is typically over a nonsecure channel. This means that any credentials written in the user interface on the client computer would be sent to the COMSOL Multiphysics server computer in an unencrypted cleartext format. For this reason, Model Manager does not allow writing a password in the **Connect to Server Database** window when running in client–server mode unless it detects that the client–server connection is over a secure channel. You are instead met with a message in the **Connect to Server Database** window informing you that a secure password prompt will be shown when connecting to the database.

1. Fill out the connection details as you would when connecting from a standalone COMSOL Multiphysics program session, except you will not provide a password.
2. Click the **Connect** button ( ).

   A **Progress** dialog is shown informing you to provide credentials for the Model Manager server in the COMSOL Multiphysics server's console window.
3. In the console window for the started COMSOL Multiphysics server process, press Enter in order to provide your credentials. Write your username and password when prompted. You can also choose to remember your password for the Model Manager

server, in which case the password will be stored in an encrypted form on the local file system of the COMSOL Multiphysics server computer.

When you have finished providing your credentials, the **Progress** dialog is automatically closed, and a connection to the Model Manager server is established.

To avoid having to provide your credentials in the COMSOL Multiphysics server's console window, either tunnel the connection between the COMSOL Multiphysics client and the COMSOL Multiphysics server using a secure SSH connection, or first provide the credentials with **Remember password** selected using a local COMSOL Multiphysics instance running on the COMSOL Multiphysics server computer.

TROUBLESHOOTING

The following are some common error messages when connecting to a Model Manager server and suggestions on how to address them:

The server address could not be resolved. The server address in the **Server** field does not match the address of any server in your organization's internal network. Verify that you have written the correct server address. Alternatively, if you are able to log in to the web interface of the Model Manager server using a web browser, click on the COMSOL logo in the upper-left corner to make sure you are on the start page, and then copy the web page's link address from the web browser's address field. That link address should then be pasted into the **Server** field.

Connection timed out/Connection refused. This typically happens when the server computer was found in your organization's internal network but the Model Manager server itself is currently not running on that server computer. It can also happen if you have missed providing, or provided the wrong, port number for the Model Manager server.

The current password is temporary and must be changed. Log in to the Model Manager server web interface with this user account to change the temporary password. You are trying to log in using the temporary password set for the default administrative user account created during installation of the Model Manager server. First log in to the Model Manager server's web interface using a web browser and change the temporary password when prompted. Use the same link address in the web browser as written in the **Server** field.

Database configuration is not activated. The default server database has been deactivated by an administrator of the Model Manager server. The server database needs to be activated again via the Model Manager server's web interface before you can connect from COMSOL Multiphysics.

No default database configured. The administrator of the Model Manager server has not set a default database for the server. One of the configured server databases has to be set as the default one via the Model Manager server's web interface before you can connect from COMSOL Multiphysics.

Backward Compatibility for Model Manager

When COMSOL Multiphysics connects to a local Model Manager database created using an older COMSOL Multiphysics version, the database will be automatically upgraded to support any functionality added in the newer version. This upgrade does not modify the storage format of models and data files, however, so older versions of COMSOL Multiphysics will still be able to load models and data files from the database even after this upgrade — an important use case when you, for example, need to recompute a database-stored model using the exact same version of COMSOL Multiphysics the model was initially solved in.

Models saved in a Model Manager database have the same version requirement constraint as models stored as MPH-files on the file system: a model saved to a Model Manager database from a newer version of COMSOL Multiphysics cannot be opened from an older version of COMSOL Multiphysics.

When you connect from a newer version of COMSOL Multiphysics to a server database hosted by an older version of Model Manager server, the Model Manager server will not be aware of any newer functionality supported by COMSOL Multiphysics. In this case, such functionality will either be hidden or disabled in the COMSOL Desktop environment.

The opposite case is similar: When you connect from an older version of COMSOL Multiphysics to a server database hosted by a newer version of Model Manager server, you will not have access to all functionality supported by the newer server from the COMSOL Desktop environment. Same as for local databases, you will still be able to load older models and data files from the server database though.

Databases in the COMSOL Modeling Environment

The Model Manager tools are integrated with the COMSOL Desktop modeling environment — wherever you interact with models and data files on the file system, there is typically equivalent functionality for interacting with such models and data files in a database. In this section, you will find details on where and how you typically encounter the Model Manager tools in the COMSOL Desktop.

- Opening Models from Databases
- Saving Models to Databases
- Saving Drafts of Models
- Geometry Parts Saved in Databases
- Inserting Parts and Other Model Contents from Databases
- The Versions Window for the COMSOL Desktop Model
- Comparing Models Saved in Databases
- Running COMSOL Batch with Models in Databases
- Selecting Files in Databases as Input Sources
- Selecting Files in Databases as Output Targets
- Loading and Saving Auxiliary Data Files Stored in Databases
- The Auxiliary Data Window for Database Input and Output

Opening Models from Databases

From the **Open** window, you can find and open versions of models from one of your configured databases (see Adding Databases). Choose the database that you want to open a model from in the list of options on the left. Choose **Add Database** () if you want to add a new database.

The **Open** window is shown with a list of models saved in the database. Select a model and click the **Open** button () to open the model in the COMSOL Desktop. If a model is also an application, you can click **Run** () to launch and run the application directly.

You can also right-click a model to open it or, if the model is also an application, run it. Select **Set Tags** ( ) to modify the tag assignments of a model — see also Assigning Tags.

	The Open Window in the *COMSOL Multiphysics Reference Manual.*

	• Opening Models • Running Applications

FINDING MODELS TO OPEN

You find models to open by writing search expressions in the search field and clicking the **Search** button ( ) or pressing Enter. You can write plain search words and any number of filter expressions using The Model Manager Search Syntax. Plain search words will match on the title, description, assigned tags, and filename of a model. Press Ctrl+Space to get completion assistance when writing filter expressions — see Search Syntax Completion.

You can also apply separate Item and Content Filters via The Filter Dialog. Select a filter from the **Add Filter** menu button ( ) in the toolbar to open the dialog. Applied filters are shown below the search field — see Applied Filter Pills.

You search for the latest versions of models on a specific branch by default. To search among all versions ever saved in the database, click the expand button next to **Latest Versions for Location** menu button ( ) and select **All Versions in Database** ( ). See Searching Versions for more details on these two search modes. To redo a previous search, select one of the entries in the **Search History** menu list ( ) — see also Search History.

Each entry in the result list contains the title of the model in that version, the time when the version was saved, the name of the user that saved the version, and — depending on the search mode — either the tags assigned to the model or the repository and branch the model version was saved in. The search result is sorted on title, with a maximum of 100 models that match the search initially included in the result list. Click the **Show More** button ( ) to append the next 100 matches to the list. You can set another value for this default page size in the **Result Page Size** field on the **Model Manager** page in the **Preferences** window.

You can change the sort field and sort order of the search result from the toolbar. Click the expand button next to the sort field button to select another sort field than the default **Title** ( ) sort field — see also Sorting Search Results. The selected sort field and sort order for the **Open** window is remembered between program sessions.

Searching and Filtering

THE OPEN WINDOW TOOLBAR

The toolbar above the search field contains the following toolbar buttons:

- Click the **Refresh** button ( ) to refresh the search result while keeping the search expression and applied filters unchanged.
- Click the **Show More** button ( ) to append more matching models to the search result.
- Click the **Search History** menu button ( ) to select a previous search that you want to redo.
- Click the **Reset** button ( ) to clear the current search expression and applied filters.
- Click the **Add Filter** menu button ( ) to apply a filter.
- Click the sort field button to toggle the sort order of the search result between ascending and descending order. Click the expand button to select another sort field.

SELECT LOCATION IN DATABASE

You can change which subset of latest model versions to search for by selecting another commit location — see also Searching Versions and Locations. Click the link button above the search field to open The Select Location Dialog to select which location to search. The link button is hidden if there is only one location available in the database, which is the default for a new database, or when **All Versions in Database** ( ) is selected.

Saving Models to Databases

Any model opened in the COMSOL Desktop that is not protected with a password can be saved as a version to a Model Manager database. Model Manager will automatically detect if the opened model has previously been saved to the database

and, if so, save a new version of the existing model. This is true even if the model has been saved as an MPH file on the file system, closed, and then reopened from the file system at some intermediate point. It is also true if you save a model back-and-forth between different databases — for example, opening a model stored in a server database, saving it to a local database on your computer, and then saving it back as a new version to the server database.

From the **Save** window, you can save a new version of the model opened in the COMSOL Desktop to one of your configured databases. Choose the database that you want to save a model to in the list of options on the left. Choose **Add Database** ( ) if you want to add a new database.

The Save Window in the *COMSOL Multiphysics Reference Manual.*

The **Save** window can appear with four different headers depending on the model's presence in the selected database:

- **Save new**: The opened model has never before been saved to the database. A new model will be created in the database with an associated first version. If the model was originally opened from another database, it will use the same unique identifier in the new database as it did in the original database — thus enabling you to save it back to the original database as a new version at some later point.
- **Save version**: The database contains previously saved versions of the opened model. A new version of an existing model will be saved to the database.
- **Save version from draft**: The opened model is a draft version. A new version of the regular model that the draft was originally created from will be saved to the database.
- **Save new from draft**: The opened model is a draft version, but the regular model that the draft was originally created from does not exist in the database. This may happen if you, or another user, permanently deletes the regular model in the database while the draft is open in the COMSOL Desktop, or if you decide to save the draft to another database. As for **Save new**, a new model will be created in the database with an associated first version.

Saving Drafts of Models

To save a model version:

1. Write the title for the model version in the **Title** field.

   This title is kept in sync with the corresponding field in the **Presentation** section in the root node's **Settings** window. The title cannot be left empty.

2. Write an optional save comment in the **Comments** field.

   You can later read and update this save comment in The Versions Window or The Commits Window. You can also find your model version by applying a filter on the comment text.

3. Change the target branch for the save by clicking the **Location** link button and selecting another branch in The Select Location Dialog.

   The **Location** link button is hidden if the database only contains a single branch, which is the default for a new database.

4. Click the **Save** button ( ) to save a new version of the model.

   If you want to force the creation of a new model instead, click the expand button next to the **Save** button ( ) and select **Save as New** ( ) — see Splitting a Model Version History in Two.

Only changing the title of a model already saved in a database does *not* mean that you will create a *new* model. While providing a descriptive title helps you with later finding it in the database, it has no bearing on the database identity of the model.

THE INFORMATION SECTION

The **Information** section in the **Save** window displays additional information concerning the save. When saving a new version of an existing model in the database, this includes both when, and by whom, various versions were saved:

- **Latest version**: The most recent version of the model being saved. The latest version depends on the target branch set in the **Location** field.
- **Current version**: The version that the current model in the COMSOL Desktop was opened from. Only shown for **Save version** if the current version is not the same as the latest version — see Save Conflicts.
- **Current draft version**: The version that the current draft in the COMSOL Desktop was opened from. Only shown for **Save new from draft**.

The panel can also contain information messages ( ) that may be of interest, as well as warning messages ( ) signaling, for example, that the opened model is in conflict with the latest version.

Save Conflicts

When the **Save** window is shown, Model Manager will make a preemptive check that the model to be saved is not in conflict with the latest, already saved, version in the database. Such a conflict can arise, for example, if:

- The latest model version saved in the database is not the same version that the model in the COMSOL Desktop was opened from. Your coworker may, for example, have opened the same model as you, made some changes, and then saved before you now try to save it.
- The latest model version saved in the database is not the same version as a draft was originally created from. You and your coworker may, for example, have both started working on two separate drafts, and then your coworker saved their draft back to the original model before you now try to save it back to the same model. See Saving Drafts of Models for more details.
- The model in the COMSOL Desktop has been deleted on the target branch before you now try to save it.

Clicking the **Save** button ( ) while there are save conflicts opens a dialog asking if you want to save anyway. Click **Save** to ignore all conflicts, or **Cancel** for closing the dialog without saving. You can also click **Compare with Latest** ( ) to open the **Comparison Result** window to compare the opened COMSOL Desktop model with the latest version in the database. See Comparing the Opened Model in the COMSOL Desktop With the Latest Version for further details.

THE DESCRIPTION SECTION

The **Description** section in the **Save** window shows the current description of the model. As for the **Title** field, changes to the description are kept in sync with the corresponding field in the **Presentation** section in the root node's **Settings** window.

	Use the **Description** field for information that should carry over between different versions of the model. Use the **Comments** field for information that only applies to the specific model version being saved.

THE TAGS SECTION

The **Tags** section shows the tags assigned to the model as a collection of *tag pills*. You can assign tags already present in the database, as well as create new tags that should be assigned. You can assign as many tags as you like. Click a tag pill and select **Remove** ( ) to remove a tag assignment.

Any changes made to the assigned tags in the **Tags** section are saved to the database first when you save the model.

See Assigning Tags to Items to learn how tags may help you in organizing your models and data files in a Model Manager database.

Click the **Add Tag** button ( ) to find an existing tag to assign to the model. Write the title of the tag in the text field to filter the popup list of available tags. The matching tags are shown with their title followed by the titles of their parent tags, if any, within parentheses. Either double-click a tag in the list or select a tag and press Enter to assign it to the model.

Click the **Set Tags** button ( ) to open the **Set Tags** dialog. The dialog contains a tag tree of all available tags. Select a checkbox for a tag in order to assign the tag to the model. Clear a selected checkbox to remove a tag assignment. Click **Clear Tags** ( ) to clear all selections. You can filter the tree of available tags by writing a tag title in the text field above the tree. Click **OK** to finish the tag assignment.

If the tag you want to assign is not present in the database, you can create a new tag from the **Tags** section. Click the **New Tag** button ( ) to open the **New Tag** dialog.

1. In the **Title** field on the **General** tab, write the title for the new tag.
2. Select the tags that the new tag itself should be assigned — in other words, its *parent tags* — in the tag tree on the **Parent tags** tab. You can filter the tree of available parent tags by writing a tag title in the text field above the tree.
3. Write an optional save comment for the created tag in the **Comments** field.
4. Click **OK** to create the tag in the database.

Unlike the other tag operations in the **Tags** section, the new tag is immediately saved to the database when you click **OK** in the **New Tag** dialog. The assignment of the new tag to the model is, however, first saved when the model is saved.

You may want to double-check that a tag with the same title is not already present in the database before creating a new tag as Model Manager will not prevent such duplicates.

THE AUXILIARY DATA SECTION

The **Auxiliary Data** section is shown in the **Save** window if the model references any auxiliary data files that are version controlled in a Model Manager database and those data files currently have unsaved changes in the current COMSOL Multiphysics program session. Saving the model gives you the opportunity of simultaneously saving these data files to the same target branch that the model is saved to.

The section contains a table with the following columns:

- The **Save** column — a checkbox that, if selected, specifies that a new version of the data file should be saved when the model is saved.
- The **Title** column — the title of the data file. You can change the title by editing directly in the table cell.
- The **Files** column — the number of files to be saved. May be more than one in the case of a fileset — see Files.
- An icon column — shows an information icon when a previous version of the data file is not already present in the target branch, or a warning icon if there are version conflicts for the data file. The icon's tooltip gives further details.

Select a row in the table and click **Details** () to see further details on an auxiliary data file.

- Loading and Saving Auxiliary Data Files Stored in Databases
- The Auxiliary Data Window for Database Input and Output

PERMISSION CHECK

When saving to a server database via a Model Manager server, the Model Manager server will check that you fulfill the permission requirements to save a new version of the model. This involves checking that you are permitted to:

- Save in the repository containing the target branch.
- Save in the target branch.
- Save versions of the model.

Only the first two requirements apply when creating a new model in the database. If there is any auxiliary data that will be saved with the model, a permission check will also be made for saving versions of these items.

Permission Levels

A preemptive permission check is performed when the **Save** window is opened. If the check fails, a link button( ) that opens a dialog explaining why it failed appears next to the **Save** button ( ) — see The Permission Requirements Dialog.

Saving Drafts of Models

To save a ***draft*** of the model opened in the COMSOL Desktop, go to the **File** menu and select **Save Draft** ( ). You can also press the keyboard shortcut Ctrl+S.

The **Save Draft** ( ) option is only available if the model is opened from, or last saved to, a database.

The keyboard shortcut Ctrl+S either saves the current model to the file system, or as a draft in the database, depending on where the model was last saved. The file system is the default for unsaved models.

See also Example: Modeling Using Version Control for a tutorial that involves using a draft in the COMSOL modeling environment.

A draft model behaves in exactly the same way as a *regular* model in the database, except that a draft offers a streamlined way of saving it back as a new version of the model it originated from.

Working with a new draft model in the COMSOL Desktop typically involves these steps:

1. Open a regular model from a database.
2. From the **File** menu, select **Save Draft** ( ), or press Ctrl+S, to save a first version of a new draft of the opened model.

3 Work on the draft, intermittently selecting **Save Draft** ( ) or pressing Ctrl+S to save additional draft versions.

4 From the **File** menu, select **Save as Version** ( ). Click **Save** ( ) to save the opened draft as a new version of the original model. Click **Save as New** ( ) to create a new model from the draft. See Saving Models to Databases.

You can also open an existing draft model from the **Open** window — perhaps in order to continue working on a draft created in a previous COMSOL Multiphysics program session.

Drafts

Geometry Parts Saved in Databases

You can version control *geometry parts* in a Model Manager database by saving the corresponding model containing the part via, for example, the **Save** window — much like you would save any other model to a database. Model Manager will automatically detect if a saved model version contains any reusable geometry parts and, if so, store associated metadata in the database. This enables you, for example, to easily find such model versions by applying a Part filter when searching in the database.

If you load a part stored in a database into the model opened in the COMSOL Desktop and then save the model to the same database as the part, Model Manager will store an explicit reference link between the saved version and the part's version. This helps you to keep track of, for example, which models are currently using a particular part, as well as preventing the part from being permanently deleted — see Reference-tracking in Model Manager for further details.

Using Geometry Parts in the *COMSOL Multiphysics Reference Manual*

Inserting Parts and Other Model Contents from Databases

From the **Select Model** window, you can find and select models stored in a database to load one of their geometry parts or to insert one of their components, geometry sequences, physics, or materials into the model opened in the COMSOL Desktop.

Choose the database that you want to select from in the list of options on the left. Choose **Add Database** ( ) if you want to add a new database.

The **Select Model** window is opened when you, for example, click:

- **Insert Components From** ( ) in the **Add Component** menu in Model Builder's **Home** toolbar.
- **Insert Physics From** ( ) in the **Insert Physics** menu in Model Builder's **Physics** toolbar.
- **Insert Sequence From** ( ) in the **Insert Sequence** menu in the Model Builder's **Geometry** toolbar.
- **Import Materials From** ( ) in the **Import Materials** menu in Model Builder's **Materials** toolbar.
- **Load Part From** ( ) in the **Load Part** menu in Model Builder's **Geometry** toolbar. A **Part** filter ( ) on **Geometry** is automatically applied in the **Select Model** window.

The **Select Model** window for a database offers identical search and filter functionality as the **Open** window — see Opening Models from Databases. Click the **Select** button ( ) once you have found and selected the sought-after model version.

You can also load geometry parts or insert components, geometry sequences, physics, and materials — as well as parameters — directly from the **Contents** section in the **Settings** window for a model version. See Inserting Contents from Models for details.

The Versions Window for the COMSOL Desktop Model

The **Versions** window in the Model Builder workspace shows the history of the model opened in the COMSOL Desktop when that model is opened from a database or was last saved to a database. You can use the **Versions** window to, for example, quickly get an overview of recently saved versions, compare what was changed between two versions, open older versions, or even restore an older version as a new latest version.

From the **Windows** menu ( ) in the **Layout** section of Model Builder's **Home** toolbar, select **Versions** ( ) to open the **Versions** window.

The window contains a table with versions of the model sorted in chronological order, most recent first. A maximum of 100 versions are initially retrieved from the database.

Click the **Show More** button ( ) to append the next 100 versions to the table. You can set another value for this default page size in the **Result Page Size** field on the **Model Manager** page in the **Preferences** window. The version that is opened in the COMSOL Desktop is highlighted in bold, which typically is the first table row.

The table columns are:

- The type column — the type of the model version represented by an icon. See Item Version Types and Item Save Types.
- The **Title** column — the title set for the model in that version.
- The **Saved** column — the time when the version was saved.
- The **Saved By** column — the name of the user that saved the version.
- The **Branch** column — the target branch the version was saved in. There is only a single branch, default named **Main**, when creating a new database.
- The **Comments** column — the optional comment provided when the version was saved.

THE VERSIONS WINDOW TOOLBAR

The toolbar above the table contains the following toolbar buttons:

- Click the **Refresh** button ( ) to refresh the table in case a new version has been saved. The table will automatically refresh if you save a new version to the database from the COMSOL Desktop.
- Click the **Show More** button ( ) to append older versions to the table.
- Click the **Version Details** button ( ) to open The Version Details Dialog containing more information on a specific version.
- Click the **Open** button ( ) to open a selected version in the COMSOL Desktop.
- Click the **Run** button ( ) to launch and run a selected version in the COMSOL Desktop. Only enabled if the selected version is an application.
- Click the **Compare** button ( ) to compare a selected version with the model opened in the COMSOL Desktop. Select two versions to compare them with each other. See Comparing Models Saved in Databases.
- Click the **Restore Version** button ( ) to save the selected version as a new latest version of the model opened in the COMSOL Desktop. The target branch for the save is the same as that of the opened model version. See Restore Version for further details.

If you right-click a model version, you can also:

- Select **Disk Space Usage** ( ) to see an estimate of the required disk space usage if the model version is saved to the file system. See Estimating Disk Space Usage.
- Select **Copy Location** ( ) to copy a text string with a URI that uniquely identifies the model version in the database to the clipboard. See Copying Model and File Locations.
- Select **Clear Computed Data** ( ) to clear all built, computed, and plotted data of the model version. Data shared with other model versions via deduplication will not be deleted from the database. Clearing cannot be undone. See Built, Computed, and Plotted Data.
- Select **Permanently Delete** ( ) to permanently delete the model version in the database. Data shared with other model versions via deduplication is not deleted. Permanently deleting cannot be undone. See Permanently Deleting Models and Data Files.

Estimating disk space usage, clearing computed data, and permanently deleting is also supported when selecting multiple model versions in the table.

SPLITTING A MODEL VERSION HISTORY IN TWO

When you create a new model from an existing model, for example by saving a new draft or selecting **Save as New** ( ) in the **Save** window, the Model Manager database stores a reference to the *origin model* from the new model (or rather, the version of the original model that was saved from). You can think of the new model as being *split off* from the original model, such that the new model receives its own identity and version history.

The **Versions** window helps you keep track of a model's potential origin by including the versions of the latter *up to* the source version that the new model was created from in the window's table — see Figure 2-1 for a schematic representation. Versions of the origin model saved *after* the current model was created are *not* included in the table, owing to the fact that their changes are not included in the current model.

You are likely to first encounter this split-off in the **Versions** window when saving a new draft from a regular model — see Saving Drafts of Models. The top table rows correspond to the versions of the saved draft. The remaining table rows are the versions of the regular model that the draft originated from.

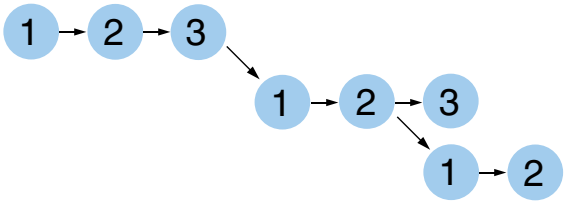


Figure 2-1: A schematic of the version history of three models, with the chronological order of versions read from left to right. The first version of the middle model was created from the third version of the top model, and the first version of the bottom model was created from the second version of the middle model. The version history of the top model includes three versions, that of the middle model includes six versions, and that of the bottom model includes seven versions. The history of the bottom model does not include the third version of the middle model as that was saved after the bottom model was split off.

THE VERSIONS WINDOW AND MULTIPLE BRANCHES

The version history of a model becomes even more complex once you create additional branches in your Model Manager database. The **Versions** window shows the history of the model with respect to the latest version on a particular branch, with the intent of visualizing how one version has progressed to the next by going upward in the table until reaching that latest version. Versions of the model saved on other branches are *not* included in the table *unless* the current branch was created from one of those other branches — see also Branching. If so, versions saved on that *parent* branch are appended at the bottom of the table if the version was saved *before* the current branch was created. Other versions on the parent branch saved at a more recent date, or versions saved on unrelated branches, are excluded by the assumption that they correspond to independent work done in parallel and whose changes are thus unrelated to the model version currently opened in the COMSOL Desktop.

You can see *all* versions of a model, irrespective of branch, by adding the model to The Maintenance Window in the Model Manager workspace. From the window, you gain a complete overview of the full footprint of the model in the database, but may lose a sense of how the model has evolved over time.

Comparing Models Saved in Databases

The **Comparison Result** window enables you to compare versions of models stored in databases. In this section, you will see five different ways you can encounter comparisons involving such models:

- Comparing Two Versions From a Model's History
- Comparing Two Models
- Comparing a Version With the Opened Model in the COMSOL Desktop
- Comparing the Opened Model in the COMSOL Desktop With the Saved Version
- Comparing the Opened Model in the COMSOL Desktop With the Latest Version

Comparing Models and Applications in the *COMSOL Multiphysics Reference Manual*

COMPARING TWO VERSIONS FROM A MODEL'S HISTORY

If you select two model versions in the **Versions** window and click **Compare** (), the **Comparison Result** window will open with the older version shown on the left, labeled as **Older Version**, and the newer version on the right, labeled as **Newer Version**. The versions need not belong to the same model: the older version may, for example, belong to an origin model of the newer version's model — see Splitting a Model Version History in Two. Since model versions are immutable in the database, it is not possible to merge or override any differences from the **Comparison Result** window.

COMPARING TWO MODELS

If you select two models in The Model Manager Window or The Databases Window and click **Compare** (), the **Comparison Result** window will open with one of the models labeled as **First** and the other as **Second**. The compared versions depend on the context where the models were selected. Selecting two models when Searching in Branches will compare the latest versions of each respective model. Selecting when Searching in Snapshots and Commits will compare the versions that were the latest at the time of the corresponding commit. Similar to when comparing two versions from a model's history, it is not possible to merge or override any differences in the **Comparison Result** window.

COMPARING A VERSION WITH THE OPENED MODEL IN THE COMSOL DESKTOP

If you select a single model version and click **Compare** (), the **Comparison Result** window will open with a comparison between the model opened in the COMSOL Desktop, labeled **Opened**, and the selected version, labeled **Saved version**.

Right-click the top node in the **Differences** tree and select **Merge Changes to Opened** () to merge all changes marked as *incoming* (that is, changes to the model tree made in the selected model version) into the desktop model. All changes marked as *outgoing* (that is, changes made to the model tree in the desktop model) are left alone.

Right-click any node in the **Differences** tree and select **Override Difference in Opened** () to write all changes in that node to the opened model, regardless if any changes are incoming or outgoing, effectively reverting any changes you have made in the desktop model.

COMPARING THE OPENED MODEL IN THE COMSOL DESKTOP WITH THE SAVED VERSION

After having opened a model version from a database and worked on it for a while in the COMSOL Desktop, you may want to see all changes you have made. As a shortcut to selecting the corresponding saved version in the **Versions** window and clicking **Compare** () from the window's toolbar, you can click **Compare with Saved** () in the Model Builder's **Developer** toolbar. The **Comparison Result** window is opened with a comparison between the model in the COMSOL Desktop and the version it was loaded from.

COMPARING THE OPENED MODEL IN THE COMSOL DESKTOP WITH THE LATEST VERSION

If you encounter a version conflict when trying to save a new version of the model opened in the COMSOL Desktop, you can open the **Comparison Result** window with a comparison between the desktop model, labeled **Opened**, and the latest version of the model being saved, labeled **Latest Version** — see also Save Conflicts. Similar to Comparing a Version With the Opened Model in the COMSOL Desktop, you can merge changes and override differences into the desktop model to include those changes you want to keep from the latest version.

Running COMSOL Batch with Models in Databases

The COMSOL Multiphysics batch mode, when launched from a command-line interface, supports using models stored in a Model Manager database as both source

input and target output for the simulation run. You specify a model version as input using the command-line argument `-inputfile` with a model version location as its value — see Copying Model and File Locations. The output is specified using the command-line argument `-outputfile` and a second model version location. The output of the simulation run is saved as a new version for the same model and branch that the version specified by the output argument belongs to. You may skip the output argument altogether, in which case it defaults to that of the input argument. You may also combine a model version in a database as input with an MPH file on the file system as output, and vice versa.

The model version location must typically be quoted when used in a command-line interface since it contains characters that are commonly reserved as operators in the shell. Write, for example, `-inputfile "<model-version-location>"`.

When running COMSOL Multiphysics in batch mode, various log and status files are continuously written by the batch processes. These files are always stored on the file system.

See COMSOL Batch Commands (Windows) and COMSOL Batch Commands (Linux) in the *COMSOL Multiphysics Reference Manual* for general details on running batch from a command-line interface.

You can stop a batch run involving a model version by launching a new batch process using the same arguments as the first process but with an additional `-stop` argument. The batch run can be restarted using an additional `-continue` argument. For the case when the output is saved to a database, however, it becomes important to use the model version that was saved when the previous batch run was stopped as a new argument value for `-inputfile`. Otherwise, you will end up recomputing all solutions already obtained by the previous batch run. You would typically find the saved model version as the latest version of the model by browsing the corresponding branch in the Model Manager workspace, or via the Model Manager API using the `-outputfile` argument of the previous run as an "anchor" — see Navigating a Model Manager Database.

In case of a failure or otherwise interrupted run, and when the output target is a database, an attempt is first made to save the current run as a model version to be used

later as a recovery. If this save fails, a recovery is instead saved on the file system. You can recover the batch run by launching a new batch process with the extra `-recover` argument in addition to the arguments used by the previously failed batch run.

You should not use the recovery model version for the `-inputfile` argument when recovering a failed batch run. That version will be automatically discovered by the recovering process.

Selecting Files in Databases as Input Sources

Data files stored in a Model Manager database can be used as input for models in the same way you can use files stored on the file system as input. From the **Select File** window for an input setting, you can find and select files from one of your configured databases to use as input source for the model opened in the COMSOL Desktop. Choose the database that you want to select files from in the list of options on the left. Choose **Add Database** ( ) if you want to add a new database.

The **Select File** window is, for example, opened when you click the expand button next to **Browse** ( ) and select **Browse From** ( ) for a model input setting in the **Settings** window in the Model Builder workspace. It is also opened when you click the expand button next to **Load from File** ( ) and select **Load From** ( ) for a table in the **Settings** window in the Model Builder workspace.

The **Select File** window for a database offers similar search and filter functionality as the **Open** and **Select Model** windows, although adapted for searching data files. Click **Select** ( ) once you have found and selected a file in the result list to use as input. If the selected data is a fileset, you will be asked to select one of its file resources via the **Select File from Fileset** dialog.

You can also right-click a file in the list to select it as input. Select **Preview File** ( ) to open the file with the default application for its file type — see also Previewing Files. Select **Set Tags** ( ) to modify the tag assignment of the file — see also Assigning Tags.

Selecting a data file from a Model Manager database via the **Select File** window and clicking **Select** ( ) typically only specifies a source for the input. The actual loading of the input data from the database often happens later — either automatically when the data is needed or manually by, for example, clicking an **Import** button. See Loading and Saving Auxiliary Data Files Stored in Databases for more details.

FINDING DATA FILES

You find data files by writing search expressions in the search field and clicking the **Search** button ( ) or pressing Enter. You can write plain search words and any number of filter expressions using The Model Manager Search Syntax. Plain search words will match on the title, description, assigned tags, and filename — or, for a fileset, filenames — of a data file. Press Ctrl+Space to get completion assistance when writing filter expressions — see Search Syntax Completion. The search can be restricted by an explicit file type filter set in the list next to the search field.

You can also apply separate Item Filters via The Filter Dialog. Select a filter from the **Add Filter** menu button ( ) in the toolbar to open the dialog. Applied filters are shown below the search field — see Applied Filter Pills.

You search for the latest versions of files on a specific branch by default. To search among all versions ever saved in the database, click the expand button next to **Latest Versions for Location** menu button ( ) and select **All Versions in Database** ( ). See Searching Versions for more details on these two search modes. To redo a previous search, select one of the entries in the **Search History** menu list ( ) — see also Search History.

Each entry in the result list contains the title of the file in that version, the time when the version was saved, the name of the user that saved the version, and — depending on the search mode — either the tags assigned to the file or the repository and branch the file version was saved in. The search result is sorted on title, with a maximum of 100 files that match the search initially included in the result list. Click the **Show More** button ( ) to append the next 100 matches to the list. You can set another value for this default page size in the **Result Page Size** field on the **Model Manager** page in the **Preferences** window.

You can change the sort field and sort order of the search result from the toolbar. Click the expand button next to the sort field button to select another sort field than the default **Title** ( ) sort field — see also Sorting Search Results. The selected sort field and sort order for the **Select File** window is remembered between program sessions.

Searching and Filtering

THE SELECT FILE WINDOW TOOLBAR

The toolbar above the search field contains the following toolbar buttons:

- Click the **Refresh** button ( ) to refresh the search result while keeping the search expression and applied filters unchanged.
- Click the **Show More** button ( ) to append more matching files to the search result.
- Click the **Search History** menu button ( ) to select a previous search that you want to redo.
- Click the **Reset** button ( ) to clear the current search expression and applied filters.
- Click the **Add Filter** button ( ) to apply a filter.
- Click the sort field button to toggle the sort order of the search result between ascending and descending order. Click the expand button to select another sort field.

SELECT LOCATION IN DATABASE

You can change which file versions to search for by selecting another *location* — see Searching Versions and Locations. Click the link button above the search field to open The Select Location Dialog to select which location to search. The link button is hidden if there is only one location available in the database, which is the default for a new database, or when **All Versions in Database** ( ) is selected.

Selecting Files in Databases as Output Targets

Output data generated by a model can be stored as data files in a Model Manager database in the same way you can store such output as files on the file system. From the **Select File** window for an output setting, you can either specify a new file or select an existing file in one of your configured databases to use as output target for the

model opened in the COMSOL Desktop. Choose the database that you want to select files from in the list of options on the left. Choose **Add Database** ( ) if you want to add a new database.

The **Select File** window is, for example, opened when you click the expand button next to **Browse** ( ) and select **Browse From** ( ) for a model output setting in the **Settings** window in the Model Builder workspace.

Click the **Select New** radio button in the **Select File** window for a database to specify a new file target. Write a title for the output data in the **Title** field. The **Filename** field will be automatically populated with a suggested filename for the main output file based on the title and the output's file extension.

Change the target branch for the output by clicking the **Location** link button and selecting another branch in The Select Location Dialog. The **Location** link button is hidden if the database only contains a single branch, which is the default for a new database, or when **All Versions in Database** ( ) is selected.

Click the **Select Existing** radio button to find and select an existing data file as output target. The window offers the same search and filter functionality as when selecting a data file from a database as input source — see Selecting Files in Databases as Input Sources.

Click **Select** ( ) once you have specified a file to use as target for the output.

Selecting a data file from a Model Manager database via the **Select File** window and clicking **Select** ( ) only specifies a target for the output. The actual writing and saving of the output data to the database happens later — either directly by clicking, for example, an **Export** button or as a two-step procedure in which the output is first written to a temporary folder on the local file system and then manually saved to the database. See Loading and Saving Auxiliary Data Files Stored in Databases for more details.

Switching to the **All Versions in Database** () search mode and selecting an existing version as output target that is not the latest version on a branch will inevitably lead to a version conflict once you try to save any written output to the database. You can choose to manually ignore this conflict from the **Export** window — see Loading and Saving Auxiliary Data Files Stored in Databases.

Loading and Saving Auxiliary Data Files Stored in Databases

File versions stored in a database that are referenced as auxiliary data by a model opened in the COMSOL Desktop — either as input source or output target — are loaded on-demand from the database to a temporary *working copy* directory located on the computer running COMSOL Multiphysics (the server computer when running COMSOL Multiphysics in client–server mode). Any input read by the model, and any output written by the model, goes via files in this directory.

There are various situations in which the model will write to the files in the working copy directory. You may, for example, have added an **Export to File** node () for a **Parametric Sweep** such that, when the corresponding study is computed, export nodes in the **Model Builder** tree run for different parameter values. When Model Manager detects that there are unsaved changes written by the model to the working copy directory, those changes can be saved as a new file version when the model itself is saved via the **Save** window — see Saving Models to Databases. You can also save a new file version directly via the **Auxiliary Data** window — see The Auxiliary Data Window for Database Input and Output.

Changes in working copy directories are not automatically saved when running an external COMSOL Multiphysics batch process. In this case, you can use the Model Manager API from a model method to save the working copy contents as a new file version.

EXPORTING OUTPUT DIRECTLY TO A DATABASE

You can save output to a database from nodes in the **Model Builder** tree with export functionality. If you have specified a database as the output target via the **Select File** window, you can click an **Export** button (), or something similar, to export and save the output as a new file version.

For better control over the saved file version, you can also perform the export via the **Export** window. This enables you, for example, to write a custom save comment or change the tag assignments of the file. Choose the database that you want to export to in the list of options on the left. Choose **Add Database** ( ) if you want to add a new database.

The **Export** window is, for example, opened when you click the expand button next to **Export** ( ) and select **Export To** ( ) in the **Settings** window in the Model Builder workspace.

To export output data and save it as a new file version:

1. Write the title for the file version in the **Title** field.
2. Write the filename of the main output file in the **Filename** field.
3. Write an optional save comment in the **Comments** field.

   You can later read and update this save comment in The Versions Window or The Commits Window. You can also find your file version by applying a filter on the comment text.
4. Click the **Save** button ( ) to export output data and save it as a new file version.

   If you want to force the creation of a new data file instead, click the expand button next to the **Save** button ( ) and select **Save as New** ( ).

The **Export** window is otherwise similar to the **Save** window used to save a new model version — see Saving Models to Databases. You can change the target branch for the export by clicking the **Location** link button and selecting another branch in The Select Location Dialog. The **Location** link button is hidden if the database only contains a single branch, which is the default for a new database. There is also an **Information** section displaying additional information concerning the saved file, a **Description** section for changing the file's description, and a **Tags** section for setting the assigned tags of the file.

The Auxiliary Data Window for Database Input and Output

The **Auxiliary Data** window includes models and data files stored in a database that are referenced in the opened model as input or output. This includes, for example,

interpolation functions, geometry parts, CAD assemblies, plots, reports, as well as many other types of *auxiliary data* for a model. For such items, you can:

- Show all versions in The Versions Window in the Model Manager workspace. Click the **Show Location** button ( ) in the toolbar. This is useful, for example, if you want to see potentially newer versions for a referenced model or data file in the database.
- Save a new version of a data file in the database using the file contents found in the working copy directory for the version. Click the **Save as Version** button ( ) in the toolbar and save a new version in the **Save File** dialog. This is useful, for example, if the model has written output data in the current COMSOL Multiphysics program session and that output has not yet been saved to the database. See also Loading and Saving Auxiliary Data Files Stored in Databases.
- Update the reference to the latest version of the model or data file in the database. Click the **Update to Latest Version** button ( ) in the toolbar.

You can also see the current status of a referenced model or file version:

- **Database not connected** — the status could not be determined as the database is not connected. Right-click a table row and select **Connect to database** ( ), or open the Model Manager workspace and activate the database from the **Database** section of the **Home** toolbar — see Activating a Database.
- **Up to date** — the referenced version is the latest version.
- **Newer versions exist** — a later version exist on the same branch as the referenced version.
- **Not authorized to access item** — you do not meet the current permission requirements for accessing the model or data file. See Permission Levels.
- **Not available in database** — the referenced version has been permanently deleted in the database.
- **Unsaved changes** — the model has written output to the working copy directory of the file version in the current COMSOL Multiphysics program session and that output has not yet been saved to the database.

You can import referenced models and data files stored on the file system into the same database as the model, thereby placing them under version control. Select the items you want to import in the table and click **Import to Database** ( ) in the toolbar — see Importing Files.

You can also save a new version of a referenced data file already present in the database by updating it from The Settings Window in the Model Manager workspace — see File Settings. Select the file in the **Auxiliary Data** window's table and click the **Update to Latest Version** button ( ) to use the new file version in the model.

	The Auxiliary Data Window in the *COMSOL Multiphysics Reference Manual*

The Model Manager Workspace

Apart from the Model Manager tools integrated with the Model Builder, Application Builder, and Physics Builder workspaces — see Databases in the COMSOL Modeling Environment — the Model Manager comes with a dedicated workspace for database-specific tasks. In this section, you will find a brief overview of this workspace. Later sections will discuss each part of the workspace in more depth.

- Opening the Model Manager Workspace
- The Home Toolbar
- The Database Toolbar
- The Maintenance Toolbar
- The Model Manager Workspace Windows

Opening the Model Manager Workspace

To open the Model Manager workspace, click the **Model Manager** () button in the **Workspace** section of the **Home** toolbar in either the Model Builder workspace, the Application Builder workspace (if running on Windows®), or the Physics Builder workspace. You can also press the keyboard shortcut Ctrl+Shift+J. The COMSOL Desktop switches to display the toolbar for the Model Manager, as well as opens windows belonging to the Model Manager workspace. A connection attempt is made to the most recently used database upon opening the workspace.

To return from the Model Manager workspace to one of the other workspaces, click on the corresponding button in the **Workspace** section of the Model Manager's **Home** toolbar. You can also press Ctrl+Shift+M for Model Builder and Ctrl+Shift+A for Application Builder.

The Home Toolbar

The **Home** toolbar contains buttons for the more commonly performed tasks in the Model Manager workspace.

THE WORKSPACE SECTION

This section contains buttons for switching to other workspaces in the COMSOL Desktop:

- The **Model Builder** button ( ) or **Physics Builder** button ( ), depending on if a model or physics is opened in the COMSOL Desktop. You can also use the keyboard shortcut Ctrl+M.
- The **Application Builder** button ( ) if running on Windows®. You can also use the keyboard shortcut Ctrl+A.

THE DATABASE SECTION

This section contains buttons for adding databases, switching the active database in the Model Manager workspace, as well as importing and exporting items in a database.

- The **Activate Database** button, to refresh the active database in the workspace. Click the lower part of the button and select one of the databases to set it as active. See Activating a Database.
- The **Add Database** button ( ), to open the **Add Database** window for adding a new database. See The Add Database Window.
- The **Import** button ( ), to import files from the file system into a database. See Importing Files.
- The **Export** button ( ), to export items from a database to the file system. See Exporting Items.
- The **New Tag** button ( ), to create a new tag in a database. See Creating New Tags.

THE ITEM SECTION

This section contains buttons that target items in the database — that is, models, data files, and tags.

- The **Open** button ( ) to open a model in the COMSOL Desktop. See Opening Models.
- The **Run** button ( ) to launch and run an application in the COMSOL Desktop. See Running Applications.
- The **Preview File** button ( ) to open a data file using the default application for its file type. See Previewing Files.
- The **Compare** button ( ) to compare models in the **Comparison Result** window. See Comparing Models.

- The **Set Tags** button ( ) to set the assigned tags of models, data files, or other tags. See Assigning Tags.
- The **Delete** button ( ) to delete items. This action is not permanent and can be reverted. See Deleting Items.

THE LAYOUT SECTION

The **Layout** section contains the following functionality for opening and rearranging windows in the Model Manager workspace:

- The **Windows** menu ( ), for opening windows that are closed by default.
- The **Reset Desktop** button ( ), to reset the desktop layout to its default state. This will close all windows except The Model Manager Window and The Settings Window.

The Database Toolbar

The **Database** toolbar contains buttons for tasks common to a database shared between multiple users — that is, a server database accessed via a Model Manager server.

THE DATABASE SECTION

This section contains the same **Activate Database** button and **Add Database** button also available in the **Database** section on The Home Toolbar. The **Databases** button ( ) toggles the visibility of The Databases Window.

THE REPOSITORY SECTION

This section contains buttons for Advanced Version Control functionality:

- The **Repository** button ( ), to add a new repository in a database. See Adding Repositories.
- The **Branch** button ( ), to create a new branch from an existing branch, snapshot, or commit. See Creating a New Branch.
- The **Snapshot** button ( ), to record a point-in-time snapshot of all items with respect to a particular commit. See Recording Snapshots.
- The **Merge** button ( ), to merge modifications of items with respect to a source location into a target branch. See Merging.

THE PERMISSIONS SECTION

The **Permissions** section contains buttons for controlling access to various objects in the database, including setting ownership and permission requirements:

- Click the **Owner** button ( ) to set the user that owns a database object. A single user may be set as the owner of multiple database objects at once by first selecting all target objects in a window. See Transfer Ownership.
- Click the **Permissions** ( ) button to set the permission requirements for a database object. See Granting Permissions.
- Click the **Permission Template** ( ) button to create a new template of predefined permission requirements that can be reused for database objects. See Creating your own Permission Templates.

THE USERS SECTION

This section contains buttons for managing users and groups:

- The **User** button ( ), to add a new user in the database. See Adding Users.
- The **Group** button ( ), to add a new group in the database. See Adding Groups.

THE STATISTICS SECTION

This section contains a **Disk Space Usage** menu ( ) with options for estimating the disk space usage of model and file versions — see Estimating Disk Space Usage. The possible options are:

- **Version Selection** ( ) — Estimate the disk space usage of the selected versions.
- **Item Selection** ( ) — For the current selection, estimate the disk space usage of all versions belonging to the same items or the items' drafts.
- **Search Result** ( ) — Estimate the disk space usage of all versions matched by the current search in the **Model Manager** window. Only available for the **All Versions in Database** ( ) search mode.

THE MAINTENANCE SECTION

This section contains an **Add to Maintenance** menu ( ) with options for adding model and file versions to The Maintenance Window. The possible options are:

- **Version Selection** ( ) — Add the selected versions in the **Model Manager** window to the **Maintenance** window. Only available for the **All Versions in Database** ( ) search mode.

- **Item Selection** ( ) — For the current selection, add all versions belonging to the same items or the items' drafts to the **Maintenance** window.
- **Search Result** ( ) — Add all versions matched by the current search in the **Model Manager** window to the **Maintenance** window. Only available for the **All Versions in Database** ( ) search mode.

The Maintenance Toolbar

The **Maintenance** toolbar contains menus with options for performing maintenance operations targeting model and file versions added to the **Maintenance** window. The toolbar is only visible when the **Maintenance** window is opened.

THE STATISTICS SECTION

This section contains a **Disk Space Usage** menu ( ) with options similar to those available in the **Statistics** section on The Database Toolbar.

- **Version Selection** ( ) — Estimate the disk space usage of the selected versions.
- **Item Selection** ( ) — For the current selection, estimate the disk space usage of all versions belonging to the same items or the items' drafts.
- **All in Maintenance** ( ) — Estimate the disk space usage of all versions currently targeted in the **Maintenance** window, irrespective of any selections.

THE COMPUTED DATA SECTION

This section contains a **Clear Computed Data** menu ( ) with options for clearing built, computed, and plotted data of model versions in the **Maintenance** window — see Built, Computed, and Plotted Data. The possible options are:

- **Version Selection** ( ) — Clear built, computed, and plotted data of the selected model versions. Clearing cannot be undone.
- **Item Selection** ( ) — Clear built, computed, and plotted data of all versions belonging to the selected models or their drafts. Clearing cannot be undone.
- **Drafts of Item Selection** ( ) — Clear built, computed, and plotted data of all versions belonging to the selected models' drafts. Data belonging to the selected models themselves will not be affected. Clearing cannot be undone.
- **All in Maintenance** ( ) — Clear built, computed, and plotted data of all model versions currently targeted in the **Maintenance** window, irrespective of any selections. Clearing cannot be undone.

THE VERSIONS SECTION

This section contains a **Permanently Delete** menu (✕) with options for permanently deleting model and file versions in the **Maintenance** window — see Built, Computed, and Plotted Data. The possible options are:

- **Version Selection** ( ) — Permanently delete the selected versions. This deletion cannot be undone.
- **Item Selection** ( ) — Permanently delete all versions belonging to the selected items or their drafts. This deletion cannot be undone.
- **Drafts of Item Selection** ( ) — Permanently delete all versions belonging to the selected items' drafts. The selected items themselves will not be deleted. This deletion cannot be undone.
- **All in Maintenance** ( ) — Permanently delete all versions currently targeted in the **Maintenance** window. This deletion cannot be undone.

The Model Manager Workspace Windows

When you open the Model Manager workspace, you will see two windows in the COMSOL Desktop by default:

- The Model Manager Window — used primarily to search for models and data files.
- The Settings Window — used to show settings for various database objects, as well as update and save them in the database. See also Overview of a Model Manager Database.

You can always restore the COMSOL Desktop to this layout by clicking the **Reset Desktop** button ( ) in the **Layout** section of The Home Toolbar. From the **Windows** menu ( ) in the same section, you can open the following optional windows in the workspace:

- The **Material Browser** window — used to browse materials in your configured material libraries.
- The **Application Libraries**, **Part Libraries**, and **Add-in Libraries** windows — used to find MPH-files in your configured libraries for applications, geometry parts, and add-ins, respectively.
- The **Java Shell** window — used to run Java® code interactively.
- The **Chatbot** window — used to have a conversation with a chatbot.
- The **Data Viewer** window — used, for example, to inspect local variables in the **Java Shell** window.

- The **Comparison Result** window — used to compare model versions saved in the database. See also Comparing Models Saved in Databases.
- The Databases Window — used to browse, organize, and administer your configured databases via The Databases Tree.
- The Commits Window — used to view the commit history of a branch, optionally filtered to only include commits that involve a particular item.
- The Versions Window — used to view the version history of an item with respect to a branch.
- The Maintenance Window — used to perform maintenance operations for item versions.
- The References Window — used to view relations between model and file versions. See also The Auxiliary Data Window for Database Input and Output.

The **Model Manager**, **Databases**, **Commits**, **Versions**, **Maintenance**, and **References** windows play a central role in the Model Manager workspace and for its toolbars — whichever has focus determines the target for the buttons in The Home Toolbar and The Database Toolbar. You may, for example, have selected one model in the **Databases** window and another model in the **Model Manager** window. The window that has focus determines which model is opened if you click the **Open** button ( ) in the **Item** section of the **Home** toolbar. You can identify the window currently in focus by its title having a brighter color than the other windows.

The selection in the **Model Manager** window, the **Databases** window, and the **Maintenance** window also determines what is shown in the **Commits**, **Versions**, and **References** windows, again depending on which one of these three former windows has focus. If you select a model in the **Model Manager** window when searching in a branch, for example, you will see commits involving that model in the **Commits** window, a version history for the model in the **Versions** window, and all item versions that the model reference in the **References** window. Moreover, the selection in any of these six windows determines what is shown in the **Settings** window.

You can turn off the automatic selection linking in the **Commits**, **Versions**, **References**, and **Settings** windows by clicking the **Link with Selection** button ( ) in their respective toolbars. Click once more to turn on the linking.

The selection in the **Databases** window determines the searched location in the **Model Manager** window, as well as the selected branch in the **Commits** and **Versions** windows. If you select a branch tree node in The Databases Tree, for example, the **Model Manager**

window will automatically switch to search with respect to that branch. Similarly, if you change the current location via The Select Location Dialog in the **Model Manager** window, the **Commits** window, or the **Versions** window, the corresponding tree node is automatically selected in The Databases Tree and the current locations in the other windows are updated accordingly.

Overview of a Model Manager Database

You will encounter several concepts and terms when working with Model Manager databases that are specific to the Model Manager tools. This section contains a guide to these concepts and terms — you may want to skip ahead to the next section, Browsing Databases, and refer back to this section as needed. See also the Glossary.

In this section:

- Models
- Files
- Tags
- Items
- Commits
- Branches
- Snapshots
- Locations
- Repositories
- Users
- Groups
- Permission Templates

A common feature of the objects described in this section is that they all have an underlying key that uniquely identifies them in the database. Unlike, for example, changing the filename of a file on the file system, you can safely change the label, name, and title of any database object without worrying about, for example, that other objects lose references they might have to the renamed object.

Models

You can create a ***model*** in a Model Manager database by, for example, saving the model opened in the COMSOL Desktop or by importing a model directly from the file system.

Every time you save a model, a new ***model version*** is created. This does not mean that a full copy of the model is saved anew in the database — Model Manager is able to reuse any data it finds unmodified between versions. This includes the model tree itself, any binary data used for geometries, meshes, solutions, and results, as well as any other

data that the model may use. Once saved, a model version cannot be modified (except for clearing generated data that can be recreated from the model, that is, Built, Computed, and Plotted Data). You can be confident that what you save to the database will always be returned when opening the model version again, given that you open it in the same COMSOL Multiphysics version as the model version was originally saved in.

There is a subtle distinction between a model and all the versions of the model in a database. You set Access Control on the models themselves, while you open, save, search, and organize versions of the models. The same remark applies to any of the Items in a Model Manager database.

- Saving Models to Databases
- Importing Files

ITEM VERSION TYPES

Versions of models come in three different *item version types*:

- **Model** (), which is the standard type obtained, for example, when creating a new model from the Model Builder.
- **Application** (), which is a model that also has an application UI as defined in the Application Builder.
- **Physics** (), which is the type obtained when saving from the Physics Builder.

You may notice that a model can start out as the first type for its first couple of versions, but then transition into the second type once you save a version in which you have added an application UI. The opposite transition will occur if you save a version in which you have removed the application UI.

ITEM SAVE TYPES

Models can be created in the database as two different *item save types*:

- **Regular** (represented by one of the icons for Item Version Types)
- **Draft** ()

Unlike Item Version Types, the item save type of a model is fixed for all its versions.

Regular Models

A regular model is one that has been created in the database by saving a new model via the **Save** window or by importing a model directly from the file system. You can think of a regular model as the main result of your modeling work. Each version represents a clear transition in which you made enough progress that it is worth keeping the version around for future reference. Perhaps you want to go back to one of the older model versions and from that create a completely new model (with its own set of model versions), or perhaps the model version that you save corresponds to a step in which your modeling work is completed, and it is time for your coworker to take over.

Drafts

A draft model is one that has been created by going to the **File** menu and selecting **Save Draft** (), or by pressing Ctrl+S, when a regular model is opened in the COMSOL Desktop.

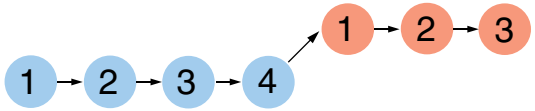
Saving Drafts of Models

A draft model is version controlled in the same way as a regular model: The first time you select **Save Draft**, a new draft model is created in the database. Selecting **Save Draft** after that will save new versions of that *same* draft model. You may think of a draft as an *ancillary* model used for saving intermediate changes, without muddying the version history of the main regular model.

The draft is automatically set with a **Private** permission template when you create a new draft in a server database accessed via a Model Manager server — see Predefined Permission Templates. Only you, as its owner, will be able to open or save the draft unless you change its permissions, although other users may still see it in search results.

The database stores a reference between the draft and the version of the regular model the draft originated from. This is analogous to how regular models created from existing models via **Save as New** in the **Save** window remembers their origin — see Splitting a Model Version History in Two. The stored reference enables Model

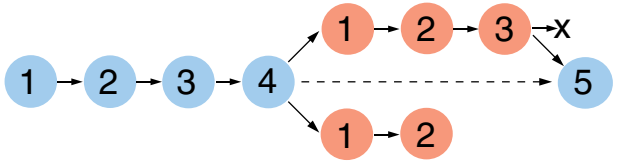
Manager, for example, to automatically discover if a newer version has been saved of the regular model *after* the draft was created — see Save Conflicts.



A schematic of a possible history of a regular model and a draft model created from the former. Starting from the left, four model versions of a regular model have been saved. A draft was then created from the fourth version, after which three versions of the draft have been saved.

Once you are finished with your draft, you can open the **Save** window and save the current draft as a version of the original model. The draft will be automatically deleted in the process, although this deletion is not permanent — you can, for example, find the deleted draft via the **Settings** window for the corresponding commit, see Commits, or when Searching All Versions in the Database.

You can create multiple drafts from the same regular model, each draft having its own identity and version history, by repeatedly opening a version of a regular model and pressing Ctrl+S. You and your coworker may, for example, both start working on your own drafts of the same regular model, perhaps with the intention of combining your work once you are finished using the **Comparison Result** window.



A schematic of a possible history of a regular model and two drafts models saved in a database. The middle row starting from the left represents four model versions of a regular model. From the fourth model version, two drafts (bottom and top rows) have been independently created. Two versions of the bottom row draft, and three versions of the top row draft, have been saved. The top row draft's latest version has then been saved back to the original regular model as a fifth version, automatically deleting the top row draft in the process (represented by the x-mark). The bottom draft may have been discarded, awaiting, for example, its owner to manually delete it.

MODEL SETTINGS

The **Settings** window for a model shows settings for a specific version of the model. Update any of the settings and click the **Save** button ( ) to save a new version of the model. You can write an optional save comment. Click **OK** to save.

The Version Section

This section displays the following fields:

- **Location**. The commit location in the database in which the model version is saved. See Locations.
- **Saved**. The time when the model version was saved.
- **Saved by**. The display name of the user that saved the model version.
- **Saved in**. The COMSOL Multiphysics version that the model version was saved in.
- **Title**. The title of the model in the saved version.

  The title is the same as shown in the **Title** field in the **Presentation** section in the root node's **Settings** window for a model opened in the COMSOL Desktop.
- **Filename**. The filename used by the model version if exporting it to the file system.
- **Update from**. An optional field in which the path to a file on the file system can be specified to save a new version of the model from that file.

  a Click the **Browse** button.

  b Select a file on the file system and click the **Open** button.

  c Click the **Save** button ( ).

  You can update from any model file that can be opened in COMSOL Multiphysics. The saved model version will first be converted to the current COMSOL Multiphysics version.
- **Description**. The description of the model in the saved version.

  The description is the same as shown in the **Description** field in the **Presentation** section in the root node's **Settings** window for a model opened in the COMSOL Desktop.

The Contents Section

This section displays the model tree as it looked when the model version was saved to the database. Use the **Collapse** button ( ) and **Expand** button ( ) to collapse and expand nodes in the tree.

Select a node in the tree and click **Details** ( ) in the toolbar below the tree to open the **Details** dialog. The dialog contains node field values and setting field values for the

selected model tree node. The **Node** table in the dialog shows the values for the properties of a node. The available properties will differ between node types. The **Settings** table shows settings for the node.

You can find models in the database by searching on the node properties and settings shown in the **Details** dialog for a model tree node. See Content Filters and The Model Manager Search Syntax for further details.

Select a node and click **Open Node** ( ) to open the model in the COMSOL Desktop with the model tree node automatically selected in the user interface.

Some node types support being inserted into the model currently opened in the COMSOL Desktop. Select, for example, a component or a geometry part node in the model tree and click **Insert into Model** ( ). You will be asked to select a target parent node for the insertion if there is more than one target available in the opened model. See Inserting Contents from Models for further details.

You can also select one or more nodes and click **Copy** ( ) or press Ctrl+C to copy the current selection to clipboard. The copied nodes can be pasted into the tree of the model opened in the COMSOL Desktop — right-click a node in the Model Builder, Application Builder, or Physics Builder tree and select **Paste** ( ) or press Ctrl+V. Not all nodes in the **Contents** section can be copied and not all nodes in the model can be the target for a paste. The tree being copied from and the tree being pasted to can, however, belong to different COMSOL Multiphysics program sessions.

The Tags Section

This section displays the Tags assigned to the model as a collection of tag pills.

See also The Tags Section for Items.

Files

You can import any type of file into a Model Manager database. A file that is not recognized as a COMSOL Multiphysics file (that is, with the file extension `mph`) or a Physics Builder file (that is, with the file extension `mphphb`) is referred to as a *data file*, or just *file*, in a Model Manager database.

Files are version controlled in exactly the same way as Models. For example, when you import from the file system, a new file is created in the database with an associated first version. You can update an existing file in the database by selecting a file on the file system and saving it as a new version from the File Settings window. You can also, for example, update a file in the database by using it as an output target for auxiliary data used by a model.

- Loading and Saving Auxiliary Data Files Stored in Databases
- Importing Files

You may wonder why COMSOL Multiphysics simulation models and data files are two different concepts in a Model Manager database, when they are all "just files" when stored on a file system. The main reason for the distinction is their respective storage characteristics and supported search functionality. The known internal structure of a model enables efficient data reuse between versions, as well as searching deep within a model's content using The Model Manager Search Syntax. A data file is stored in the database as a chunk of binary or text data whose contents is opaque to Model Manager.

ITEM VERSION TYPES

Versions of files come in two different *item version types*:

- **File** (), which is the standard type obtained when the version consist of just a single file when stored on the file system. Examples include a text file containing interpolation function data or a movie file for a results animation.
- **Fileset** (), which is a version consisting of multiple binary or text files when stored on the file system. Examples include CAD data consisting of a main CAD assembly file and one or more external component files that the assembly reference, or an HTML report consisting of a main HTML document and one or more image files referenced by the document.

You may notice that a file can, for example, start out as the first type for its first couple of versions, but then transition into the second type if you save a version that contains multiple files.

Whether or not multiple data files stored on the file system should be saved as separate files or as a single fileset in a Model Manager database is best answered depending on if the data is naturally version controlled as a "collective whole" or not. Prefer assigning Tags if all you want to accomplish is some organizational grouping in your database.

An individual binary or text file that belongs to a file version is referred to as a *file resource* in order to distinguish it from the version-controlled file itself, with the latter having additional metadata such as a title, a description text, and possible tag assignments — see File Settings. A fileset is thus a file version containing multiple file resources.

FILE SETTINGS

The **Settings** window for a file shows settings for a specific version of the file. Update any of the settings and click the **Save** button () to save a new version of the file. You can write an optional save comment. Click **OK** to save.

The Version Section

This section displays the following fields:

- **Location**. The commit location in the database in which the file version is saved. See Locations.
- **Saved**. The time when the file version was saved.
- **Saved by**. The display name of the user that saved the file version.
- **Title**. The title of the file in the saved version.
- **File size**. The size of the file version when stored on the file system. For a fileset, this is the sum of the individual file sizes.
- **Description**. The description of the file in the saved version.

The title is automatically set to the filename when importing a file from the file system into a Model Manager database. You are free to change the title to something else, however — the unique identity of the file itself in the database will not be affected.

The Contents Section

This section shows a table with all file resources associated with the file version. The table has a **File** and **Size** column for the filename and file size respectively of each file resource. A file version with zero or multiple file resources is a fileset; a file version with a single file resource is a plain file.

File resources in a file version may be organized into a hierarchy of *directories*. A directory is shown with a triangle symbol next to its name in the **File** column. Click the triangle to list the file resources and, possibly, subdirectories found inside that directory. File resources found on the top level in the table are thought of as belonging to an implicit *root directory*.

Click the **Add** button (+) to add a file from the file system as a new file resource to the table. If either no file resource or a file resource on the top level is currently selected in the table, the new file resource will also be added to the top level. To add a file resource under an existing directory, either select that directory or a file resource found inside the directory. A confirmation dialog for replacing is shown if an existing file resource with the same filename already exists.

You can also add an entire folder from the file system to the table. Click the expand button and select **Add Folder** ( ) in the menu. The folder itself, as well as all files and subfolders inside the folder, are added to the table at the position determined by the current selection in the same way as a single file is added.

Click the **Remove** button ( ) to remove selected file resources and directories from the table. Click the **Replace** button ( ) to replace a selected file or a directory in the table with a file or directory from the file system. Only the contents will be updated — the replaced file resource or directory will keep its current name.

	If you have a license for the CAD Import Module available when adding or replacing with a CAD assembly file, an attempt will be made to automatically resolve and include external component files that the assembly references.

	A directory hierarchy of a file version is only stored implicitly in the database via the relative paths, with respect to the implicit root directory, of the file resources. No information is stored about the directories themselves. As a consequence, an empty directory selected via **Add Folder** ( ) is ignored by Model Manager.

A fileset with a deeply nested hierarchy is expected to be rare. The prime example of a fileset with a hierarchy is an HTML report with a main HTML file at the top and associated images inside a folder.

Select a file resource and click **Preview File** ( ) to open the file resource with the default application for its file type. The preview will include any locally modified file resource in the table.

The Tags Section

This section displays the Tags assigned to the file as a collection of tag pills.

See also The Tags Section for Items.

Tags

A *tag* ( ) is used to label and organize models, files, and even other tags in the database — see Assigning Tags to Items to learn more. Tags can be created, for example, by importing them from folders on the file system or by manually creating new tags from within the Model Manager workspace.

- Importing Files
- Creating New Tags

Similar to Models and Files, tags are version controlled in the database. Whenever you save a tag in the database — for example by giving it a new title — a new version of the tag is created. Changing the title of a tag does not mean that other items lose their tag assignment — you will still be able to find items with the tag assigned using the new title. Unlike versions of models and files, however, versions of tags do not have any underlying content associated with them.

TAG SETTINGS

The **Settings** window for a tag shows settings for a specific version of the tag. Update any of the settings and click the **Save** button ( ) to save a new version of the tag. You can write an optional save comment. Click **OK** to save.

The Version Section

This section displays the following fields:

- **Location**. The commit location in the database in which the tag version is saved. See Locations.
- **Saved**. The time when the tag version was saved.
- **Saved by**. The display name of the user that saved the tag version.
- **Title**. The title of the tag in the saved version.

The Tags Section

This section displays the parent tags assigned to the tag as a collection of tag pills.

See also The Tags Section for Items.

Items

Models, Files, and Tags are collectively referred to as *items* in a Model Manager database. Items share many common features and functionality, including:

- Items are version controlled. Every time you save an item, a new version is created in the database.
- Items can be labeled and organized in the database by Adding and Removing Tag Assignments and by Organizing Items in Repositories.
- Items can be imported and exported from and to the file system via Bulk Operations.
- Item versions can be found via Searching and Filtering.
- Items support Advanced Version Control such as Branching, Reverting, and Merging.

ITEM SETTINGS

The **Settings** window for an item shows settings for a specific version of the item. Some settings are unique to a model, file, or tag, while others are shared by all items. Update any of the settings and click the **Save** button () to save a new version of the item. You can write an optional save comment. Click **OK** to save.

The Tags Section

All items have a **Tags** section in their **Settings** window that displays the Tags currently assigned to the item as a collection of tag pills. Click a tag pill and select **Remove** ( ) to remove a tag assignment.

Click the **Add Tag** button ( ) in the section's toolbar to find an existing tag to assign to the item. Write the title of the tag in the text field to filter the popup list of available tags. The matching tags are shown with their title followed by the titles of their parent tags, if any, within parentheses. Either double-click a tag in the list or select a tag and press Enter to assign it to the item.

Click the **Set Tags** button ( ) to open the **Set Tags** dialog. The dialog contains a tag tree of all available tags. Select a checkbox for a tag in order to assign the tag to the item. Clear a selected checkbox to remove a tag assignment. Click **Clear Tags** ( ) to clear all selections in the tree. You can filter the tree of available tags by writing a tag title in the text field above the tree. Click **OK** to finish the tag assignment.

Use the dialog opened via the **Set Tags** button ( ) on The Home Toolbar in the Model Manager workspace over the **Settings** window if you only want to update the assigned tags of an item without saving a new version — see also Adding and Removing Tag Assignments. That dialog also supports updating the assigned tags of multiple items at once.

Commits

A *commit* ( ) is a set of related changes made to Items — that is, Models, Files, and Tags — within a single database save operation. This includes anything from saving item versions, changing the assigned tags of items, or deleting items, to creating a new branch, merging into a branch, and reverting a commit. The changes are saved to the database as a "unit" and, as such, can also be *reverted* as a unit.

A commit saved in the database includes:

- The branch that the commit was saved to — see Branches.
- The time when the commit was saved.
- The user that saved the commit.
- An optional commit comment provided by the user.
- The set of related changes made to items in the commit.

See Figure 2-3 for an example of a commit taken from the tutorial Example: Modeling Using Version Control in which a draft has been saved back as a new version of its original model.

Given a particular commit, you can browse and search the versions of items that were the *latest versions* at the time of that commit. A schematic representation of this is shown in Figure 2-2: In the first commit, a first version of a model A and a model B were saved. In the second commit, a second version of model A was saved, a first version of a tag T was saved, and model B was tagged by T. In the third commit, model A was deleted and a second version of model B was saved. The database can be browsed and searched with respect to each of the three commits, with each big circle surrounding what you will find.

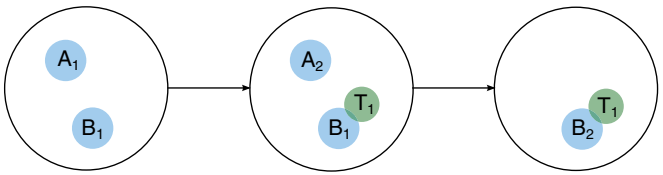


Figure 2-2: A schematic of three hypothetical commits (big circles) saved in a database. You can browse and search versions and tag assignments with respect to any one of these commits.

A synonym to commit often found in other version control systems is *revision*.

- Locations
- Basic Version Control
- Searching Versions

COMMIT SETTINGS

The **Settings** window for a commit shows the branch the commit was saved to, the point in time when the commit was saved, the user that saved the commit, and any comments provided with the commit. You will also find a table with the set of changes made to Items in the commit.

You can update the comments for the commit. Rewrite the text as you see fit and click the **Save** button ( ).

For a Model Manager server database, only an administrator or the user that saved the commit can update the comments.

The General Section

This section displays the following fields:

- **Location**. The database, repository, and branch that the commit was saved to.
- **Date**. The point in the time when the commit was saved.
- **User**. The name of the user that saved the commit.
- **Comments**. The optionally provided comments when the commit was saved.

The Changes Section

This section shows a table with the set of related item changes made in the commit. Each change entry in the table is represented by the latest version of the affected item at the time of the commit. Apart from changes corresponding to saving a new version, this also includes deletions and tag assignments of items.

From the toolbar below the table, you can:

- Click the **Open** button ( ) to open a model version. Only enabled for a change that involved a model.

- Click the **Preview File** button ( ) to open a file version using the default application for its file type. Only enabled for a change that involved a file.
- Click the **Version Details** button ( ) to open The Version Details Dialog containing more information on a version.

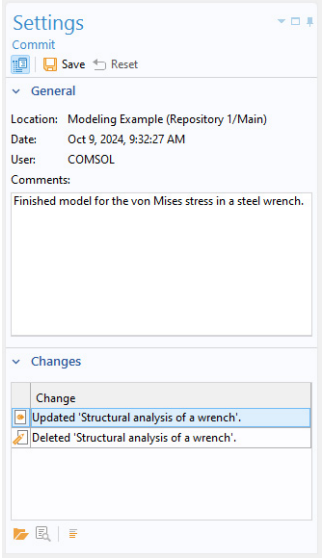


Figure 2-3: The Settings window for a commit in which a new version has been saved of a model, and a draft has been deleted.

Branches

A *branch* ( ) is a chronologically ordered sequence of Commits saved in the database. The sequence of commits form a history of things that has happened to Items in the database including, for example, saved versions, tag assignments, and deletions. An important commit in a branch is the one saved most recently — that is, the *latest commit* on the sequence.

When browsing and searching in the branch, one finds the latest version of each item saved on the branch as well as the items' currently assigned tags. Two branches in the same repository will typically not return the same latest version or assigned tags for an item if work on the item is being done in parallel on the branches. The two branches can be synchronized by *merging* changes to items made in the first branch into the second branch and vice versa.

The branch itself is typically used as a representative in the Model Manager workspace to specify that you want search or browse the latest item versions in Model Manager:

- Expand a branch node ( ) in The Databases Tree to browse the latest versions of all items, given that **Items** is selected in the **Show** menu ( ) in The Databases Window Toolbar.
- Select a branch in The Select Location Dialog to search the latest model and file versions in the **Open**, **Select File**, **Select Model**, and **Model Manager** windows. This is also the default choice in these windows.
- Select a branch in The Select Location Dialog to export the latest versions from the **Export** dialog.

You can create a new branch from an existing parent branch by *branching off* from a particular source commit. This introduces an *alternative* commit history that runs in parallel with that of the parent branch — see Branching.

	When referring to the latest versions of items in a Model Manager database without any additional qualifications, it is always understood as being with respect to a particular branch.

	Creating a New Branch

DEFAULT BRANCH

An initial, default, branch ( ) is automatically created for a new repository. This is the branch that all save actions targeting that repository use by default.

You can change the default branch in case you have created multiple branches in a repository. Select the corresponding branch tree node in the **Databases** tree, right-click, and select **Set Default Branch** ( ).

BRANCH SETTINGS

The **Settings** window for a branch shows:

- **Database**. The label of the database that the branch belongs to.
- **Repository**. The name of the repository that the branch belongs to.

- **Name**. The name of the branch.
- **Search**. The type of item data that can be searched in the branch. Select **Item fields and content** in the list to enable the full Model Manager search functionality for the branch. Select **Only item fields** to restrict the filtering support to those that target the field values of models and files, not filters targeting their contents. See also Searching in Branches.

Clicking the **Save** button ( ) saves the **Name** and **Search** fields for the branch.

Snapshots

A *snapshot* ( ) is a reference to a particular commit on a branch.

- Commits
- Branches

The snapshot itself is typically used as a representative, or *recording*, of the item versions that were the latest at the time of the commit:

- Expand a snapshot node ( ) in The Databases Tree to browse the recorded item versions, given that **Items** is selected in the **Show** menu ( ) in The Databases Window Toolbar.
- Select a snapshot in The Select Location Dialog to search the recorded model and file versions in the **Open**, **Select File**, **Select Model**, and **Model Manager** windows.
- Select a snapshot in The Select Location Dialog to export the recorded versions from the **Export** dialog.

Recording Snapshots

SNAPSHOT SETTINGS

The **Settings** window for a snapshot shows:

- **Database**. The label of the database that the snapshot belongs to.
- **Repository**. The name of the repository that the snapshot belongs to.
- **Date**. The timestamp of the commit that the snapshot references.
- **Name**. The name of the snapshot.

Clicking the **Save** button ( ) saves the **Name** field for the snapshot.

Locations

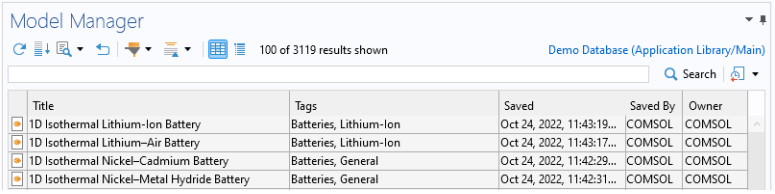
Commits, Branches, and Snapshots are all referred to as *commit locations*, or *locations,* in Model Manager. Each location is a commit in its own right or acts as a representative of a particular commit:

- **Branch**. The most recent commit saved on the chronologically ordered sequence of commits.
- **Snapshot**. The commit that the snapshot references.

A branch location is the most common, and useful, of the three types. By browsing and searching with respect to such a location, you are guaranteed to see the latest version of each item.

THE SELECT LOCATION DIALOG

You can select a branch or, depending on context, snapshot for a particular window via the **Select Location** dialog. You open the dialog by clicking on a *location link button*, seen here in the upper-right corner of The Model Manager Window:



A search result in the Model Manager window obtained for a specific branch. The matched item versions are the latest versions for their corresponding items and with respect to the selected branch location.

The text of a location link button is shown with the name of the branch or snapshot. A commit is shown with its date. Typically the label of the database and the name of the repository are also included in the text, such that the text format is `Database (Repository/Location)`.

In the tree in the **Select Location** dialog, expand nodes until you reach a branch or snapshot leaf node, select the leaf node, and click **OK**. You can also select a nonleaf node, in which case a suitable default location is automatically inferred based on the selection. If **Deleted Branches** or **Deleted Snapshots** has been selected in the **Show** menu

( ) in The Databases Window Toolbar, such branches or snapshots will also be available in the dialog.

Repositories

A *repository* ( ) is a container for a collection of items and their versions in the database. When the database is created, a first repository is automatically added for you.

You would typically add more repositories to the database if you want to restrict access to a collection of items for a particular set of Users and Groups. By Granting Permissions on the repository, you can, for example, restrict who is able to browse and search the item versions in the repository.

Each repository contains one or more branches. Whenever you add a new repository, an initial Default Branch in that repository is created.

Adding Repositories

DEFAULT REPOSITORY

If you have added multiple repositories to the database, Model Manager tries to remember the repository you last accessed in, for example, the **Open**, **Select File**, and **Select Model** windows. You can also set one of the repositories as the default repository ( ). Analogous to the Default Branch, this is the repository that all save actions targeting the database use by default.

The first repository created in the database is initially set as the default one. To change the default, select the corresponding repository tree node in the **Databases** tree, right-click, and select **Set Default Repository** ( ).

REPOSITORY SETTINGS

The **Settings** window for a repository shows:

- **Database**. The label of the database that the repository belongs to.
- **Name**. The name of the repository.
- **Default Branch Name**. The name of the repository's Default Branch.

Clicking the **Save** button ( ) saves the **Name** field for the repository.

Users

A *user* ( ) is someone who has connected to a Model Manager database. For a local database, the user is created with the user account name on your computer. For a server database, the user is created using the credentials you provide when connecting to the Model Manager server — see Connecting to a Server Database.

A user in a Model Manager database is primarily used to identify the individual that has saved a commit, the owner of a database object, or the individuals that have been granted permissions for a database object.

USER SETTINGS

The **Settings** window for a user shows:

- **Database**. The label of the database that the user belongs to.
- **Name**. The username of the user.
- **Display Name**. An alternative name used for display purposes.
- **Group Memberships**. The groups in the database that the user is a member of.

Clicking the **Save** button ( ) saves the display name and the group memberships for the user.

Group Memberships

Click **Add** to add the user as a member to a group in the database. In the **Search** dialog, type a name or display name for groups and click the **Search** button. Select groups in the search result table and click **OK**.

Select groups in the **Group Memberships** list and click **Remove** to remove group memberships from the user.

Click the **Save** button ( ) to save any changed group memberships to the database.

Managing Users

Groups

A group ( ) is a collection of users and other groups. You typically create groups to more easily manage permissions common to several users in the database.

GROUP SETTINGS

The **Settings** window for a group shows:

- **Database**. The label of the database that the group belongs to.
- **Name**. The name of the group.
- **Display Name**. An alternative name used for display purposes.
- **Group Members**. The users, and other groups, in the database that are members of the group.

Clicking the **Save** button () saves the display name and the group members for the group.

Group Members

Click **Add** to find a user, or another group, in the database to add as a member to the current group. In the **Search** dialog, type a name or display name for users and groups and click the **Search** button. Select users and groups in the search result table and click **OK**.

Select users and groups in the **Group Memberships** list and click **Remove** to remove group members from the group.

Click the **Save** button () to save any changed group members to the database.

	Group membership is transitive: if a user is a member of a group, which in turn is a member of another group, the user is considered a member of the latter group as well.

	Managing Groups

Permission Templates

A *permission template* () is a saved list of permission assignments for a set of users and groups that you can apply to database objects. You can, for example, use a permission template to reuse the same permissions for a large collection of models, only having to update in one place — the permission template itself — in case you want to change these permissions.

Model Manager comes with three Predefined Permission Templates, descriptively named Public, Protected, and Private, for each of the database objects that you can control access to. You can also create custom permission templates for Models and Files — see Creating your own Permission Templates.

Reusing Permission Assignments Using Permission Templates

PERMISSION TEMPLATE SETTINGS

The **Settings** window for a permission template shows:

- **Database**. The label of the database that the permission template belongs to.
- **Name**. The name of the permission template.
- **Type**. The object type that the permission template can be applied to. Either **Model** or **File**.
- A table containing the permission assignments of the permission template.

Clicking the **Save** button ( ) saves the **Name** field and permission assignments for the permission template.

Permission Assignments

Click **Add** to add permissions for a user or group to the permission template. In the **Add** dialog, type a name or display name for users and groups and click the **Search** button. Select the user or group you want to add in the search result table. You can also select the special **Everyone** or **Owner** options — see Everyone and Owner. At the bottom of the dialog, select the permissions to assign the selection. Click **OK** to add the permission assignment.

Select a row in the permission assignment table and click **Edit** to change the permissions for the selection. Click **Remove** to remove the selection from the table.

Click the **Save** button ( ) to save any changed permission assignments to the database.

Browsing Databases

For models and data files stored on the file system, there is typically only one way to browse them — what you see is what you get. Taking the same approach in a Model Manager database could, however, quickly become confusing and hard to navigate — models with hundreds of versions may take up an entire search result, making models with only a few versions harder to find. As the title of a model rarely changes, you would also have to meticulously compare dates when opening a model version to know you are working off the most recent.

In this section, you will learn how various windows in the Model Manager workspace can be used to browse and search in useful subsets of model and file versions in your databases — including browsing and searching the latest versions of items, listing the version history of a particular item, listing the history of all changes made to items, as well as navigating the relationships between different versions.

- The Model Manager Window
- The Databases Window
- The Settings Window
- The Commits Window
- Activating a Database
- The Versions Window
- The References Window
- Opening Models
- Running Applications
- Previewing Files
- Comparing Models
- Copying Model and File Locations

The Model Manager Window

Use the **Model Manager** window to find versions of Models and Files by writing search expressions in the search field and clicking the **Search** button ( ) or pressing Enter. You can write plain search words and any number of filter expressions using The Model Manager Search Syntax. Plain search words will match on the titles, descriptions, assigned tags, and filenames of versions. Press Ctrl+Space to get completion assistance when writing filter expressions — see Search Syntax Completion.

You can also apply separate Item and Content Filters via The Filter Dialog. Select a filter from the **Add Filter** menu button ( ) in the toolbar to open the dialog. Applied filters are shown below the search field — see Applied Filter Pills.

The searched location is initially set to the Default Branch in the Default Repository, which means searching for the latest versions of models and files with respect to this branch. Click the top-right link button above the **Search** button ( ) to select another location to search via The Select Location Dialog.

To search among all versions ever saved in the database, click the expand button next to the **Latest Versions for Location** menu button ( ) and select **All Versions in Database** ( ). You can also toggle between these two search modes by clicking the menu button itself — see Searching Versions for more details. To redo a previous search, select one of the entries in the **Search History** menu list ( ) — see also Search History.

You can switch the presentation of the search result between either a Table View or a Tree View when **Latest Versions for Location** ( ) is selected — the former primarily used when you want to quickly find a particular item and the latter when you want to browse a larger collection of items. Only the Table View is available when **All Versions in Database** ( ) is selected.

You can change the sort field and sort order of the search result from the toolbar. Click the expand button next to the sort field button to select another sort field than the default **Title** ( ) sort field — see also Sorting Search Results. The selected sort field and sort order for the **Model Manager** window are remembered between program sessions.

	Searching and Filtering

	If Model Manager detects that a commit has been saved on a searched branch, an information message ( ) is displayed at the bottom of the **Model Manager** window. This includes commits made by you from within the COMSOL Desktop as well as commits made by your coworkers if connected to a server database. Either click the information message text itself or the **Refresh** button ( ) to see any updated results.

THE MODEL MANAGER WINDOW TOOLBAR

The toolbar in the **Model Manager** window contains the following toolbar buttons:

- Click the **Refresh** button ( ) to refresh the search result while keeping the search expression and applied filters unchanged.

- Click the **Show More** button ( ) to include more matching models and files in the search result. The button is disabled when the Tree View is shown for a branch.
- Click the **Search History** menu button ( ) to select a previous search that you want to redo.
- Click the **Reset** button ( ) to clear the current search expression and applied filters.
- Click the **Add Filter** button ( ) to apply a filter.
- Click the sort field button to toggle the sort order of the search result between ascending and descending order. Click the expand button to select another sort field.
- Click the **Table** button ( ) to view the search result in a table. The button is disabled when **All Versions in Database** ( ) is selected.
- Click the **Tree** button ( ) to view the search result in a tree. The button is disabled when **All Versions in Database** ( ) is selected.

TABLE VIEW

The *table view* of the search result shows matching model and file versions in a table. For the **Latest Versions for Location** ( ) search mode, the columns are:

- The type column — the type of the model or file represented by an icon. See Item Version Types and Item Save Types.
- The **Title** column — the title set for the model or file in that version.
- The **Tags** column — the tags set for the model or file with respect to the searched location.
- The **Saved** column — the time when the version was saved.
- The **Saved By** column — the name of the user that saved the version.
- The **Owner** column — the name of the user that owns the model or file.

For the **All Versions in Database** ( ) search mode, the columns are:

- The type column — the type of the model or file represented by an icon. See Item Version Types and Item Save Types.
- The **Title** column — the title set for the model or file in that version.
- The **Saved** column — the time when the version was saved.
- The **Saved By** column — the name of the user that saved the version.
- The **Owner** column — the name of the user that owns the model or file.

- The **Repository** column — the repository the version was saved in.
- The **Branch** column — the branch the version was saved in.
- The **Comments** column — the optional comment provided when the version was saved.

A maximum of 100 versions that matched the search are initially included in the table. Click the **Show More** button ( ) in the toolbar to append the next 100 matches to the list. You can set another value for this default page size in the **Result Page Size** field on the **Model Manager** page in the **Preferences** window.

Using the Table View is recommended when you expect to find a small number of models and data files for your search expression and applied filters.

TREE VIEW

The *tree view* of the search result shows matching model and file versions under their assigned tags in The Tag Tree. Given that an item may be assigned multiple tags, you can encounter the same model or file in multiple positions in the tree. Each model or file tree node shows the title set for the version, the time when the version was saved, and the name of the user that saved the version. The appearance and behavior of the tree differ somewhat depending on if you are searching item versions with respect to Branches or with respect to Snapshots or Commits, with the former typically offering a richer experience — see also Searching Latest Versions for Locations.

Using the Tree View is recommended when you expect to browse through a large search result of potentially thousands of items.

Searching in Branches

When searching in a branch, the tree view shows a count of the number of matching models and files found under each tag node within parentheses next to the tag's title. If you have created a large number of tags in your database, the count may be shown first when you expand a tag node (with ellipses shown within parentheses until then).

If you have not written a search expression in the search field or added any filters, tags that are not assigned to any items are also included in the tree. This enables you to browse through all items in The Tag Tree — tags included. If there is a search expression or applied filter, tags are excluded from the tree if the count is known to be zero.

A maximum of 100 versions that matched the search are initially included under a tag tree node. When more results are available under a tag node, a **Show More** tree node ( ) can be expanded to reveal the next 100 versions. You can set another value for this default page size in the **Result Page Size** field on the **Model Manager** page in the **Preferences** window.

Searching in Snapshots and Commits

When searching versions in a snapshot or commit, the tree view shows the first 100 matching versions (sorted on title), irrespective of their assigned tags. You can append more search results to the tag tree until all matching versions have been fetched by repeatedly clicking the **Show More** button ( ) in the toolbar.

There is a subtle difference between the tag tree shown in the tree view of the **Model Manager** window when searching in Snapshots and the corresponding tag tree shown in The Databases Tree. For the former tree, you only see the tags assigned to models and files that are included in the search result. This means that more and more tags may show up when you repeatedly click the **Show More** button ( ) in the toolbar. Tags that are not assigned to any models or files will never show up. For the latter tree, you always see all tags at a fixed level when you expand a tree node.

The Databases Window

The **Databases** window shows a tree with all databases you have added in the COMSOL Desktop. You typically use the tree to browse and administer the content of databases, including, for example, edit Database Configurations, add Repositories, create Branches, record Snapshots, and, for server databases, manage Users and Groups.

Click the **Step In** button ( ) to show a subtree of the tree in the window. You can continue stepping into nodes until only leaf nodes are visible. Click **Step Back** ( ) to return one step, or **Step Home** ( ) to show the whole tree. You can toggle the visibility and appearance of various nodes in the **Show** menu ( ) and the **Item Tree Node Text** menu ( ).

TOGGLING THE DATABASES WINDOW

Click the **Databases** button ( ) in the **Database** toolbar to toggle the visibility of the **Databases** window.

THE DATABASES WINDOW TOOLBAR

The toolbar in the **Databases** window contains the following toolbar buttons:

- Click the **Step Home** button ( ) to show all nodes in the **Databases** window's tree; that is, to return to the default tree view after stepping into nodes.
- Click the **Step Back** button ( ) to show the nodes previously shown before stepping into a node.
- Click the **Step In** button ( ) to only show the child nodes, and their descendants, of a selected node in the **Databases** window's tree.
- Click the **Refresh** button ( ) to refresh the child nodes of the selected node.
- Click the **Show** menu ( ) to set visibility options for nodes.
- Click the **Item Tree Node Text** menu ( ) to set text options for nodes corresponding to models and files.

THE DATABASES TREE

The top nodes in the **Databases** window's tree are the Database Configurations of all databases that have been added to the COMSOL Desktop: local databases and server databases accessed via a Model Manager server.

The child nodes to the databases configurations consist of all Repositories you are authorized to see, as well as a **Security** node ( ) containing child nodes related to user management. The **Security** node is hidden for a local database.

Each repository node contains a **Branches** node ( ) and a **Snapshots** node ( ). The **Branches** node and the **Snapshots** node contain all Branches and Snapshots, respectively, you are authorized to see.

Each of the branch and snapshot nodes contain the latest versions of Items — that is, Models, Files, and Tags — with respect to the point in time of the corresponding commit. For a branch node, these are the latest versions at the present time, while for a snapshot, these are the versions that were the most recent ones at the time of the snapshot's referenced commit.

See Locations and Commits for a discussion on how a branch and snapshot are associated with a specific commit, and how that commit in turn identify a subset of latest versions in the database.

The items are further organized into a subtree based on their assigned tags with respect to the corresponding commit — see also The Tag Tree. You may think of the branch node or snapshot node as representing a root tag in this subtree.

Select **Items** in the **Show** menu button ( ) to see items under branch and snapshot nodes.

The items under a tag node are sorted on title, with a maximum of 100 models and files initially shown. When more items are available under a tag node, a **Show More** tree node ( ) can be expanded to reveal the next 100 items. You can set another value for this default page size in the **Result Page Size** field on the **Model Manager** page in the **Preferences** window.

The **Security** node contains a **Users** node ( ), a **Groups** node ( ), and a **Permission Templates** node ( ). Each one contains, respectively, all Users, Groups, and custom Permission Templates in the database.

Right-click a node and select **Refresh** ( ) to refresh all child nodes of that node. You may find this useful, for example, to see changes made in a server database from another COMSOL Multiphysics program session, perhaps run by your coworker.

Deleting and Restoring Unversioned Database Objects

You can delete Repositories, Branches, Snapshots, Users, Groups, and Permission Templates by right-clicking their corresponding tree nodes and selecting **Delete** ( ). This deletion is not permanent but rather marks these database objects as hidden in the COMSOL Desktop by default.

Clicking **Show** ( ) enables you to show deleted (hidden) nodes in the tree. You can right-click such nodes and select **Restore** ( ) to remove their hide-marker, making them visible again everywhere in the COMSOL Desktop. The options set via the **Show** toolbar button also determines the visibility of deleted repositories, branches, and snapshots in The Select Location Dialog.

Permission templates can also be permanently deleted. Right-click and select **Delete Permanently** ( ). This requires that the permission template is not assigned to any items.

The Settings Window

The **Settings** window shows settings for the database object selected in The Model Manager Window, The Databases Window, The Commits Window, The Versions Window, The Maintenance Window, or The References Window (depending on which window has focus). Click **Link with Selection** ( ) to disable this automatic linking. You can still update the **Settings** window with a new database object by right-clicking the object and selecting **Settings** ( ).

You use the **Settings** window to view and update an object stored in the database. To update the object, change any of its values in the **Settings** window's fields and click the **Save** button ( ). In contrast to the **Settings** windows in the Model Builder, the Application Builder, or the Physics Builder workspaces, changes made in the window are not automatically saved to the database. The **Reset** button ( ) will be enabled when there are unsaved changes to an object shown in the **Settings** window.

Remember to click **Save** ( ) to save any changes made to an object in the **Settings** window.

Model Manager will try to remember unsaved changes for an object during the program session: If you change some fields for a database object, select another database object in one of the other windows (so that the **Settings** window is updated), and then return to the first database object, your unsaved changes for the object will still be there. This enables you to save the changes to the database at your convenience. Model Manager will also show a warning dialog if you have unsaved changes when the **Settings** window is updated with a new database object.

You can disable the warning for unsaved changes: clear the **Warn if the Settings window contains unsaved changes** checkbox on the **Model Manager** page in the **Preferences** window.

When you click the **Save** button ( ) for a model, file, or tag, a dialog is shown in which you can provide an optional commit comment. Click **OK** to save a new version of the item. If you have made any changes to the tag pills in the **Tags** section, the

assigned tags of the item are also updated. Clicking **Save** for any other database object immediately saves the changes to the database.

For models, files, and tags, the **Save** button ( ) is only enabled when viewing the settings with respect to a branch location. Only then are you guaranteed to update from the *latest version* of the item when saving. The button is disabled when viewing settings with respect to a fixed commit, which is the case for a selection in the **Versions**, **Maintenance**, or **References** windows, or for a selection in the **Model Manager** window when searching all versions in the database.

- Model Settings
- File Settings
- Tag Settings
- Commit Settings
- Branch Settings
- Snapshot Settings
- Repository Settings
- User Settings
- Group Settings
- Permission Template Settings

THE SETTINGS WINDOW TOOLBAR

The toolbar in the **Settings** window contains the following toolbar buttons:

- Click the **Link with Selection** button ( ) to enable or disable whether to automatically update the **Settings** window based on the current selection in the **Model Manager**, **Databases**, **Commits**, **Versions**, **Maintenance**, or **References** window (which ever has focus).
- Click the **Save** button ( ) to write any changes made in the **Settings** window to the database.
- Click the **Reset** button ( ) to reload the shown object in the **Settings** window from the database, thereby discarding any pending changes you have in the window. The button is disabled if no changes have been detected.

The Commits Window

You use the **Commits** window to view the history of all Commits saved in a branch. Click the link button to select another branch in The Select Location Dialog.

The commits are shown in a table with a maximum of 100 most recent commits initially retrieved from the database. Click **Show More** ( ) to append the next 100 commits to the table. You can set another value for this default page size in the **Result Page Size** field on the **Model Manager** page in the **Preferences** window.

If the branch was created from another branch, older commits on the parent branch are also appended in the table. The table columns are:

- The **Date** column — the time when the commit was saved.
- The **User** column — the name of the user that saved the commit.
- The **Branch** column — the name of the branch that the commit belongs to.
- The **Comments** column — the optionally provided comments when the commit was saved.

Select a row in the table to see the corresponding Commit Settings in The Settings Window. This enables you to, for example, update the comments of the commit or see the set of related item changes saved in the commit.

The commit history shown in the table follows the current selection in The Model Manager Window or The Databases Window (depending on which window has focus). If you select a particular item in one of these windows, only commits involving that item are shown in the table. To see all commits in the corresponding branch, press Ctrl and click the selected item to deselect it or simply select a branch tree node in the The Databases Window.

Click **Link with Selection** ( ) to disable the automatic linking. You can still update the **Commits** window with a new database object by right-clicking the object and selecting **Commits** ( ), when available.

THE COMMITS WINDOW TOOLBAR

The toolbar in the **Commits** window contains the following toolbar buttons:

- Click the **Refresh** button ( ) to refresh the table to see any new commits that have been saved on the branch. The table will automatically refresh if you save a new commit to the branch from the COMSOL Desktop.
- Click the **Show More** button ( ) to append older commits to the table.

- Click the **Link with Selection** button ( ) to enable or disable whether to automatically update the **Commits** window based on the current selection in the **Model Manager** window or the **Databases** window (which ever has focus).
- Click the **Branch** button ( ) to open The Create Branch Dialog to branch off from the selected commit.
- Click the **Snapshot** button ( ) to record a snapshot for the selected commit.
- Click the **Merge** button ( ) to open The Merge Window to merge all changes made up to the selected commit to a target branch.
- Click the **Revert** button ( ) to open The Revert Window for the selected commit.

If you right-click a commit in the table, you can also

- Select **Export** ( ) to export all versions of items that were the latest versions at the time of the commit to the file system — see Exporting Items.
- Select **Search in Commit** ( ) to set the selected commit as the location in The Model Manager Window. You can then search for all versions of items that were the latest versions at the time of the commit. See also Searching in Snapshots and Commits.

When you click **Revert** ( ) for a commit, the inverse of the set of changes shown in The Changes Section of the **Settings** window can be applied from The Revert Window.

Activating a Database

You can select one of your configured databases from the menu in the **Database** section in the **Home** toolbar to *activate* it in the Model Manager workspace. This will:

- Automatically connect to the database if it was not already connected. You may be prompted for credentials if connecting to a server database.
- Set the location in The Model Manager Window and The Commits Window to the default branch in the default repository of the database.
- Select the default branch in the default repository in The Databases Tree.

The Versions Window

The **Versions** window in the Model Manager workspace shows the version history of an item — that, is a model, file, or tag — with respect to the latest version on a

particular branch. The version history entries correspond to a subset of Commits on the branch for which a new version of the item was saved. Click the link button in the upper right corner to select another branch in The Select Location Dialog. If the branch was created from another branch, older versions saved on the parent branch are also appended to the table. If the item was created from another item — via, for example, **Save as New** ( ) in the **Save** or **Export** windows — the version history of that item is appended as well.

As for The Versions Window for the COMSOL Desktop Model, the **Versions** window in the Model Manager workspace is used to visualize how one version has progressed to the next by going upward in the table until reaching the latest version on the selected branch. Versions saved on a parent branch *after* the selected branch was created are not included in the table by the assumption that they correspond to independent work done in parallel and whose changes are not reflected in the latest version.

You can see *all* versions of an item, irrespective of branch, by adding the item to The Maintenance Window in the Model Manager workspace. From the window, you gain a complete overview of the full footprint of the item in the database, but may lose a sense of how the item has evolved over time.

The version history shown in the window follows the current item selection in The Model Manager Window or The Databases Window (depending on which window has focus). Click **Link with Selection** ( ) to disable this automatic linking. You can still update the **Versions** window with a new item by right-clicking and selecting **Versions** ( ).

The versions are shown in a table sorted in chronological order. A maximum of 100 most recent versions are initially retrieved. Click the **Show More** button ( ) to append the next 100 versions to the table. You can set another value for this default page size in the **Result Page Size** field on the **Model Manager** page in the **Preferences** window. If a viewed model is opened in the COMSOL Desktop, the table row for the corresponding model version is highlighted in bold in the table.

The table columns are:

- The type column — the type of the item represented by an icon.
- The **Title** column — the title set for the item in that version.

- The **Saved** column — the time when the version was saved.
- The **Saved By** column — the name of the user that saved the version.
- The **Branch** column — the target branch of the commit in which the version was saved.
- The **Comments** column — the optional comment provided when the version was saved.

Select a row in the table to see the corresponding settings for the model, file, or tag in The Settings Window at the point in time the version was saved.

THE VERSIONS WINDOW TOOLBAR

The toolbar in the **Versions** window contains the following toolbar buttons:

- Click the **Refresh** button ( ) to refresh the table in case any new versions have been saved. The table will automatically refresh if you save a new version on the branch from the COMSOL Desktop.
- Click the **Show More** button ( ) to append older versions to the table.
- Click the **Link with Selection** button ( ) to enable or disable whether to automatically update the **Versions** window based on the current selection in the **Model Manager** window or the **Databases** window (which ever has focus).
- Click the **Version Details** button ( ) to open The Version Details Dialog containing more information on the version.
- Click the **Open** button ( ) to open a selected model version in the COMSOL Desktop.
- Click the **Run** button ( ) to launch and run a selected version in the COMSOL Desktop. Only enabled if the selected version is an application.
- Click the **Preview File** button ( ) to open a selected file version using the default application for its file type. See Previewing Files.
- Click the **Compare** button ( ) to compare a selected model version with the model opened in the COMSOL Desktop. Select two model versions to compare them with each other.
- Click the **References** button ( ) to open The References Window to view all references between the selected version and other item versions.
- Click the **Restore Version** button ( ) to save the selected version as a new latest version of the item whose version history is shown in the window. The target branch for the save is given by the currently selected branch in the location link button. See Restore Version for further details.

If you right-click a version in the table, you can also

- Select **Disk Space Usage** () to see an estimate of the required disk space usage if the version is saved to the file system. See Estimating Disk Space Usage.
- Select **Export** () to export the version to the file system — see Exporting Items.
- Select **Export to New Local Database** () to export the version to a new local database — see Export to New Local Database.
- Select **Branch** () to create a new branch from the commit that the version was saved in.
- Select **Snapshot** () to record a snapshot of the commit that the version was saved in.
- Select **Copy Location** () to copy a text string with a URI that uniquely identifies a model or file version in the database to the clipboard. See Copying Model and File Locations for what you can do with this text string.
- Select **Clear Computed Data** () to clear all built, computed, and plotted data of a model version. Data shared with other model versions via deduplication will not be deleted from the database. Clearing cannot be undone. See Built, Computed, and Plotted Data.
- Select **Permanently Delete** () to permanently delete a version in the database. Data shared with other versions via deduplication is not deleted. Permanently deleting cannot be undone. See Permanently Deleting Models and Data Files.
- Select **Settings** () to update the **Settings** window with the version even if the **Link with Selection** button () is disabled in the latter window.

Estimating disk space usage, clearing computed data, and permanently deleting is also supported when selecting multiple versions in the table.

THE VERSION DETAILS DIALOG

The **Version Details** dialog shows further details for an item version. The following is shown for all items:

- **Location** — the database, repository, and branch that the version was saved to.
- **Saved** — the time when the version was saved.
- **Saved by** — the name of the user that saved the version.
- **Title** — the title set on the item in the saved version.
- **Comments** — the optional comments provided when the version was saved.

For a model version:

- **Saved in** — the COMSOL Multiphysics versions that the model version was saved in.
- **Item save type** — if the model is a regular model or a draft model.
- **Item version type** — if it is a model, application, or physics.
- **Filename**. The filename used by the model version when exporting it to the file system.
- **Description**. The description of the model in the saved version.

For a file version:

- **Item version type** — if it is a file or fileset.
- **File size**. The size of the file version when stored on the file system. For a fileset, this is the total size of all files.
- **Description**. The description of the file in the saved version.

Click the **Edit Comments** button ( ) to update the comments for the commit that the version was saved in — see also Commit Settings. Click **Save** to save the updated comments to the database. Click **Cancel** to revert back to the original comments.

For a Model Manager server database, only an administrator or the user that saved the version can update the comments.

You can find an older version in your database based on its commit comment by selecting **All Versions in Database** ( ) in the **Model Manager** window, for example, and applying a Commit Comment filter.

EXPORT TO NEW LOCAL DATABASE

You can export a single model version to a new local database by right-clicking a version in the Versions window and selecting **Export to New Local Database** ( ). Select a name for the database directory in the **New Local Database** dialog and click **Save**. You can then open the created database in the COMSOL Desktop — see Opening a Local Database.

The References Window

You use the **References** window to view and browse versions that are *referenced* from a particular version or, conversely, versions that are *referencing* a particular version. This

enables you, for example, to discover model-file and model-part relationships without first having to open a model in the COMSOL Desktop.

The versions listed in the **References** window for a model version correspond to the subset of entries in the **Auxiliary Data** window whose **Location** column point to the database. These entries can either be data files used as input or output, or geometry parts used as input.

The references shown in the window follows the current version selection in The Model Manager Window, The Databases Window, The Versions Window, or The Maintenance Window (depending on which window has focus). Click **Link with Selection** ( ) to disable this automatic linking. You can still update the **References** window with a new version by right-clicking and selecting **References** ( ).

You can view either referenced versions or referencing versions by clicking **Show Referenced Versions** ( ) or **Show Referencing Versions** ( ), respectively. If you open the window for a selected model version that does *not* contain reusable geometry parts, referenced versions are shown by default. For a file version or a model version containing such parts, referencing versions are shown. The title of the selected version is shown at the top of the window. If a model version reference found in the table is opened in the COMSOL Desktop, the corresponding table row is highlighted in bold.

The referenced or referencing versions are shown in a table sorted in chronological order. A maximum of 100 most recent versions are initially shown. Click the **Show More** button ( ) to append the next 100 versions to the table. You can set another value for this default page size in the **Result Page Size** field on the **Model Manager** page in the **Preferences** window.

The table columns are:

- The type column — the type of the item represented by an icon.
- The **Title** column — the title set for the item in that version.
- The **Reference Type** column — the type of reference between a model version and a file version describing in what way they are related.
- The **Saved** column — the time in which the version was saved.
- The **Saved By** column — the name of the user that saved the version.
- The **Repository** column — the repository that the version belongs to.

- The **Branch** column — the target branch in which the version was saved.
- The **Comments** column — the optional comment provided when the version was saved.

Select a row in the table to see the corresponding settings for the referenced model or file in The Settings Window at the point in time the version was saved.

You can iteratively explore how different item versions depend on each other by repeatedly selecting a row in the table and clicking **References** ( ).

!	The database protects the *referential integrity* between item versions: a file version that is referenced by a model version, for example, cannot be permanently deleted, as long as both versions are saved in the same database.

THE REFERENCES WINDOW TOOLBAR

The toolbar in the **References** window contains the following toolbar buttons:

- Click the **Refresh** button ( ) to refresh the table in case any new version references have been created. The table will automatically refresh if you save a new version to the database from the COMSOL Desktop.
- Click the **Show More** button ( ) to append older versions to the table.
- Click the **Link with Selection** button ( ) to enable or disable whether to automatically update the **References** window based on the current selection in the **Model Manager** window, the **Databases** window, the **Versions** window, or the **Maintenance** window (which ever has focus).
- Click the **Version Details** button ( ) to open The Version Details Dialog containing more information on the version.
- Click the **Open** button ( ) to open a selected model version in the COMSOL Desktop.
- Click the **Run** button ( ) to launch and run a selected version in the COMSOL Desktop. Only enabled if the selected version is an application.
- Click the **Preview File** button ( ) to open a selected file version using the default application for its file type. See Previewing Files.
- Click the **Compare** button ( ) to compare a selected model version with the model opened in the COMSOL Desktop. Select two model versions to compare them with each other.

- Click the **References** button ( ) to show version references for the selected version.
- Click the **Show Drafts** button ( ) to enable or disable whether to include version references that are draft versions.
- Click the **Show Referenced Versions** button ( ) to show versions that are referenced by the viewed version (as indicated by the title next to the toolbar).
- Click the **Show Referencing Versions** button ( ) to show versions that are referencing the viewed version (as indicated by the title next to the toolbar).
- Select from the **Reference Types** menu ( ) the types of references to include in the table. The available options are **Input File**, **Output File**, and **Geometry Part**.

If you right-click a version in the table, you can also

- Select **Disk Space Usage** ( ) to see an estimate of the required disk space usage if the version is saved to the file system. See Estimating Disk Space Usage.
- Select **Export** ( ) to export the version to the file system — see Exporting Items.
- Select **Export to New Local Database** ( ) to export the version to a new local database — see Export to New Local Database.
- Select **Branch** ( ) to create a new branch from the commit that the version was saved in.
- Select **Snapshot** ( ) to record a snapshot of the commit that the version was saved in.
- Select **Copy Location** ( ) to copy a text string with a URI that uniquely identifies a model or file version in the database to the clipboard. See Copying Model and File Locations for what you can do with this text string.
- Select **Clear Computed Data** ( ) to clear all built, computed, and plotted data of a model version. Data shared with other model versions via deduplication will not be deleted from the database. Clearing cannot be undone. See Built, Computed, and Plotted Data.
- Select **Permanently Delete** ( ) to permanently delete a version in the database. Data shared with other versions via deduplication is not deleted. Permanently deleting cannot be undone. See Permanently Deleting Models and Data Files.

  You will only be able to perform this deletion for versions that are not referenced by other versions.

- Select **Restore Version** ( ) to save the selected version as a new latest version. The target item and target branch for the save is the same as that of the selected version. See Restore Version for further details.
- Select **Settings** ( ) to update the **Settings** window with the version even if the **Link with Selection** button ( ) is disabled in the latter window.

REFERENCE-TRACKING IN MODEL MANAGER

Model Manager keeps explicit track of references between versions when those versions are all stored in the same database. To see versions from other databases that are referenced by a model version, open the model version in the COMSOL Desktop and scan the **Auxiliary Data** window for input or output with a database set in the **Location** column.

References between a model version and other item versions are automatically discovered and tracked when a model is saved to a database. Model Manager looks for versions referenced in the model tree of the saved model and matches them with versions stored in the target database. This reference-tracking mechanism may lead to some surprising results that are good to be aware of:

- If you export output from the model opened in the COMSOL Desktop directly to a database, the resulting saved file version will not be referenced in the database *until* you also save a new version of the model in that database. See also Loading and Saving Auxiliary Data Files Stored in Databases.
- If you update a file from the Model Manager workspace — for example via the **Settings** window — the saved file version will not be referenced by any model version, even if the previous file version was referenced. To see the references for the older file version, open The Versions Window for the file item, select the older version, and click the **References** button ( ).
- If you open the **References** window for a file version, you are likely to discover that it is referenced by a whole sequence of model versions. For a file version with reference type **Output file**, the oldest model version in that sequence is likely the best candidate for reproducing the output.

Opening Models

You can open a model version in the COMSOL Desktop from The Model Manager Window, The Databases Tree, or The Maintenance Window by double-clicking a model version or by selecting a model version and clicking **Open** ( ) in the **Item**

section of the **Home** toolbar. You can also double-click or click **Open** from the toolbars of other windows in the Model Manager workspace that show model versions.

When you try to open a model version from the **Open** window or the **Model Manager** window and the searched location is set to a branch, or when you try to open from the **Databases tree** under a branch node, Model Manager first checks if any newer versions of drafts of the model exist on the same branch. If so, The Open Dialog is shown in which you can choose to open one of these newer draft versions instead. This helps you, for example, from accidentally creating a new draft when one is already ongoing — perhaps created and made accessible to you by one of your coworkers. If there is instead a newer version of the model itself — saved *after* any potential draft versions of the model — that version is automatically opened by Model Manager.

A similar safeguard exists for the **Versions** window: When you try to open the latest version visible in the window, Model Manager checks if a later version has been saved to the same branch and, if so, gives you the choice to open that one instead. As in the previous case, this version may have been saved by a coworker while you were working elsewhere in the COMSOL Desktop. This check, however, does not include other drafts of the model.

THE OPEN DIALOG

The **Open** dialog shows model versions that Model Manager assumes may be of particular interest when you try to open a model version that you expect to be the *latest* one. The contents of the dialog depend on the window: For a window that shows the latest versions of multiple models with respect to a branch, the versions are presented as a tree with the root node being the latest version of the model initially selected. Child nodes are the latest versions of drafts created from that model. For a window that shows the version history of a specific model, two versions are instead shown in a table — the selected version thought to be the latest one (bottom row) and the actual latest version found when checking the branch in the database (top row). Select one of the tree nodes or table rows and click **Open** to open a model version in the COMSOL Desktop.

PREVENTING SIMULTANEOUS ACCESS

While Model Manager has no concept of automatically *locking* a model for exclusive access by a single user, you can achieve the same effect via permissions — see Granting

Permissions. You can, for example, assign the **Open model** permission or **Save model** permission exclusively to yourself so that only you can open or save versions of a model.

If a simulation model is being developed in a collaborative setting, and you want to guarantee that only one person is modifying the model at a time, assign a **Protected** permission template and hand over the model by changing its owner — see Owners.

OPENING A MODEL ON A CLUSTER

You can open a model stored in a database on a cluster as long as the cluster's root node can connect to the database. In the **Preferences** window, select **From intermediate file** in the **Load model on nonroot nodes** list on the **Cluster** page under the **Model Manager** page to only require that the root node can connect. Select **Direct from database** in case also nonroot nodes can connect.

When **From intermediate file** is selected, the model is first saved as a temporary file on the root node's file system and then transferred to all nonroot nodes by copying the file. When **Direct from database** is selected, all nodes load the model directly from the database without any intermediate step.

NETWORK CONNECTION ISSUES

When you open a model from a database, binary data such as meshes, geometries, solutions, and result plots are only loaded on-demand. This reduces, for example, unnecessary bandwidth usage when connected to a server database via a Model Manager server.

If you experience network connection issues when COMSOL Multiphysics needs to load such binary data from the database, you will be asked to save any ongoing modeling work as a recovery on the file system. The model can be opened from recovery once the network connection is reestablished.

Keeping and Opening Recovery Files in the *COMSOL Multiphysics Reference Manual*

Running Applications

Click **Run** (▶) in the **Item** section of the **Home** toolbar to launch and run an application version (A) from The Model Manager Window, The Databases Tree, or The Maintenance Window. You can also click **Run** from the toolbars of other windows in the Model Manager workspace that show application versions.

Much like when Opening Models, Model Manager first checks for the presence of other versions that may be of interest when you try to run an application version that you expect to be the *latest* one. If any are found, the **Run** dialog is shown in which you can choose to run one of these versions instead. Select one of the tree nodes or table rows and click **Run** to launch and run an application version in the COMSOL Desktop.

Inserting Contents from Models

You can insert components, geometry parts, parameters, and other model contents from versions in your database into the model currently opened in the COMSOL Desktop. This is a useful way of reusing modeling setups from existing models when you, for example, start with a new model.

From the **Settings** window of a model version, select the node you want to insert into the opened model from the model tree in the **Contents** section. Click **Insert into Model** ( ) in the toolbar below the tree to insert the selected node.

Only a subset of all node types support insertion. The **Insert into Model** button ( ) is disabled for unsupported selections.

The **Insert into Model** button ( ) is a useful alternative to Inserting Parts and Other Model Contents from Databases via the **Select Model** window when you want to reuse multiple model tree nodes from the same source model. Going via the **Contents** section also has the advantage that you can right away select which node you want to insert from a source model in case the model has multiple such nodes of the same type.

Many nodes in the **Contents** section also support being copied to clipboard. This enables you, for example, to copy nodes from a model version in one COMSOL Multiphysics program session into the model opened in another program session: Select one or more nodes and click **Copy** ( ) or press Ctrl+C. Right-click a node in

the Model Builder, Application Builder, or Physics Builder tree and select **Paste** ( ) or press Ctrl+V.

COMPONENTS, GEOMETRIES, MATERIALS, AND PHYSICS

Select a component, geometry sequence, material, or physics node and click **Insert into Model** ( ) to insert the selected node. If there is more than one target available in the opened model, the **Insert into Model** dialog is shown giving you a list of possible targets for the insertion. Select a target parent node and click **OK**. The node is copied into the opened model and then selected in the **Model Builder** window.

GEOMETRY PARTS

Select a geometry part and click **Insert into Model** ( ) to insert the selected part under **Global Definitions** > **Geometry Parts** in the model opened in the COMSOL Desktop. Unlike the plain copying done when inserting Components, Geometries, Materials, and Physics, the geometry part is loaded as auxiliary data into the model tree with a reference back to its source model version.

- Geometry Parts Saved in Databases
- The Auxiliary Data Window for Database Input and Output
- Reference-tracking in Model Manager

If the opened model contains components or geometry parts with the same space dimension as the selected geometry part, you have the option to also create a part instance of the inserted part. From the **Insert into Model** dialog:

1. In the **Insert As** list, select **Geometry part**.
2. Select the **Create part instance in geometry** checkbox.
3. Select a geometry under one of the available geometry parts and components in the tree.
4. Click **OK**.

The geometry part is loaded into the model and a new part instance is created.

You can also insert the geometry sequence of the part as a plain copy into the opened model. Select **Copy of geometry sequence** in the **Insert As** list, select a target geometry, and click **OK**. In this case, no reference to the source model version is included.

The **Copy of geometry sequence** option may be used, for example, to create a new part by starting from the sequence of an existing part. Create a new part under **Global Definitions** in the **Model Builder** window with the same space dimension as the existing part in the database. Select the existing part in the **Contents** section, click **Insert into Model** ( ), and select the new part as the target for the copy.

PARAMETERS

You can insert parameters from a model version in the database into the model opened in the COMSOL Desktop. Perhaps you have a list of parameters whose names and values you find yourself reusing over and over again in your modeling work. You can then save these parameters with a model in your database to be used as a template source for parameters when creating new models.

Select a **Parameters** node and click **Insert into Model** ( ) to insert its list of parameters into the opened model. You can also click **Details** ( ) to get a preview of the parameters in the **Settings** table of the **Details** dialog — see also Model Settings.

Only global parameters found under **Global Definitions** support insertion. Inserting local parameters is not supported.

For a model in the COMSOL Desktop with a single **Parameters** node having an empty parameter list, the inserted parameters are added to the existing node's list. The label of the **Parameters** node is also updated to that of the selected node's label. For a model in the COMSOL Desktop that already contains parameters, the inserted parameters are instead added as a new **Parameters** node. Any **Cases** nodes present under the select **Parameters** node are inserted as well.

An initial **Parameters** node with an empty parameter list is automatically added when you create a new blank model in the Model Builder workspace. You may want to insert your list of parameters from the existing model version *before* writing any new parameters to avoid ending up with multiple **Parameters** nodes in the new model.

Parameter Name Collisions

An **Insert into Model** dialog is shown asking you what to do if there are any name collisions between the parameters being inserted and the ones already present in the opened model. Select **Overwrite** to replace the current values of such duplicate parameters with those being inserted from. Select **Rename Duplicates** to instead rename inserted parameters found to be duplicates by appending a numerical suffix to their names. Other parameters, having no name collisions, are inserted as previously described.

Previewing Files

You can preview a data file by opening it with the default application for its file type. Double-click a file version, or select a file version in The Model Manager Window, The Databases Tree, or The Maintenance Window and click **Preview File** ( ) in the **Item** section of the **Home** toolbar. You can also click **Preview File** from the toolbars of other windows that show file versions in the Model Manager workspace. If the file version is a fileset, you will be asked to select which file resource to preview from the **Select File from Fileset** dialog. Select a row in the table and click **OK**.

Previewing a file can also be done from the **Contents** section in the **Settings** window for a file version — see File Settings. Select the file resource to preview in the table and click **Preview File** ( ) in the toolbar below.

Comparing Models

You can compare the model opened in the COMSOL Desktop with a model version from The Model Manager Window, The Databases Tree, or The Maintenance Window by selecting the version and clicking **Compare** ( ) in the **Item** section of the **Home** toolbar. You can compare two model versions with each other by selecting two versions and clicking **Compare** ( ). You can also click **Compare** from the toolbars of other windows that show model versions in the Model Manager workspace.

	Comparing Models Saved in Databases

Copying Model and File Locations

A model version or a file version stored in a Model Manager database can be uniquely identified using an *item version location*. This is a text string URI that you can copy to your computer's clipboard by right-clicking an item version in the Model Manager workspace and selecting **Copy Location** ( ).

An item version location should not be confused with a commit location. The former identifies an item version stored in a Model Manager database; the latter identifies a commit in a Model Manager database — see also Commits and Locations.

USING COPY LOCATION WITH MODELS

You can use **Copy Location** ( ) to open a specific model version in another COMSOL Multiphysics program session:

1. Right-click a model version in the Model Manager workspace and select **Copy Location** ( ).

   A text string URI that uniquely identifies the model version is copied to your computer's clipboard.
2. From the **File** menu in another COMSOL Multiphysics program session, select **Open From** ( ).
3. Select the **Clipboard** menu option in the **Open** window. The model version is shown as a single option in the list on the right. Click **Open** ( ).

An identical **Clipboard** menu option also appears in the **Select Model** window when you have copied a model version location — see Inserting Parts and Other Model Contents from Databases.

You can also use **Copy Location** ( ) to change the source of a loaded geometry part found in the model opened in the COMSOL Desktop, as long as the old part and the new part use the same node tag:

1. Right-click a model version containing the geometry part in the Model Manager workspace and select **Copy Location** ( ).
2. Open the Model Builder workspace and select the Loaded Part in the **Model Builder** window.
3. In the **File** section of the **Settings** window, click the **Location** menu ( ) and select **Paste Location** ( ).

4 Click **Reload** () to load the geometry part into the model.

Using the **Copy Location** action from The Versions Window is a good alternative to finding the version via the **Select Model** window when you want to reference an older version of a geometry part.

Inserting Parts and Other Model Contents from Databases

The location text string URI can also be used to access the model version via method code using the Model Manager API or to specify input and output when Running COMSOL Batch with Models in Databases. While the URI itself has a rather opaque format, you can recognize the URI by it always starting with `dbmodel:///`.

USING COPY LOCATION WITH FILES

For a file version, you can use **Copy Location** () to reference the file version from an input or output setting in the model opened in the COMSOL Desktop:

1 Right-click a file version in the Model Manager workspace and select **Copy Location** (). If the file version is a fileset, you will be asked to select which file resource whose file location you want to copy via the **Select File from Fileset** dialog. Select a row in the table and click **OK**.

   A text string URI that uniquely identifies the file resource is copied to your computer's clipboard.

2 Open the Model Builder or Application Builder workspace and select a model tree node that has a **Filename** field with a **Location** menu() in the **Settings** window.

3 Paste the copied location into the **Filename** field.

   The filename of the file resource, and the database the file belongs to, is displayed as a label in the **Filename** field. Depending on the input or output nature of the setting, you can now import or export data from or to the file in the database.

Using the **Copy Location** action from The Versions Window is a good alternative to finding the version via the **Select File** window when you want to reference an older version of a data file.

- Selecting Files in Databases as Input Sources
- Selecting Files in Databases as Output Targets
- Loading and Saving Auxiliary Data Files Stored in Databases

The location text string URI can also be used to access the file version via method code using the Model Manager API. While the URI itself has a rather opaque format, you can recognize the URI by it always starting with `dbfile:///`.

Organization of Models and Files

Model Manager supports two means of organizing models and files in a database — assigning Tags to them and placing them in Repositories. The first primarily helps you with search and workflow management, the second with access control.

In this section:

- Assigning Tags to Items
- Organizing Items in Repositories

Assigning Tags to Items

You can assign Tags to your models and data files to help you with organizing them in the database. You can even assign tags to the tags themselves, thereby building a hierarchical tree structure of tags — see The Tag Tree. In this regard, tags may remind you of folders on the file system. But tags go beyond that:

- You can find models and data files in the database by searching on their assigned tags — either by matching textually on tag titles via Full Text Search or by applying a Tag filter on the unique keys of tags. You can even find models and data files by searching on *ancestor* tags — that is, tags assigned to the items' tags themselves. For example, if you create an `Interpolation Functions` tag and assign it a My `Project` parent tag, you may assign the former tag to data files and then find those same data files by filtering on the latter tag.
- You can assign multiple tags to a model, data file, or even to a tag itself.
- Tag assignments are version controlled. A new commit is saved on the branch whenever you add or remove assigned tags for an item — see Adding and Removing Tag Assignments. This has the benefit that:
  - You can track changes to the assigned tags of items over time, giving you an organizational history. Perhaps you introduce tags for different stages in your modeling workflow, say `In Progress`, `To Review`, `Approved`, and `Needs More Work`.
  - You can revert any updated tag assignments from The Revert Window.
  - You can search items based on their tag assignments as they were at a particular time. Perhaps you want to find all items that were assigned a tag, say `In`

`Progress`, a month ago, or merely see what the tag tree looked like — see The Tag Tree for details.

- The settings for tags are version controlled. If you change the title of a tag and click **Save** ( ) in the **Settings** window, a new version of the tag is created. Any tag assignments involving the saved tag will, however, be left unchanged.

CREATING NEW TAGS

Click **New Tag** ( ) in the **Database** section of the **Home** toolbar to open the **New Tag** dialog. The **Location** field shows the database, repository, and branch that the new tag will be created in.

1. In the **General** tab, write the title for the new tag in the **Title** field.
2. Select the **Add to selected models and files** checkbox if you want the new tag to be automatically assigned to the current selection of model and files in The Model Manager Window or The Databases Window (depending on which window has focus). The checkbox is disabled if no models or files are selected.
3. In the **Parent tags** tab, select tags in The Tag Tree that will be assigned to the new tag — in other words, its parent tags. Leaving all nodes in the tree cleared will create the new tag under the root. You can filter the tree of available parent tags by writing a tag title in the text field above the tree.

   The tree is empty when you create your first tag.
4. In the **Comments** field, write an optional comment for the associated commit.
5. Click **OK** to save the first version of the new tag in the database.

If a tag version is selected in The Model Manager Window or The Databases Window when you click **New Tag** ( ), that tag will be initially selected as a parent tag in The Tag Tree in the **New Tag** dialog.

You can see your new tag under the branch node in The Databases Tree if you have selected **Items** when clicking **Show** ( ) in the **Databases** window's toolbar. You can also see it in the Tree View in The Model Manager Window as long as you have not written any search expressions in the search field or applied any filters.

THE TAG TREE

Since tags can be assigned to other tags, items in a Model Manager database can be represented as tree nodes in an hierarchical tree structure. Items assigned a particular tag appear as child nodes to the tag node in tree. You can find a particular item in

multiple positions in the tree as items can be assigned more than one tag. You can think of items not assigned any tags at all as placed under a hidden *root tag*.

To help you with identifying the regular model that a draft originated from, the draft node is placed as a child node to the regular model node in the tag tree, whenever both the model and draft share the same assigned tag. See also Item Save Types.

You can browse the tag tree in The Databases Window if you have selected **Items** in the **Show** menu ( ) or in the The Model Manager Window if you have selected the **Tree** button ( ). Expanding a branch tree node in the former window or searching a branch in the latter window shows the tag assignments at present time — see also Searching in Branches. To browse the tag assignments at some point in the past:

1. Right-click one of your branches under the **Branches** node ( ) in the **Databases** window and select **Commits** ( ).
2. In the opened **Commits** window, right-click one of the commits in the table and select **Search in Commit** ( ).
3. The Tree View in the **Model Manager** window is updated to show the tag assignments as they were at the time the commit was saved.

If you have created a snapshot to record a specific commit, you can expand its tree node in the **Databases** window to see the tag tree at the time the recorded commit was saved — see also Searching in Snapshots and Commits.

ASSIGNING TAGS

To assign already created tags to an item, do one of the following:

- Select the item in The Model Manager Window or The Databases Window and click the **Set Tags** button ( ) in the **Item** section of the **Home** toolbar. You can select multiple models and files if you want to set the tag assignments of more than one item at once — see also Adding and Removing Tag Assignments.
- Set assigned tags from the **Tags** section in the **Settings** window of Items. Do not forget to click **Save** ( ) in the **Settings** window to save any changed tag assignments.

- Set assigned tags from the **Tags** section in the **Save** window when saving a model version or in the **Export** window when exporting a file version — see also Saving Models to Databases and Exporting Output Directly to a Database.
- Select to import folders as tags when bulk importing files from the file system — see also Importing Files.

All four options sets the tag assignments via a commit. The second, third, and fourth option also save a new version of the item. The first option does not, which may be preferable as to not create unnecessary Save Conflicts if, for example, a model item happens to be simultaneously open in the COMSOL Desktop.

GUIDELINES ON TAG TREE SETUP

Model Manager will neither prevent you from creating tags having multiple parent tags, nor will it prevent you from creating tags having the same title. While this gives you complete freedom in setting up your tag tree, there are some pros and cons that should be weighed.

Assigning more than one parent tag to a tag enables you to have multiple organizational subtrees with overlapping contents. You can, for example, have a main subtree used to uniquely categorize all your models and data files based on characteristics — much like files in folders on the file system. The tag layout in that subtree, once created, would seldom change. You can then have various other subtrees used to organize projects in which you want to work with whole collections of models and data files from the main subtree — rather than assigning the project tag to all those items directly, you assign it to some of the items' assigned tags. This requires some discipline, though, to not end up with a tag tree that is hard to navigate.

By creating multiple tags with the same title, you can organizationally separate different collections of models and data files in the tag tree while still labeling these items using a common concept. This enables you, and your coworkers, to easily find the items when searching on the tag title, without making it difficult to browse them when navigating the tag tree. Having the same title for different tags will make it harder, though, to quickly identify where in the tag tree a specific item is located when viewing it in a result table or list. Moreover, you must now remember to select all of them when applying a corresponding filter for the common concept represented by the tags.

Organizing Items in Repositories

You can create multiple Repositories if you want to group your models and data files into separate containers in the database. You may, for example, want to set up permissions that differ between various collections of models — perhaps hiding sensitive models from all but a select group of users.

There is nothing preventing you from saving versions of the same model or data file in more than one repository — for example by changing to a branch in another repository when saving from the **Save** or **Export** window. Doing so, however, will make it difficult to track how an item has evolved over time via its different versions, as well as make it harder to reason about the Access Control of the item. You may be better off saving a new item from the existing item using **Save as New** ( ) in the **Save** or **Export** window — in this way, you keep items within their respective repositories, while still being able to track how a new item in one repository evolved from an existing item in another. See also Splitting a Model Version History in Two.

ADDING REPOSITORIES

Click **Repository** ( ) in the **Repository** section of the **Home** toolbar to open the **Add Repository** dialog. The **Database** field shows the label of the database in which the repository will be created.

1. Write the name of the new repository in the **Name** field.
2. Write the name of the initial, default, branch in the **Default branch name** field — see Default Branch.
3. You can provide an optional comment for the initial commit that will be saved when the branch is created in the **Comments** field.
4. You can set up permissions for the new repository in the **Permissions** field. This field is only shown if connected to a server database via a Model Manager server. See Granting Permissions.
5. Click **OK** to add the repository.

The added repository appears as a new child node to the corresponding database node in The Databases Tree.

Basic Version Control

You modify items — that is, models, files, and tags — in the database by saving a *commit* on a branch. The set of changes in such a commit can be categorized as:

- Added, updated, and restored items — these are changes that save new versions of items.
- Added and removed tags — these are changes that modify the assigned tags of an item.
- Deleted items — these are changes that effectively hide items on a branch.

Commits involving the first category of changes have a direct correspondence with the table entries in The Versions Window — each version is saved in a commit. The latter two categories, which do not save new versions, only appear in The Commits Window.

You can provide an optional *commit comment* when saving a new commit. Oftentimes a comment describing the changes is already suggested. The comments can later be read and modified from The Commits Window or from The Versions Window. You can also find versions saved in a particular commit by applying a filter on the comment in The Model Manager Window.

In this section:

- Saving Versions
- Adding and Removing Tag Assignments
- Deleting Items
- Recording Snapshots

Advanced Version Control

Saving Versions

There are various ways in which you can save a new version of an item, both within the Model Manager workspace and from one of the other workspaces. You can, for example, save a new version of the model opened in the COMSOL Desktop from the **File** menu by clicking **Save Draft** ( ), **Save as Version** ( ), or **Save To** ( ) — see Saving Models to Databases.

In the Model Manager workspace, you can save a new version by clicking **Save** ( ) in the **Settings** window. For a model this will, for example, update the model's title, description, and filename, but leave the model tree, as well as any binary data like meshes, geometries, and solutions, unchanged. Other examples of saving versions include:

- Exporting output from a model directly to the database — see Exporting Output Directly to a Database.
- Importing files on the file system into the database — see Importing Files.
- Creating a new tag — see Creating New Tags
- Renaming an item — see Rename Item
- Restoring an older item version as a new latest version — see Restore Version.
- Merging updated items from a source branch to a target branch — see Merging.
- Reverting a commit that included added, updated, or deleted items — see Reverting.

You generally want to avoid saving a new version of the model opened in the COMSOL Desktop from the Model Manager workspace. Otherwise you will get Save Conflicts when trying to save a new version of the desktop model.

Creating a New Branch does not save new versions of the items included on the new branch — each item version on the new branch is initially the same as that on the parent branch.

RENAME ITEM

From The Model Manager Window or The Databases Tree, right-click and select **Rename** ( ), or press the keyboard shortcut F2, to change the title of an item via the **Rename Item** dialog. The **Location** field shows the database, repository, and branch that the item will be renamed on

1. Write the new title for the item in the **Title** field.
2. Write an optional comment in the **Comments** field.
3. Click **OK** to save the renamed item as a new version in the database.

RESTORE VERSION

You can restore an older version in the version history of an item as the new latest version. This enables you to, for example, recover an older model version without first having to open it in the COMSOL Desktop and manually save it as a new version via **Save as Version** ( ).

From The Versions Window, click **Restore Version** ( ) to open the **Restore Item Version** dialog. The **Location** field shows the database, repository, and branch that the item version will be restored to. The title of the restored version is shown in the **Title** field.

1. Write an optional comment in the **Comments** field.
2. Click **OK** to perform the restore.

The target item when restoring a version in The Versions Window is given by the item whose version history is currently shown, which may not be the same item that the selected version belongs to. For example, restoring from a selected version of a regular model via the version history of a draft, with the draft created from that regular model, saves a new latest version of the draft.

Reverting Changes on a Branch

Restoring Versus Reverting

The use case for restoring a version versus that of reverting a commit in which a version was saved are best exemplified via two examples:

- You have created a draft from a model and then saved ten versions of that draft. By the tenth version, you realize that the modeling setup has gone wrong for the draft, and that things started to go bad by version four. Solved by restoring version three from The Versions Window such that a version eleven, identical to version three, is saved directly in the database.
- You have decided to save what you, initially, think is a finished draft back to its original model. Such an action simultaneously saves a new version of the original model and deletes the draft. You then realize that you did this too soon — there were more things you wanted to first change in the draft. Solved by reverting the commit in which the draft was saved back to the original model from The Commits

Window: the draft is added back to the branch and the model is updated to its previous version before it was overwritten by the draft.

Adding and Removing Tag Assignments

You can modify the assigned tags of an item without saving a new version. This is useful if you, for example, want to change the tags of the model that is opened in the COMSOL Desktop without introducing Save Conflicts. You can also modify the tag assignments of multiple items at once — for example by assigning all of them with the same tag.

From The Model Manager Window or The Databases Window, select one or more items and click the **Set Tags** button ( ) in the **Item** section of the **Home** toolbar to open the **Set Tags** dialog. The **Location** field shows the database, repository, and branch in which the assigned tags will be modified. The **Item** field indicates the item, or items, whose tag assignments will be modified.

1. Under **Tags**, select and clear checkboxes for tags that will be added and removed, respectively. Leaving all nodes in the tree cleared will remove all tags from the item and place the item under the root tag — see also The Tag Tree. You can filter the tree of available tags by writing a tag title in the text field above the tree.

   If multiple items were selected when you opened the **Set Tags** dialog, some checkboxes may be shown in an *indeterminate state*. These correspond to tags that some, but not all, of the items were assigned prior to opening the dialog. Leaving the checkboxes in their indeterminate state means that the corresponding tag assignments will *not* be modified — an item that already had the tag assigned will keep that tag, an item that did not have the tag will not gain it.
2. Write an optional save comment in the **Comments** field.
3. Click **OK** to save the new tag assignments to the database.

You can create a new tag from the **Set Tags** dialog. Click **New Tag** ( ) to open the **New Tag** dialog — see Creating New Tags. Once created, the new tag is added to the tag tree in the **Set Tags** dialog in an initially selected state.

Deleting Items

You can delete an item in the database that you no longer have any need for. The item and all its versions will still remain in the database after deletion, but will be excluded when browsing or searching the latest versions of items. You can still find the item and

its versions by, for example, Searching in Snapshots and Commits or by Searching All Versions in the Database.

DELETING MODELS AND FILES

Select one or more models and files and click **Delete** ( ) in the **Item** section of the **Home** toolbar to open the **Delete Item** dialog. You can also press the Del key. The **Location** field shows the database, repository, and branch that the items will be deleted on. All items that will be deleted are shown in a table under **Items**.

1 Write an optional comment in the **Comments** field.

2 Click **OK** to delete the items in the database.

As with all commits, the deletion targets a specific branch. If you have created other branches that include the selected items, the items will remain visible on those branches.

To find a model or file previously deleted on a branch, select **Latest Versions for Location** ( ) in The Model Manager Window and the set the searched location to a snapshot created before the item was deleted. You can also browse to a commit saved *before* the item was deleted in The Commits Window for the branch, right-click the commit, and select **Search in Commit** ( ). In both cases you can find the item by searching in the **Model Manager** window.

DELETING TAGS

Select a tag and click **Delete** ( ) in the **Item** section of the **Home** toolbar to open the **Delete Tag** dialog. The **Location** field shows the database, repository, and branch that the tag will be deleted on. The **Title** field shows the title of the latest version of the tag.

1 In the **Tagged Items** field:

- Select **Remove tag assignment** to unassign the tag from any items that the tag is currently assigned to. Items that were only assigned the deleted tag will be found under the root tag in The Tag Tree after the deletion.
- Select **Assign nearest ancestor tag** to unassign the tag from any items that the tag is currently assigned to and instead assign the tag's parent tags to these items. This option is only available if the tag to be deleted has parent tags other than the root tag.

2 Write an optional comment for the associated commit in the **Comments** field.

3 Click **OK** to delete the tag. Any items that were assigned the deleted tag will have their tag assignments updated as specified by the **Tagged Items** value.

Recording Snapshots

You can save a reference to a particular commit in a repository by *recording* a snapshot. You may, for example, find snapshots useful when:

- You have reached a stage in your modeling workflow when the versions of a collection of models and files on a branch have reached a finished state. Via the snapshot, you can easily find all these versions in the future.
- You repeatedly want to browse in the database based on how it looked at the time of the commit. See Searching in Snapshots and Commits.

Recorded snapshots are shown under the **Snapshots** node in The Databases Tree. Click **Show** ( ) in the **Databases** windows toolbar and select **Items** to see the correspondingly recorded item versions.

Click **Snapshot** ( ) in the **Repository** section of the **Home** toolbar to open The Record Snapshot Dialog. When The Model Manager Window has focus and the searched location is a branch, the latest versions will be recorded. The same is true when The Databases Window has focus and a node in a branch subtree is selected.

	Snapshots

THE RECORD SNAPSHOT DIALOG

The **Location** field shows the database, repository, and branch where the snapshot is recorded. The **Date** field shows the time of the corresponding commit that will be referenced by the snapshot.

1 Write a name for the new snapshot in the **Name** field.

2 You can set up permissions for the new snapshot in the **Permissions** field. This field is only shown if connected to a server database via a Model Manager server. See Granting Permissions.

3 Click **OK** to record the new snapshot.

The recorded snapshot appears as a new child node to the **Snapshots** node in The Databases Tree.

Bulk Operations

Model Manager supports bulk import and bulk export of models and data files to and from the file system.

In this section:

- Importing Files
- Exporting Items

Importing Files

You can import models and data files from the file system directly into a database without, for example, having to open each model in the COMSOL Desktop and manually save it. You can also choose to preserve any folder organization you made on the file system by importing folders as Tags. Files inside a folder will then be assigned the corresponding tag.

Imported models will be converted to the format used by the current COMSOL Multiphysics version.

Click **Import** ( ) in the **Database** section of the **Home** toolbar to open the **Import** dialog. A **Target Location** for the imported files will be automatically suggested based on the current selection in The Model Manager Window, The Databases Window, or The Maintenance Window (depending on which window has focus). You can change the target by clicking the link button and choosing another branch in The Select Location Dialog.

- Click **Add Folder** ( ) to add a folder containing files to import. Files in subfolders will also be included.
- Click **Add File** ( ) to add a file to import.
- Click **Exclude File** ( ) to exclude selected files from being imported.

The files to import are shown in a table with columns:

- The type column — the type of file to import based on its file extension represented by an icon. The types are ( ) for `mph`, ( ) for `mphphb`, and ( ) for any other file extension.

- The **File** column — the filename of the file to import.
- The **Source** column — the directory path to the folder that was selected if added via **Add Folder** or the file path if added via **Add File**.

Click **OK** to begin the import. Clicking **Stop** in the progress window stops the import, but already imported files are not removed from the database. You can see a summary of the import in the **Messages** window in the Model Builder workspace once the import finishes.

If you have a license for the CAD Import Module available when importing a CAD assembly file, an attempt will be made to automatically resolve and include external component files that the assembly references.

The import functionality in the Model Manager workspace only lets you import the files as separate items in the database, with one version for each item. If you have, for example, a collection of MPH files that you want to import as versions of the *same* model, you can accomplish this instead by writing custom method code using the Model Manager API.

IMPORT OPTIONS

The import procedure can be configured under **Options** in the **Import** dialog.

Select the **Include auxiliary files found in imported models** checkbox to automatically import files on the file system that are referenced as input by an MPH file. This includes, for example, interpolation functions, geometry parts, CAD assemblies, and many other types of auxiliary data used by a model. The model will be automatically updated so that it references the corresponding auxiliary data saved in the database. Files referenced as output by a model are not imported.

In the **Tags** list:

- Select **None** to not assign any tags to imported items.
- Select **Import subfolders as tags** to import all subfolders found under a directory path in the **Source** column as Tags. Imported models and files found in these subfolders will be assigned the corresponding tags.
- Select **Use existing tags** to select tags to assign to imported models and files among tags already present in the database.

In the **Computed data** list:

- Select **Include or exclude based on application settings** if built, computed, and plotted data of an MPH file should be included or excluded in the imported model based on the corresponding save settings on the model's root node and the save settings in the **Preferences** window. See also The Root Settings Window in the *COMSOL Multiphysics Reference Manual*.
- Select **Include** to include built, computed, and plotted data of an MPH file in the imported model, regardless of any save settings.
- Select **Exclude** to exclude built, computed, and plotted data of an MPH file in the imported model, regardless of any save settings.

Built, Computed, and Plotted Data

Importing a large number of files with a complicated folder structure and auxiliary file dependencies may require a bit of planning to get the desired result. You are encouraged to first create a throwaway local database in which you can experiment with your import — perhaps clicking **Stop** when only a subset of your files have been imported.

Model Manager will detect and skip *duplicate* files, that is files with the same filename and file content. This includes both files that you have explicitly added to the table in the **Import** dialog, as well as referenced auxiliary files if you have selected **Include auxiliary files found in imported models**. If duplicate files are located in different subfolders, and you have selected **Import subfolders as tags**, the imported files will still be assigned tags corresponding to all those subfolders.

IMPORT TO TARGET TAG

You can import models and data files into the database so that all items are placed under a specific tag in The Tag Tree.

Right-click a tag and select **Import** ( ). The **Import** dialog opens with the title of the selected tag shown in a **Target tag** field. You add folders and files to import in exactly the same way as before — the only difference is that **Use existing tags** is not available under **Tags**.

Exporting Items

You can export models and data files from the database to the file system. Any tag organization you have made can be preserved on the file system by exporting tags as subfolders. Models and data files assigned a specific tag will be exported to the corresponding subfolder.

Click **Export** () in the **Database** section of the **Home** toolbar to open the **Export** dialog. The **Source Location** field shows the location from which versions of items will be exported. Click the link button or **Browse** button in the **Target folder** field to select a folder on the file system to which the items will be exported.

The models and files that will be exported are shown in a table with a type column and an **Item** column with the title of the corresponding item version. The models and files in the table are based on the current selection in The Model Manager Window, The Databases Window, The Commits Window, The Versions Window, The References Window, or The Maintenance Window (depending on which window has focus):

- Selecting a location — that is, a branch, snapshot, or commit — gives the latest versions of all items with respect to the corresponding commit of the location — see also Locations.
- Selecting tags gives the latest versions of all items that are assigned those tags or are assigned a tag with a selected tag as an ancestor in The Tag Tree.
- Selecting models or files gives their latest version with respect to the corresponding commit of the browsed location.

Select table rows and click **Exclude** () to exclude models and files in the table from being exported. Click **OK** to begin the export. You can see a summary of the export in the **Messages** window in the Model Builder workspace once the export finishes.

The file resources of a fileset will be exported to a directory whose directory name is the title of the fileset.

The export functionality in the Model Manager workspace only lets you export the latest version of each item selected for the export. If you want to export, for example, multiple versions of the *same* model as separate MPH files, you can accomplish this instead by writing custom method code using the Model Manager API.

EXPORT OPTIONS

The export procedure can be configured under **Options** in the **Export** dialog.

Select the **Include auxiliary files found in exported models** checkbox to automatically export model and file versions located in the database that are referenced by an exported model version. The model will be automatically updated so that it references the corresponding files on the file system. Item versions referenced as output by a model are not exported.

Select the **Export tags as subfolders** checkbox to export tags assigned to exported items as subfolders on the file system, with exported files placed inside these subfolders.

If you select **Export tags as subfolders**, and an exported item version has multiple tags, only one of these tags will be exported as a subfolder (the first one when sorting tag titles alphabetically). This includes items that you have explicitly added to the table in the **Export** dialog and any referenced auxiliary files if you have selected **Include auxiliary files found in exported models**.

In the **Computed data** list:

- Select **Include or exclude based on application settings** if built, computed, and plotted data of a model version should be included or excluded in the exported MPH file based on the corresponding save settings on the model's root node and the save settings in the **Preferences** window. See also The Root Settings Window in the *COMSOL Multiphysics Reference Manual.*
- Select **Include** to include built, computed, and plotted data of a model version in the exported MPH file, regardless of any save settings.
- Select **Exclude** to exclude built, computed, and plotted data of a model version in the exported MPH file, regardless of any save settings.

Built, Computed, and Plotted Data

User Management

Model Manager comes with tools for managing users, as well as groups of users, in a server database accessed via a Model Manager server.

In this section:

All user management functionality is either hidden or disabled for a local database in the Model Manager workspace.

Managing Users

You can manage Users in a Model Manager server database from the Model Manager workspace. You can, for example, see all users that have been active in the database or add users as members to Groups. The latter is useful, for example, if the Model Manager server has been set up with an authentication scheme that does not support providing such group memberships from an external credentials storage when a user logs in.

When you connect and authenticate with a Model Manager server, a new user will be automatically created in the database.

See also *Users* in the *Model Manager Server Manual*.

ADDING USERS

There may be situations where you want to add a new user to the server database before the user has connected for the first time. One example is when you preemptively want to grant them permissions to various database objects.

Click **User** () in the **Users** section in the **Database** toolbar to open the **Add User** dialog. The **Database** field shows the label of the database in which the user will be added.

1 Write the username of the user in the **Name** field. This username should match the one used when authenticating with the Model Manager server.

2 Write an alternative display name for the new user in the **Display Name** field. This is the name that will primarily be shown in the user interface.

3 All groups that the new user will be set as a member of is shown as a list under **Group memberships**.

   a Click **Add** to open the **Search** dialog in which you can search for groups via their names and display names.

   b Select groups in the table and click **OK** to add them to the list.

   Select groups in the list and click **Remove** to remove them from the list.

4 Click **OK** in the **Add User** dialog to add the user to the database.

The added user appears as a new child node to the **Users** node under **Security** in The Databases Tree.

The username in the **Name** field is the same one used to authenticate with the Model Manager server.

Managing Groups

You can manage Groups of users in a Model Manager server database from the Model Manager workspace. You can, for example, see all groups in the database or add a new group to the database. The latter is useful, for example, if the Model Manager server has been set up with an authentication scheme that does not support providing groups from an external credentials storage when a user logs in to the server.

See also *Groups* in the *Model Manager Server Manual*.

ADDING GROUPS

Click **Group** () in the **Users** section in the **Database** toolbar to open the **Add Group** dialog. The **Database** field shows the label of the database in which the group will be added.

1 Write the name of the group in the **Name** field.

2 Write a display name for the new group in the **Display Name** field. This is the name that will primarily be shown in the user interface.

3 You can add existing users or other groups as members to the new group under **Group members**.

   a Click **Add** to open the **Search** dialog in which you can search for users and groups via their names and display names.

   b Select users and groups in the table and click **OK** to add them to the list.

   Select users and groups in the list and click **Remove** to remove them from the list.

4 Click **OK** in the **Add Group** dialog to add the group to the database.

The added group appears as a new child node to the **Groups** node under **Security** in The Databases Tree.

Access Control

You can set up permissions to control access to items stored in a server database accessed via a Model Manager server. You can define these at various levels in The Databases Tree — from coarse-grained permissions set on Repositories to fine-grained permissions set on individual Models and Files.

When you authenticate with a Model Manager server, the server matches the provided username with a user stored in the database. Your user, as well as any groups that you are a member of, are checked whenever you perform a database action that requires authorization. The one exception is if you authenticate using an administrator account, in which case authorization checks automatically pass.

In this section, you will find answers to:

- Who can set permissions? See Owners.
- How are permissions set? See Granting Permissions
- What permissions can be set for database objects? See Permission Catalog.
- How do permissions combine to control access to models and files? See Permission Levels.
- How can the same permission assignments be reused between different database objects? See Reusing Permission Assignments Using Permission Templates.

All access control functionality is either hidden or disabled for a local database in the Model Manager workspace.

Owners

You control who can set permissions on database objects via *ownerships* — every object has an owner, which is one of the Users in the database. Except for administrators, only the owner can change a database object's permissions. The user that creates an object is automatically set as its initial owner.

TRANSFER OWNERSHIP

You can transfer the ownership of a database object to another user if you own the object, or if you are authenticated as an administrator.

Click **Owner** ( ) in the **Permissions** section of the **Database** toolbar. The **Owner** dialog is opened for the current selection in The Model Manager Window, The Databases Window, or The Maintenance Window (depending on which window has focus).

The **Database** field shows the label of the database that the object belongs to, the **Name** field shows the name of the object, and the **Current owner** field shows the username of the user that owns the object. If setting the ownership of multiple database objects at once, the **Name** field shows the static text `<Multiple selected>`. Under **New Owner**, write the name or display name of the user to transfer ownership to in the search field. Click **Search**. Select one of the users in the table and click **OK** to save the new ownership.

You can select different types of database objects when setting their common owner. This includes, for example, setting the same owner of a repository *and* a branch.

Granting Permissions

You can grant permissions to the following database objects:

- Repositories
- Branches
- Snapshots
- Models
- Files
- Tags

There is also a set of global permissions for the database itself — see Granting Database Permissions.

You can change the permissions for a database object if you own the object, or if you are authenticated as an administrator.

Click **Permissions** ( ) in the **Permissions** section of the **Database** toolbar. The **Permissions** dialog is opened for the current selection in The Model Manager Window, The Databases Window, or The Maintenance Window (depending on which window has focus).

The **Database** field shows the label of the database that the object belongs to, the **Name** field shows the name of the object, and the **Current owner** field shows the username of the user that owns the object.

1 In the **Permissions** list:

   a Select **None** to not set any permission requirements on the database object. See also Permission Levels to learn how an object may still be protected by a parent object in The Databases Tree.

   b Select **Public**, **Protected**, or **Private** to set one of the Predefined Permission Templates.

   c Select one of the previously created permission templates — see Creating your own Permission Templates.

   d Select **Custom** to set up Custom Permissions for the database object.

2 Click **OK** to save the permissions for the database object.

CUSTOM PERMISSIONS

Selecting **Custom** in the **Permissions** field enables you to customize the permissions for a database object. Granted permissions and their grantees — that is, users or groups — are shown in a table with the following columns:

- The type column — the type of the grantee represented by an icon.
- The **Name** column — the name of the grantee.
- The **Permissions** column — all granted permissions.

Click the **Add** button to add another user or group with a set of permissions.

1 In the opened **Add** dialog, write the names or display names of users and groups in the search field. Click **Search**.

2 Select a user or group in the table.

3 Under **Permissions**, select the permissions to grant the user or group. Use **Select all** to grant all permissions.

4 Click **OK** to add the granted permissions and the grantee to the table.

Select a row in the table and click the **Edit** button to modify already granted permissions. In the opened **Edit** dialog, select and clear the **Permissions** checkboxes accordingly.

Click the **Remove** button to remove a grantee from the table.

The set of grantees and corresponding granted permissions is known as an *access-control list* (ACL).

EVERYONE AND OWNER

You can grant permissions to a special group, **Everyone**, that automatically includes all users. Use this when you, for example, want to grant a subset of permissions to all users but restrict another subset of permissions to only some users and groups.

The user that is the owner of a database object can be implicitly granted permissions by adding an **Owner** from the **Add** dialog. Use this when you, for example, want to make sure that the permissions are transferred to the correct user when ownership is changed — see Transfer Ownership.

GRANTING DATABASE PERMISSIONS

You can grant permissions for actions that target the database itself. You can think of this as delegating a few administrative tasks that would otherwise be restricted to administrators. Only administrators can grant database permissions.

Right-click a database node in The Databases Tree and select **Database Permissions** ( ) to open the **Database Permissions** dialog. The **Database** field shows the label of the database.

You add, edit, and remove granted permissions and their grantees in a table in the same way as when selecting **Custom** in the **Permissions** field for a database object, except that Everyone and Owner are not available — see Custom Permissions.

Permission Catalog

This section contains a catalog of all permissions available for database objects.

DATABASE

TABLE 2-1: ALL AVAILABLE PERMISSIONS FOR THE DATABASE ITSELF.

PERMISSION	DESCRIPTION
Add repositories	Allowed to create a new repository.
Cleanup	Allowed to perform manual data deduplication and other database cleanup actions.
Clear computed data	Allowed to clear built, computed, and plotted data of model versions.
Index	Allowed to perform maintenance operations on search indexes.
Manage security	Allowed to create, delete, edit, and restore users, groups, and permission templates.
Permanently delete items	Allowed to permanently delete versions of models and files.
See disk space usage	Allowed to see the disk space usage of model and file versions.

REPOSITORY

TABLE 2-2: ALL AVAILABLE PERMISSIONS FOR REPOSITORIES.

PERMISSION	DESCRIPTION
Create branch	Allowed to create a new branch inside the repository.
Delete repository	Allowed to delete the repository.
Record snapshot	Allowed to record a new snapshot inside the repository.
Restore repository	Allowed to restore the repository when deleted.
Save in repository	Allowed to save commits as well as perform any other save operation inside the repository. This is, for example, a necessary permission for saving branches, snapshots, and item versions inside the repository.
Save repository	Allowed to save the repository itself to, for example, rename it.
See repository	Allowed to see the repository in the user interface. This is, for example, a necessary permission for browsing items inside the repository.
Set default branch	Allowed to set the default branch inside the repository.

BRANCH

TABLE 2-3: ALL AVAILABLE PERMISSIONS FOR BRANCHES.

PERMISSION	DESCRIPTION
Delete branch	Allowed to delete the branch.
Restore branch	Allowed to restore the branch when deleted.
Save in branch	Allowed to save commits as well as perform any other save operation inside the branch. This is, for example, a necessary permission for saving item versions on the branch.
Save branch	Allowed to save the branch itself to, for example, rename it or change search capabilities.
See branch	Allowed to see the branch in the user interface. This is, for example, a necessary permission for browsing all item versions on the branch.

SNAPSHOT

TABLE 2-4: ALL AVAILABLE PERMISSIONS FOR SNAPSHOTS.

PERMISSION	DESCRIPTION
Delete snapshot	Allowed to delete the snapshot.
Restore snapshot	Allowed to restore the snapshot when deleted.
Save snapshot	Allowed to save the snapshot itself to, for example, rename it.
See snapshot	Allowed to see the snapshot in the user interface. This is, for example, a necessary permission for browsing the latest item versions with respect to the snapshot's referenced commit.

MODEL

TABLE 2-5: ALL AVAILABLE PERMISSIONS FOR MODELS.

PERMISSION	DESCRIPTION
Delete model	Allowed to delete the model.
Open model	Allowed to open the model's versions.
Save model	Allowed to save versions of the model.

FILE

TABLE 2-6: ALL AVAILABLE PERMISSIONS FOR FILES.

PERMISSION	DESCRIPTION
Delete file	Allowed to delete the file.
Download file	Allowed to download the file's versions.
Save file	Allowed to save versions of the file.

TAG

TABLE 2-7: ALL AVAILABLE PERMISSIONS FOR TAGS.

PERMISSION	DESCRIPTION
Delete tag	Allowed to delete the tag.
Save tag	Allowed to save versions of the tag.

Permission Levels

When Model Manager authorizes a database action that targets item versions on a branch, it consults up to three levels of protection — the repository that the items belong to, the branch the item versions belong to, and, possibly, the items themselves. An analogous level consultation is done for item versions recorded by a snapshot.

PERMISSION COMBINATIONS FOR BRANCHES

The necessary permission combinations for possible database actions that target item versions on a branch are summarized as follows:

TABLE 2-8: NECESSARY PERMISSION COMBINATIONS FOR PERFORMING POSSIBLE DATABASE ACTIONS THAT TARGET ITEMS ON A BRANCH.

ACTION	REPOSITORY	BRANCH	ITEM
Browse and search item versions	See repository	See branch	N/A
Open a model in the COMSOL Desktop	See repository	See branch	Open model
Download data files	See repository	See branch	Download file
Create a new item	Save in repository	Save in branch	N/A
Assign tags to items	Save in repository	Save in branch	N/A
Save a new model version	Save in repository	Save in branch	Save model
Save a new file version	Save in repository	Save in branch	Save file
Save a new tag version	Save in repository	Save in branch	Save tag
Delete a model	Save in repository	Save in branch	Delete model

TABLE 2-8: NECESSARY PERMISSION COMBINATIONS FOR PERFORMING POSSIBLE DATABASE ACTIONS THAT TARGET ITEMS ON A BRANCH.

ACTION	REPOSITORY	BRANCH	ITEM
Delete a file	Save in repository	Save in branch	Delete file
Delete a tag	Save in repository	Save in branch	Delete tag

An important takeaway from Table 2-8 is that if you grant a **See repository** permission and a **See branch** permission to users, they will be able to see settings of all models and files with versions on the branch via, for example, The Model Manager Window, The Databases Window, or The Versions Window. If some models contain sensitive information exposed through, for example, the **Contents** section in the Model Settings, you must restrict access to the repository or branch to protect it. Limiting who can open an individual model should not be relied upon to keep its inner workings secret.

You may wonder why access to models and data files cannot be controlled via their assigned tags? After all, tags have many similarities with folders on the file system, and access to files on the file system can typically be controlled via folder permissions. The motivation is twofold:

- Models and files can have several tags, that is, they can appear in multiple places in The Tag Tree.
- Tag assignments are version controlled via commits — see Adding and Removing Tag Assignments.

This makes it hard to reason about the access granted to a model or file. An item could perhaps be protected under one tag's permissions but exposed under another tag's permissions. An item could be protected under a tag at the present time, but older versions may be exposed in the commit history if the item was previously tagged by another tag with less restrictive permissions.

PERMISSION COMBINATIONS FOR SNAPSHOTS

The necessary permission combinations for possible database actions that target the item versions recorded by a snapshot are summarized as follows (there are no database actions that save to the database):

TABLE 2-9: NECESSARY PERMISSION COMBINATIONS FOR PERFORMING POSSIBLE DATABASE ACTIONS THAT TARGET THE ITEM VERSION RECORDED BY A SNAPSHOT.

ACTION	REPOSITORY	SNAPSHOT	ITEM
Browse and search item versions	See repository	See snapshot	N/A
Open a model in the COMSOL Desktop	See repository	See snapshot	Open model
Download data files	See repository	See snapshot	Download file

THE PERMISSION REQUIREMENTS DIALOG

Model Manager performs a preemptive authorization check whenever you open a window or dialog that is intended for saving to the database. If the check fails, a link button () whose button text summarizes why the check failed is shown.

Click the link button to open a dialog with the necessary permission requirements shown in a table. Select the **Show only nongranted required permissions** checkbox to only see permission requirements that you lack.

Reusing Permission Assignments Using Permission Templates

You can create your own Permission Templates to reuse permission assignments for different database objects. This saves you the tedious work of manually granting the same set of permissions to users and groups for multiple database object. Creating your own permission templates also enables you to propagate permission requirement changes to multiple database objects by only updating the permissions in one place — the permission assignments of the permission template itself.

Updating the permission assignments for a permission template affects not only future applications of the template but also the permissions of database objects to which the template has already been applied.

PREDEFINED PERMISSION TEMPLATES

Model Manager comes with three predefined permission templates for each of the database object types — **Public**, **Protected**, and **Private**. Dividing the permissions in the Permission Catalog into *read* permissions and *write* permissions:

TABLE 2-10: THE PERMISSIONS GRANTED FOR THE PREDEFINED PERMISSION TEMPLATES.

PERMISSION TEMPLATE NAME	GRANTED READ PERMISSIONS	GRANTED WRITE PERMISSIONS
Public	Everyone	Everyone
Protected	Everyone	Owner
Private	Owner	Owner

CREATING YOUR OWN PERMISSION TEMPLATES

You can create new permission templates for models or files in the Model Manager workspace.

Click **Permission Template** ( ) in the **Permissions** section of the **Database** toolbar to open the **Add Permission Template** dialog. The **Database** field shows the label of the database in which the template is added.

1. Write the name of the permission template in the **Name** field.
2. Select whether to add a **Model** or **File** permission template in the **Type** field.
3. You add, edit, and remove granted permissions and their grantees in a table in the same way as when selecting **Custom** in the **Permissions** field for a database object — see Custom Permissions.
4. Click **OK** to add the new permission template to the database.

The added permission template appears as a new child node to the **Permission Templates** node under **Security** in The Databases Tree.

Maintenance

A challenge with archiving simulation data is the inevitable growth of disk space usage, regardless if such data is stored in a database or on the file system. Oftentimes you will not be able to persist some, or all, of the Built, Computed, and Plotted Data generated by your models for longer periods of time due to the sheer amount of storage capacity that would be required.

In this section, you will find various ways in which Model Manager can help you keep the total size of your simulation data under control via maintenance operations. Performing these maintenance operations for a server database accessed via a Model Manager server typically requires that you have authenticated using an administrator account or have been granted the relevant database permissions — see Granting Database Permissions.

- Estimating Disk Space Usage
- Built, Computed, and Plotted Data
- Permanently Deleting Models and Data Files
- Collecting Models and Files for Maintenance
- The Maintenance Window

Estimating Disk Space Usage

The disk space required to store a model version in a database depends on the size of the model contents and the size of both computed and other data associated with the model. For a file version, it depends on the size of the binary or text data of the individual file resources.

Model Manager enables you to estimate the disk space usage for various sets of versions including, but not limited to:

- A selection of versions in a window.
- All versions belonging to a selection of items in a window.
- All versions matching a search in the **Model Manager** window, independent of if all versions are currently shown in the table or if there are more results to retrieve via **Show More** ( ).
- All versions currently targeted in the **Maintenance** window.

With **Latest Versions for Location** ( ) selected in the **Model Manager** window, click **Disk Space Usage** ( ) in the **Statistics** section on the **Database** toolbar and select **Version Selection** ( ) to open The Estimated Disk Space Usage Dialog for the current selection. This will compute the disk space usage of the latest versions of items for the searched location. To compute the disk space usage of *all* versions belonging to the same items as the selected versions, select **Item Selection** ( ) in the **Disk Space Usage** menu ( ).

With **All Versions in Database** ( ) selected in the **Model Manager** window, you can also select **Search Result** ( ) in the **Disk Space Usage** menu ( ). This will compute the disk space usage of all versions that are matched by the current search term and applied search filters.

As a special case for the **Search Result** ( ) option, you can leave out any search term and search filters in the **Model Manager** window to obtain the disk space usage of all versions in the database.

Estimating Disk Space Usage as a Maintenance Operation

THE ESTIMATED DISK SPACE USAGE DIALOG

The **Estimated Disk Space Usage** dialog — opened, for example, via the **Disk Space Usage** menu ( ) on the **Database** toolbar — contains an estimate of the total disk space usage of a set of model and file versions. A textual description of the set of versions is shown under **Selection type**. Under **Disk space**, you will find:

- **Versions** — the number of versions included in the estimate.
- **Data** — the disk space usage of the versions in bytes.
- **Computed data** — the disk space usage of all Built, Computed, and Plotted Data of the versions in bytes.

A variant of the dialog is shown when estimating the disk space usage of all versions of items via the **Item Selection** option ( ) in the **Disk Space Usage** menu ( ). As a convenience, all versions of the items' drafts are automatically included in the estimate: Under **Disk space - total**, you will find the total disk space usage of both the items and

all drafts created from those items. Under **Disk space - drafts**, you will find the disk space usage of only the drafts.

Summing the estimated disk space usage numbers obtained for different sets of versions may result in an overestimate as identical data used by different versions is never duplicated in the database, but will still be included in the usage estimate for each set.

Built, Computed, and Plotted Data

A significant contribution to the overall disk space required for storing your models comes from generated simulation data. This includes built geometries and meshes, computed solutions, and results plots — *built, computed, and plotted data* for short. For large models, it is often unfeasible for you or your organization to keep this data around for longer periods of time, preferring instead to regenerate the data by rerunning a simulation as needed.

There are two ways in which you can avoid storing built, computed, and plotted data in your databases:

- Exclude the data altogether already when saving a model to the database. See the Save section of The Root Settings Window in the *COMSOL Multiphysics Reference Manual*.
- Delete the data directly in the database by clearing it from all model versions that reference it — see also Clearing Computed Data as a Maintenance Operation.

You can see the COMSOL Multiphysics version that a model version was saved in from the **Saved in** field in Model Settings. This is useful when you want to regenerate built, computed, or plotted data for a model version whose data is no longer present in the database using that same COMSOL Multiphysics version.

To clear built, computed, and plotted data from model versions in the database, you can for example:

- Right-click versions in The Versions Window for a model and select **Clear Computed Data** ( ).
- Select one of the options in the **Clear Computed Data** menu ( ) on The Maintenance Toolbar, which is available for versions in The Maintenance Window when the window has focus.

Rebuilding a geometry using a COMSOL Multiphysics version or operating system different from the one the previously cleared geometry was built in may require you to first adapt geometry feature settings — possibly resulting in a rebuilt geometry that differs from the original one. This could affect simulation outcomes. A similar remark holds true when rebuilding a previously cleared mesh.

Clearing Computed Data as a Maintenance Operation

Permanently Deleting Models and Data Files

You can permanently delete versions of models and data files in a Model Manager database as a way of reducing disk space usage. You may, for example, want to get rid of old drafts that you no longer have any use for or some particular version that is unsuitable to keep in the database.

Beware that permanently deleted versions cannot be recovered unless you have a backup of your database.

You can permanently delete versions of models and files by:

- Right-clicking versions in The Versions Window for an item and selecting **Permanently Delete** (✕).
- Select one of the options in the **Permanently Delete** menu (✕) on The Maintenance Toolbar, which is available for versions in The Maintenance Window when the window has focus.

Since Model Manager does not duplicate unchanged data when saving versions of a model, do not be surprised if permanently deleting a single version does not reclaim much disk space.

Permanently deleting versions will remove most traces of them ever existing in the database. If you find, for example, a commit for which the **Changes** table is surprisingly empty in the Commit Settings, chances are that this commit involved saving a version that now has been permanently deleted. The commit comment, if any, may give a clue of what saving that version entailed.

Permanently Deleting Versions as a Maintenance Operation

Collecting Models and Files for Maintenance

The **Versions** window may be used to perform maintenance operations on a few versions belonging to the same model or file. Select the targeted versions, right-click, and then select the maintenance operation you have in mind from the context menu.

Maintenance operations that target a more complex collection of versions typically involves the following steps:

1. Finding the versions you want to target by searching and filtering in the **Model Manager** window.
2. Adding versions from the search result in the **Model Manager** window to the **Maintenance** window.
3. Selecting a maintenance operation that targets either a selection of versions or all versions in the **Maintenance** window from the **Maintenance** toolbar.

Steps one and two may be repeated before proceeding to step three.

The sets of versions that you can add to the **Maintenance** window are similar to the ones available when Estimating Disk Space Usage:

- A selection of versions in the **Model Manager** window.
- All versions belonging to the same items as a selection of versions in the **Model Manager** window.
- All versions matching the search in the **Model Manager** window, independent of if all versions are currently shown in the table or if there are more results to retrieve via **Show More** ( ).

With **Latest Versions for Location** ( ) selected in the **Model Manager** window, click **Add to Maintenance** ( ) in the **Maintenance** section on the **Database** toolbar and select **Item Selection** ( ) — all versions belonging to the same items, or the items' drafts, of the current selection are added to the **Maintenance** window.

With **All Versions in Database** ( ) selected in the **Model Manager** window, select **Version Selection** ( ) in the **Add to Maintenance** menu ( ) to add the currently selected versions to the **Maintenance** window. Select **Search Result** ( ) in the **Add to Maintenance** menu ( ) to add the current search term and applied search filters as an "implicit selection rule for versions" to the **Maintenance** window.

The **Search Result** option ( ) is by far the most powerful one. It enables you, for example, to target:

- All draft versions with a saved date older than some fixed time by applying a Item Save Type filter and a Saved filter.
- All versions with built, computed, and plotted data with a file size larger than some value by applying a Computed Data filer.
- All versions saved by yourself by applying a Saved By filter.

At most one search result can be included in the **Maintenance** window if the search expression uses full text search words. Consider replacing the full text search words with Title, Description, and Tag filters if you want to add multiple search results.

The **Search Result** option ( ) enables you to target versions in the **Maintenance** window without requiring you to first having to retrieve them all as separate table rows in the **Model Manager** window. This is useful, for example, when the versions number in the thousands.

The Maintenance Window

The **Maintenance** window in the Model Manager workspace may be used to perform maintenance operations targeting versions added to the window from The Model Manager Window and The Databases Window. This includes, for example, clearing Built, Computed, and Plotted Data from model versions or Permanently Deleting Models and Data Files. You can also use it as a browsing tool for exploring the total version footprint of items that you have added to the window from other windows in the workspace.

Collecting Models and Files for Maintenance

The model and file versions are shown in a table, with the versions grouped together in *top level table rows* based on how they were added to the window:

- Versions added via a **Version Selection** () are shown first with one table row per version.
- Versions added as an **Item Selection** () are shown as one table row per item. Click the triangle symbol next to the icon to show versions belonging to the item or the item's drafts as separate table rows.

  All versions in all repositories and branches that an item has been saved to and that you are authorized to see are included.
- Versions added as a **Search Result** () are shown as a single table row representing the search expression — click the triangle symbol to show versions matching the search expression.

The versions under an **Item Selection** () or a **Search Result** () are sorted in chronological order. A maximum of 100 most recently saved versions are initially shown. Click the triangle symbol for a **Show More** table row () to append the next 100 versions to the table. You can set another value for this default page size in the **Result Page Size** field on the **Model Manager** page in the **Preferences** window.

The table columns are:

- The **Target** column — a description of the targeted versions. The title in the case of a single version, a representative title chosen from one of its versions in the case of an item, or a summary of the search expression in the case of a search result.
- The **Saved** column — the time when a version was saved.

- The **Saved By** column — the name of the user that saved a version.
- The **Repository** column — the repository that a version was saved in.
- The **Branch** column — the branch that a version was saved in.
- The **Comments** column — the optional comment provided when a version was saved.

Viewing the version table rows under an item in the **Maintenance** window as "the version history" of a model or file can often be misleading. Since versions of a model's drafts are included in the table, and these drafts may have been worked on in parallel by multiple users, going from one version to the next in the table may not represent how the model has evolved over time. A similar observation holds when a model or file has been worked on in parallel on multiple branches. A more honest version history of an item is best viewed from The Versions Window.

THE MAINTENANCE WINDOW TOOLBAR

The toolbar in the **Maintenance** window contains the following toolbar buttons:

- Click the **Refresh** button ( ) to refresh the table in case any new versions have been saved. The table will automatically refresh if you save a new version to the database from the COMSOL Desktop.
- Click the **Clear** button ( ) to the clear the table.
- Click the **Remove from Maintenance** button ( ) to remove top level table rows from the table.

THE SETTINGS WINDOW FOR ITEMS

The top level table row for an **Item Selection** ( ) in the **Maintenance** window is a representation of the item itself in the database. The **Settings** window shows a summary of the item and its associated versions in the database.

The General Section

This section shows the database of the item in the **Database** field and its item save type in the **Item save type** field.

The Disk Space Usage Section

This section shows disk space usage statistics of the item and all its drafts. The fields are:

- **Versions** — the total number of versions of the item and the item's drafts.

- **Data** — the disk space usage of the versions in bytes.
- **Computed data** — the disk space usage of all Built, Computed, and Plotted Data of the versions in bytes.

ESTIMATING DISK SPACE USAGE AS A MAINTENANCE OPERATION

You can estimate the total disk space usage of versions targeted in the **Maintenance** window. From the **Disk Space Usage** menu ( ) in The Statistics Section on The Maintenance Toolbar, select **Version Selection** ( ) or **Item Selection** ( ) to estimate the disk space usage for the current table selection. Select **All in Maintenance** ( ) to estimate the disk space usage for all versions in the **Maintenance** window, independent of any table selection or if all **Show More** table rows ( ) have been expanded.

Estimating Disk Space Usage

CLEARING COMPUTED DATA AS A MAINTENANCE OPERATION

You can clear built, computed, and plotted data of model versions targeted in the **Maintenance** window. From the **Clear Computed Data** menu ( ) in The Computed Data Section on The Maintenance Toolbar, select **Version Selection** ( ) to clear such data from the current selection of versions. For a selection of items, select **Item Selection** ( ) to clear all versions of the models *and* their drafts. To *only* clear the item's drafts, select **Drafts of Item Selection** ( ). Select **All in Maintenance** ( ) to clear such data of all versions in the **Maintenance** window, independent of any table selection or if all **Show More** table rows ( ) have been expanded.

You will be asked to confirm that you want to clear the built, computed, and plotted data of the model versions.

Clearing built, computed, and plotted data of a model version is permanent and cannot be undone. You will either need to regenerate the data by recomputing it or restore the database in its entirety from a backup.

Clearing computed data of versions in the database is done via batched transactions. You may stop the clearing at any point by clicking the **Stop** button, but all versions processed up to that point will already have been permanently cleared. Clearing a model version in itself does not physically delete the computed data — the data may, for example, be shared with other versions via deduplication. Once all batches of

versions have been cleared, Model Manager will automatically proceed by permanently deleting all computed data that is no longer used by any version via a separate database cleanup operation.

Database Cleanup

PERMANENTLY DELETING VERSIONS AS A MAINTENANCE OPERATION

You can permanently delete versions targeted in the **Maintenance** window. From the **Permanently Delete** menu () in The Versions Section on The Maintenance Toolbar, select **Version Selection** () to permanently delete the current selection of versions. For a selection of items, select **Item Selection** () to permanently delete all versions of the items *and* their drafts. To *only* permanently delete the item's drafts, select **Drafts of Item Selection** (). Select **All in Maintenance** () to permanently delete all versions in the **Maintenance** window, independent of any table selection or if all **Show More** table rows () have been expanded.

You will be asked to confirm that you want to proceed with the permanent deletion of the versions.

Permanently deleting versions cannot be undone. You will need to restore the database in its entirety from a backup in case of an accidental deletion.

Model Manager does not allow permanently deleting a version if that version is referenced by another version that is *not* set to be deleted. If you include such a version when targeting an explicit selection in the **Maintenance** window, the permanent deletion will be immediately canceled by Model Manager with nothing deleted in the database. If you include such a version for the **All in Maintenance** option (), that version will be excluded from the deletion but the deletion of other versions will still be allowed to proceed.

Reference-tracking in Model Manager

Permanently deleting versions in the database is done via batched transactions. You may stop the deletion at any point by clicking the **Stop** button, but all versions processed up to that point will already have been permanently deleted. Any data that

potentially may be shared with other versions via deduplication will not be deleted during this initial batch processing. Once all batches of versions have been deleted, Model Manager will automatically proceed by permanently deleting data that is no longer used by any version via a separate database cleanup operation.

Database Cleanup

Database Administration

In this section, you will find details on how you can administer your configured databases within the Model Manager workspace. You will also learn how to move, delete, and perform backups of databases stored locally on your computer.

- Database Configurations
- Updating Search Index
- Database Cleanup
- Compacting Local Databases
- Moving and Deleting Local Databases
- Backup for Local Databases

Database Configurations

You can manage added databases in the COMSOL Desktop from the top nodes in The Databases Tree. Right-click and select **Delete Configuration** (✕) to remove an added database from the COMSOL Desktop. This will only delete the configuration for the database, not the database itself.

DATABASE SETTINGS

The **Settings** window for a database shows its configuration in the COMSOL Desktop. Click the **Save** button (💾) to update the current configuration.

Saving the configuration automatically disconnects the database from the COMSOL Desktop. Click anywhere in the Model Manager workspace that requires database access to reconnect.

The General Section

This section shows the label of the database in the **Label** field and its optional *alias* in the **Alias** field. When set, the alias must only contain lowercase Latin alphabet

characters, numerical characters, underscores, and dashes. The alias must not contain more than 100 characters.

Both the label and the alias field are only stored in the configuration used to connect to the Model Manager database — they are not stored inside the database itself. If you remove a database from the COMSOL Desktop by deleting its configuration and then add it again, you will need to write these values anew.

The alias of a Model Manager database can be used as a read-friendly, alternative, identifier when accessing the database via the Model Manager API.

The Storage Section

This section is only shown for a local database. The fields are:

- **Database file**. The path to the SQLite® database file on the file system.
- **Resources directory**. The path to the resources directory on the file system.
- **Search indexes directory**. The path to the search indexes directory on the file system.

A Local Database on the File System

Click the **Show in System Explorer** button () to open the database directory in the system explorer native to the operating system.

Click the **Compact** button ( ) to compact the database file — see Compacting Local Databases.

Click the **Refresh Connection** button () to refresh the connection to the local database. The current connection, if any, is closed, all changes made in the window are saved, and then a new connection is established. You may find this useful if another, no-longer running, COMSOL session was already connected to the database when the current session connected, and you now want to enable full search capabilities for the current session — see Opening a Local Database from Multiple COMSOL Multiphysics Processes.

The Connection Section

This section is only shown for a server database accessed via a Model Manager server. The fields are:

- **Server**. The server address to the Model Manager server.
- **Require secure connection**. Selected if the network connection is required to use a secure connection, with transport layer security provided by HTTPS (as opposed to plain HTTP). A warning message is shown if the checkbox is cleared.
- **Username**. The username used when connecting to the server.
- **Password**. The password used when connecting to the server. The current password is not accessible.

  The **Password** field is hidden when running COMSOL Multiphysics in client–server mode — see also Connecting Via a COMSOL Multiphysics Server.
- **Remember password**. Selected if the password is remembered between program sessions. The password is stored in an encrypted form on the local file system.

Connecting to a Server Database

COMSOL Desktop disconnects from a connected Model Manager server when you click the **Save** button ( ) in the **Settings** window. You can reestablish the connection by, for example, activating the database in the Model Manager workspace — see Activating a Database.

Click the **Test Connection** button ( ) to test if the filled-in values in the **Connection** section can be used to successfully connect to the Model Manager server database.

Click the **Refresh Connection** button () to refresh the connection to the server database. The current connection, if any, is closed, all changes made in the window are saved, and then a new connection is established.

The Information Section

This section is only shown for a server database accessed via a Model Manager server. When connected, the **Server version** field displays the current version number of the Model Manager server. Otherwise, a message is displayed informing you that the server database is not connected. Click the message to reconnect.

CHANGING USER ACCOUNT CREDENTIALS

When connecting to a Model Manager server database via The Add Database Window, Model Manager stores the username and — if you selected the **Remember Password** checkbox — the (encrypted) password in the created database configuration. This enables you to automatically connect to the server database in future program sessions.

You can change the user account credentials you use to connect to the server database. This includes, for example, using a new password or if you want to connect with a different user account.

1. Select the database node in The Databases Tree.
2. In The Connection Section in the **Settings** window, write the username and password of the user account you want to connect with under **User**.
3. Click the **Test Connection** button ( ) to test that the account credentials are correct.
4. Click the **Save** button ( ) to save the new account credentials in the database configuration.

   The new user account will be used in the current and any future program sessions until you repeat the previous steps.

Updating Search Index

Model Manager uses a specialized *search index* when searching and filtering item versions. Under normal program execution, Model Manager automatically pushes any updates made to items in the database to this search index, both for a local database and for a server database. This push may fail, however — for a Model Manager server, for example, due to intermittent network issues when the Model Manager server process tries to communicate with the process managing the search index.

You can manually trigger the push of changes to the search index, thereby letting the search index catch up with any changes made in the database. Right-click a database node in The Databases Tree and select **Index** ( ) to push changes made on all branches. Do the same for a branch node to only push changes made on that branch.

Database Cleanup

When you clear built, computed, and plotted data of model versions, or permanently delete model and file versions, Model Manager automatically detects and deletes unused data that may have been previously shared between versions via data

deduplication. If the process is unexpectedly terminated while this cleanup is ongoing, for example if there is a power failure, that data will remain in the database until the next time you run such a maintenance operation. Right-click a database node in The Databases Tree and select **Cleanup** ( ) to manually run the cleanup. This may take a few seconds up to several minutes depending on the size of the database.

- Clearing Computed Data as a Maintenance Operation
- Permanently Deleting Versions as a Maintenance Operation

Compacting Local Databases

You can optimize a local database by compacting its SQLite® database file. Right-click the database node in The Databases Tree and select **Compact** ( ) to open the **Compact** dialog. The **File size** field shows the current size of the SQLite® database file and the **Estimated file size after compacting** field shows the estimated size after the file has been compacted. Click **OK** to compact the file. This may take a few seconds up to several minutes depending on the size of the database file.

A Local Database on the File System

Moving and Deleting Local Databases

You can move a Model Manager database stored locally on your computer to another file system location:

1. Open The Databases Window in the Model Manager workspace and select the top database tree node for the database you want to move. In The Settings Window, under **Storage**, note the file system location of your database. You can also click the **Show in System Explorer** button ( ) to open the database directory in the system explorer native to the operating system.
2. Right-click the database tree node and select **Delete Configuration** ( ) to delete the configuration for the database.
3. Move the directory containing your database to the new location on your computer's file system — see also A Local Database on the File System.
4. Click the **Add Database** button ( ) in the **Database** section in the **Home** toolbar.

5 Click the **Open Local Database** button () and select the SQLite® database file in the moved database directory — see also Opening a Local Database. Model Manager will automatically connect to the local database in the new location.

To permanently delete a local Model Manager database, follow the previous steps but instead delete the database directory on the file system after you have deleted the configuration for the database.

Backup for Local Databases

Local Model Manager databases can be backed up using any regular backup software used for backing up the workstation. Make sure to include all files and subdirectories in the database directory as described in A Local Database on the File System: the SQLite® database file, the `resources` directory, and the `indexes` directory. You may reduce the total disk space usage of your backups by optionally skipping the `indexes` directory as its file contents can always be recreated by Model Manager.

It may be tempting to place the database directory in a local folder on your computer that is automatically synchronized using cloud storage software with a remote cloud drive. This is not supported, however, and COMSOL Multiphysics will report an error if it detects such a file system path. For more information, see https://www.comsol.com/support/knowledgebase/1295.

To restore a previously backed-up database directory, copy the database directory and all its file contents from your backup location to your computer and follow the steps in Opening a Local Database.

See the *Model Manager Server Manual* for backup and restore of a Model Manager server database.

3

Searching and Filtering

In this chapter you will learn how to search for models and data files in a Model Manager database. You will see how you can match such items by performing both full text searches, as well as by applying various filter criteria. You will also learn how you can search "deep" within the model tree, matching models on their node properties, parameters, features, and other settings.

In this chapter:

Searching Versions

Whenever you search for model versions and file versions in a Model Manager database, you can select between two *search modes*: **Latest Versions for Location** ( ) or **All Versions in Database** ( ). These two modes correspond to searching in the subset of versions that are the *latest versions* saved on a particular branch or in the set of *all versions* ever saved to the database, respectively.

You can toggle between the two search modes by clicking the menu button next to the **Search** button ( ) in a window with Model Manager search functionality — that is, the **Model Manager** window, the **Open** window, the **Select File** window or the **Select Model** window. You can also click the expand button next to the menu button and select the desired search mode from the menu.

A typical use case for selecting the first mode is when you want to open a model for editing in the COMSOL Desktop. For that, you generally want to be sure that you are starting off from the latest version of the model on its branch. The second mode is useful when you want to find versions in order to perform maintenance operations on them. This includes, for example, computing versions' total disk space usage or clearing versions' built, computed, and plotted data.

The **Latest Versions for Location** ( ) search mode is the default mode in all windows with Model Manager search functionality. It is also the recommended search mode for typical workflows involving the Model Manager — see Databases in the COMSOL Modeling Environment.

Searching Latest Versions for Locations

There are two ways you can specify the subset of latest versions that you want to search for when the **Latest Versions for Location** ( ) search mode is selected:

- The latest versions on a branch at the present time. This is the most commonly used subset for most workflows in the COMSOL Desktop that involve the Model Manager search functionality.
- The versions that were the latest on a branch at some particular point in time in the past. This can be useful, for example, when you need to go back to the state of a branch as it was at the time of some project milestone.

These two subsets correspond to searching with respect to different types of Locations in a database. In this section you will learn how the search capabilities of Model Manager differ when searching in such locations.

SEARCHING IN BRANCHES

Select a branch in The Select Location Dialog to search for the latest versions of models and files in that branch at the present time. This is also the default location type when you select a database from the list on the left in the **Open**, **Select File**, and **Select Model** windows, or in the **Model Manager** window when Activating a Database in the Model Manager workspace.

The location link button used to open The Select Location Dialog is hidden in the **Open**, **Select File**, and **Select Model** windows when a database only contains a single branch and no snapshots.

Which search tools are available when searching for the latest versions in the branch is determined by the value selected in the **Search** list found in the branch's **Settings** window — see Branch Settings. Select **Item fields and content** in the list to enable the full Model Manager search functionality for the branch. This includes both all Full Text Search tools and all Item and Content Filters. Select **Only item fields** to restrict the filtering support to those that target the field values of models and files, not filters targeting their contents.

The **Item fields and content** value is the preferred and default selection for the main branch in a repository. There is, however, a cost in terms of disk space usage by the additional search data required for the corresponding search index — see also Updating Search Index. You may want to select **Only item fields** when:

- You create a new branch off the main branch that contains only a few models and files. Finding a particular model or file is then a matter of simply scanning a short list of items — that is, no advanced content filtering is required.
- You create a new branch off the main branch on which models and files will rarely be added or updated. Advanced content filtering can then be performed on the main branch — selecting the corresponding version of an item on the new branch

may be done, for example, from The Versions Window or from The Maintenance Window.

When Opening a Local Database from Multiple COMSOL Multiphysics Processes, the search functionality will be further limited to that used when Searching in Snapshots and Commits for all processes except the first one to open the database.

SEARCHING IN SNAPSHOTS AND COMMITS

Select a snapshot in The Select Location Dialog, or right-click a commit and select **Search in Commit** ( ) in The Commits Window, to search for the versions that were the latest at a fixed point in time — the saved date of the corresponding commit — on a particular branch. You may find this useful if, for example, you want to find a version of an old model or file that has been deleted on the branch, or if you perhaps want to see how the branch was organized with tags at the time of the commit.

Searching the latest versions with respect to a snapshot or a manually selected commit is limited functionality-wise as compared to Searching in Branches. You can perform full text searches by selecting **Text** or search on the titles of assigned tags by selecting **Tags** from the list next to the **Search** button ( ) in The Model Manager Window. Other search windows offer only full text search. You can also apply a separate filter on file type extensions from a list in the **Select File** window. Other Item and Content Filters or The Model Manager Search Syntax are not available.

The **Add Filter** menu button ( ) is disabled in the **Open**, **Select File**, **Select Model**, and **Model Manager** windows' toolbars when you search with respect to a snapshot or commit.

Searching All Versions in the Database

When the **All Versions in Database** ( ) search mode is selected, you can match any item version in the database irrespective of which branch it was saved to or if it is the latest version of its corresponding item. The location link button, available in a search window's upper right corner when Searching Latest Versions for Locations, is hidden for this search mode and, in the case of the **Model Manager** window, replaced by the label of the database.

The **All Versions in Database** ( ) search mode lets you browse and search the database in its entirety at the expense of possibly hard-to-navigate search results with often identically-titled versions. It is primarily suited for when you want to define a selection of versions based on various filter criteria in order to perform maintenance operations on them via The Maintenance Window or when you want to find an older version by applying filters on commit comments, saved dates, or other history-related item fields.

The search mode supports full text search and all Item Filters expect for Tag. No Content Filters except Part are supported.

Unsupported filters are disabled in the **Add Filter** menu ( ) when you select the **All Versions in Database** ( ) search mode in the **Open**, **Select File**, **Select Model**, and **Model Manager** windows.

When Opening a Local Database from Multiple COMSOL Multiphysics Processes, the search functionality in the **All Versions in Database** ( ) search mode will be further limited to only support full text search for all processes except the first one to open the database. No filters are supported

Sorting Search Results

The results returned when searching for item versions in a Model Manager database are sorted alphabetically on their titles by default. When a branch is selected in the **Latest Versions for Location** ( ) search mode or when using the **All Versions in Database** ( ) search mode, you may select one of the following sort fields:

- **Title** ( ) — sort on the title of an item version.
- **Saved** ( ) — sort on the date when an item version was saved.
- **Size** ( ) — sort on the estimated disk space usage of an item version if the version is exported to the file system.
- **Computed Data** ( ) — sort on the estimated disk space usage of an item version's built, computed, and plotted data if the version is exported to the file system.
- **Rank** (123) — sort on the relevance ranking of an individual item version found in the search result. Item versions that are deemed more relevant for the search expression are sorted before those with less relevance.

The **Title** sort field uses ascending sort order by default; the other sort fields use descending sort order by default. All but the **Rank** sort field support switching between ascending and descending sort order.

The **Open**, **Select File**, **Select Model**, and **Model Manager** windows will each remember the last applied sort field and sort order made in the window between program sessions.

Changing sort field or sort order is not supported when Searching in Snapshots and Commits — the search result is always sorted alphabetically on titles.

Search History

You can reapply previous searches made in a Model Manager database via the **Search History** menu list ( ) in the **Model Manager**, **Open**, **Select File**, and **Select Model** window's toolbars. This may be useful, for example, if you have applied various filters in one window and now quickly want to reuse those filters in another window.

A new entry is added to the top of the list every time you perform a new search — either by clicking the **Search** button ( ) or by pressing Enter. The entry contains the search term used in the search field as well as any applied filters — see Applied Filter Pills. At most 20 unique search entries are shown in the list. If a previous search made in another window used filters that are not applicable in the current window, the corresponding entry will be excluded in the current window's search history list.

Selecting an entry resets the search term and the filters, thereby replacing any existing search term and filters already set in the window. Select **Clear Search History** ( ) to clear the list of all its entries.

Each Model Manager database you add to the COMSOL Desktop keeps its own list of previous searches. If you want to reuse a previous search in another database, select the entry in the **Search History** menu list ( ) of the first database from the **Model Manager** window and then activate the other database via the **Database** section on the **Home** toolbar — see Activating a Database. The **Model Manager** window will automatically update with the activated database as the search target while keeping the existing search term and applied filters.

Full Text Search

A fast way for you to find models and files in a Model Manager database is to write individual search words, separated by spaces, in the search input field of the **Open**, **Select File**, **Select Model**, or **Model Manager** window and click the **Search** button (). In this section, you will learn how Model Manager performs a *full text search* to match these search words against fields shown in The Settings Window for models and files.

In this section:

- Combining Full Text Search Words
- Matched Fields

Search words match in a case-insensitive way when searching in Model Manager.

Combining Full Text Search Words

Model Manager combines all search words that you write with `AND`-logic. In practical terms this means that the models and files included in a returned search result are such that all search words matched at least once in one of the searched fields.

An example from the **Application Libraries** window: If you import `busbar.mph` found under **COMSOL Multiphysics** > **Multiphysics** into a database, it may, for example, have the following fields:

- **Title**: Electrical Heating in a Busbar
- **Description**: This example analyzes the resistive heating of a busbar designed to conduct direct current.
- **Tags**: Multiphysics

You can find this model by writing some or all of the search words: `busbar example multiphysics`. The first word, `busbar`, matches on a word in the title and the description, the second word, `example`, matches a word in the description, and the last word, `multiphysics`, matches an assigned tag.

You can use a *wildcard* asterisk character in a search word to match zero or more arbitrary characters when searching the latest versions in a branch or when searching all versions in the database — see Searching Versions. Appending, for example, a

wildcard to a search word will match that word against the beginning of a word or the whole word in the searched text. See also Wildcard Matching.

You can also match on phrases — that is, multiple words in a sequence — by enclosing search words in quotation marks. Searching on `"conduct direct current"` will match in the description in the example, while other permutations of these search words will not match. See also Phrase Matching.

Leading and trailing punctuation markers — for example, dots and hyphens — are ignored when searching words in a field of type Text Field. That is why the trailing dot in `current` can be omitted.

Including a wildcard asterisk character or using phrase quotation marks have no special meaning when Searching in Snapshots and Commits. Model Manager will, however, automatically append an *implicit* wildcard to each search word in those cases.

Matched Fields

The fields that a full text search will match against differ slightly depending on the selected search mode and, in the case of **Latest Versions for Location** (), the selected location type.

- Searching in Branches
- Searching in Snapshots and Commits
- Searching All Versions in the Database

MATCHED FIELDS WHEN SEARCHING IN BRANCHES

Full text search words match the following Model Settings and File Settings when searching the latest versions on a branch:

- The **Title** field of a model and file version.
- The **Description** field of a model and file version.

- The **Filename** field of a model version and all filenames of a file version's file resources as listed in the file version's **Contents** section. The file type extension is considered as a part of the filename.
- The title of assigned **Tags**. Either tags assigned directly to an item or an ancestor to such a tag in The Tag Tree.

Unlike when Searching in Snapshots and Commits, the full text search does not match separately against file type extensions. You can, however, search on such extensions using a File Type filter.

MATCHED FIELDS WHEN SEARCHING IN SNAPSHOTS AND COMMITS

Full text search words match the following Model Settings and File Settings when searching the latest versions with respect to a snapshot or a manually selected commit in a branch:

- The **Title** field of a model and file version.
- The **Description** field of a model and file version.
- The **Filename** field of a model version.
- The file type extensions of a file version's file resources as found in the **Contents** section.

Selecting **Tags** instead of the default **Text** in the list next to the **Search** button () in The Model Manager Window matches search words on the title of assigned tags — either tags assigned directly to an item or an ancestor to such a tag in The Tag Tree.

MATCHED FIELDS WHEN SEARCHING ALL VERSIONS IN THE DATABASE

Full text search words match the following Model Settings and File Settings when searching all versions in a database:

- The **Title** field of a model and file version.
- The **Description** field of a model and file version.
- The **Filename** field of a model version and all filenames of a file version's file resources as listed in the file version's **Contents** section. The file type extension is considered as a part of the filename.

Item and Content Filters

In this section you will learn how to apply the predefined filters available from the **Filter** dialog. To learn how to write your own custom filters, see The Model Manager Search Syntax.

- Field Types
- The Filter Dialog
- Item Filters
- Content Filters
- Applied Filter Pills

Filters match in a case-insensitive way in Model Manager.

Field Types

The Item Filters and Content Filters that are available in the Model Manager search tools can be categorized based on the types of fields that they match on. These *field types* affect, for example, how the search words specified in a Full Text Search or the field values specified in filters are interpreted when searching the database.

- Text Field
- Keyword Field
- Date Field
- Numeric Field
- File Size Field
- Boolean Field
- Selection Field

TEXT FIELD

A filter on a *text field* matches search words in a text. Individual "words" in the text are obtained by splitting on spaces, punctuation markers, and other delimiter characters. Examples of text fields are the title and description of a model or file version.

KEYWORD FIELD

A filter on a *keyword field* matches a search word against a string value. Keyword fields are typically found for short, name-like, search data, such as the filename of a model version.

To include spaces in a keyword field's value, precede each space using a backslash character — see Escaping Reserved Characters.

If you omit the backslash character before a space in the value for a keyword field, the value will be interpreted as two words combined with Boolean `AND`-logic — see Basic Field Expressions. The search word `electrical heating busbar.mph` for a filename field is equivalent to `electrical AND heating AND busbar.mph` — an expression that can never match the filename of a model (a model can have at most one filename). Write `electrical\ heating\ busbar.mph` instead.

DATE FIELD

A filter on a *date field* matches against a date and time value. You can specify the date and time format according to the localization rules for the language set in the COMSOL Desktop or according to the ISO-8601 standard format. An example of a date field is the saved date of a model or file version.

The following are all valid date and time values for a date filter matching August 24, 2021, 2:30 p.m. when running the COMSOL Desktop with language set to `English`:

- `8/24/21`
- `8/24/21, 2:30:00 PM`
- `2021-08-24`
- `2021-08-24T14:30:00`

Dates and times can either be matched exactly or as an interval. The intervals can be open on one or both sides. See also Range Matching. You can also specify intervals from a select set of *date shorthands*. This includes, for example, `TODAY` for the current day and `CURRENTWEEK` for a date interval covering the current week. See Table 3-6 for the complete list.

NUMERIC FIELD

A filter on a *numeric field* matches on a real or complex scalar value. An example of a numeric field is the last computation time of a study, or a numerical setting or parameter value. The implicit unit used for dimensioned scalar values depends on the context.

Numeric fields can either be matched exactly or by specifying intervals for the real and imaginary parts separately. The intervals can be open on one or both sides. See also Range Matching.

FILE SIZE FIELD

A filter on a *file size field* is a variant of a real Numeric Field. It matches on a file size value expressed in bytes. An example is the total size of all Built, Computed, and Plotted Data of a model version.

The field value expression is written as a numerical value and a *byte unit expression* — a unit multiple based on the byte. The supported byte unit expressions are B, KB, MB, GB, and TB. These expressions are defined using the nonstandardized but, in the context of computer memory, often conventional binary memory format:

- B = 1 byte
- KB = 1024 bytes
- MB = 1024^2 bytes
- GB = 1024^3 bytes
- TB = 1024^4 bytes

The space between the real value and the byte unit is optional, but if included, the space itself must be preceded by a backslash character — see Escaping Reserved Characters. You may also omit the byte unit expression altogether, in which case the unit is assumed to be in bytes.

Filters on file sizes are typically most useful as ranges. To help with this, the Model Manager search tools will automatically apply transformation rules to such field expression values: A field expression value that is not written as a range will be automatically converted into a range, while a numerical value inside a range will either be suitably decreased or suitably increased depending on if it is a lower bound or an upper bound. These transformation rules align with the rounding rules applied when file sizes are shown in the Model Manager workspace — rounding rules that are typically also found in the system explorers of various operating systems:

- A file size less than 1 KB is shown as is.

- A file size in the kilobyte range, that is greater or equal to 1 KB but less than 1 MB, are shown in KB rounded up to the nearest integer.
- A file size in the megabyte, gigabyte, or terabyte ranges are shown in MB, GB, or TB respectively, rounded using the round-half-up-algorithm to one decimal.

These rounding rules imply, for example, that file size values between 1025 bytes and 2048 bytes are all shown as 2 KB in the Model Manager workspace. To match on all item versions shown with these sizes, Model Manager will thus automatically transform the filter expression `@fileSize:2KB` to `@fileSize:[1025 TO 2048]`. These rounding rules also guarantee that you will disjointly cover the entire range up to say 4 KB using first `@fileSize:[0 TO 2KB]` and then `@fileSize:[3KB TO 4KB]`, that is as long as you use the same number of decimal places and the same unit for the upper bound of the first interval and the lower bound of the second interval when splitting an interval into two.

To avoid Model Manager applying any transformation rules, write your field expression value using the B byte unit expression or, equivalently, leave out the byte unit expression altogether.

BOOLEAN FIELD

A filter on a *Boolean field* matches on either `true` or `false`. An example is whether or not a 1D or 2D component is axisymmetric.

SELECTION FIELD

A filter on a *selection field* is a specialized filter that involves selecting values from a predetermined set. Examples include Item Save Types of models or the user that saved a model or file version.

The Filter Dialog

You can apply filters to a model and file search from the **Filter** dialog. Click the **Add Filter** button ( ) from the toolbar of the **Open**, **Select File**, **Select Model**, or **Model Manager** windows. Select one of the Item Filters or Content Filters in the menu. The **Filter** dialog is opened with fields and options appropriate for the selected filter.

The **Filter** field shows the name of the selected filter. The other input fields depend on which of the filters you selected.

Select any of the Filter Options under **Options** to quickly modify how the filter's field values will be combined and interpreted. You can see a preview of the corresponding

filter query expression written using The Model Manager Search Syntax under **Filter query preview.**

Spaces entered in an input field for a Text Field in the **Filter** dialog will be automatically replaced by Boolean `OR` operators by default — note the generated filter query expression under **Filter query preview**. This is in contrast to when writing a custom filter query using The Model Manager Search Syntax in which spaces are interpreted as Boolean `AND` if no Boolean operator is explicitly written.

Spaces entered in an input field for a Keyword Field in the **Filter** dialog will be automatically escaped — see Escaping Reserved Characters.

Click the **Customize filter query** button ( ) to replace the filter query with your own customized query.

Click **OK** to apply the field values as a new filter.

FILTER OPTIONS

You can modify how filters are applied in the **Filter** dialog by selecting one of the filter options under **Options**.

Select **Prefix match** to automatically append a wildcard to each search word in a Text Field or at the end of a Keyword Field. See also Wildcard Matching.

You include a wildcard by writing an asterisk character in a search word. The wildcard will match any number of arbitrary characters.

Select **Match all terms** to replace the `OR`-logic used between search words in an input field for a Text Field or selected checkboxes in a Selection Field with `AND`-logic.

Select **Match whole phrase** to require that the search words match a sequence of words (that is, a *phrase*) in a Text Field or Keyword Field. See also Phrase Matching.

You match search words as a phrase by enclosing them in quotation marks.

Select **Negate match** to require that the specified field values do *not* match any models or files in the search result. See also Negated Matching.

Item Filters

Item filters match on field values typically found in the **Version** section and **Tags** section for Model Settings and File Settings.

- Title
- Description
- Tag
- Item Version Type
- Item Save Type
- Saved
- Saved By
- Commit Comment
- Size
- Computed Data
- Filename
- File Type
- Owner

TITLE

The **Title** filter ( ) is a Text Field filter that matches on the title of a model or file version. You may find this filter useful when the searched title is a word commonly found also in descriptions and tags, such that performing a Full Text Search would yield too many results.

DESCRIPTION

The **Description** filter ( ) is a Text Field filter that matches on the description of a model or file. Similar to an Title filter, you may find this useful when the searched description is also a common title or tag.

TAG

The **Tag** filter ( ) is a Selection Field filter that matches on the assigned tags of a model or file with respect to a branch. Select one or more tags in The Tag Tree shown

in the **Filter** dialog. Click **Clear Tags** ( ) to clear all selections — the filter will then match on all items.

	You will notice that the generated filter query under **Filter query preview** does not show the titles of the tags, but rather their unique keys in the database. This is to ensure that you find exactly the items tagged by the selected tag, and not items tagged by a tag that happen to share the same title.

	The **Tag** filter ( ) is not available when Searching All Versions in the Database.

ITEM VERSION TYPE

The **Item Version Type** filter ( ) is a Selection Field filter on the Item Version Types of a model or file version. Select from **Model**, **Application**, **Physics**, **File**, and **Fileset**.

ITEM SAVE TYPE

The **Item Save Type** filter ( ) is a Selection Field filter on the Item Save Types of a model or file (the latter only has regular as available item save type) that a version belongs to. Select from **Regular** and **Draft**.

SAVED

The **Saved** filter ( ) is a Date Field filter for when a version was saved. Select one of the date shorthands in the **Range** list to use a preset date interval. Select **Manual** and write a start date in the **From** field and an end date in the **To** field. Leave one or both input fields empty to not set a corresponding bound.

SAVED BY

The **Saved By** filter ( ) is a Selection Field filter on the user that saved a version. Write the name or display name of a user in the **Name** field. Spaces in names are automatically escaped with backslashes.

COMMIT COMMENT

The **Commit Comment** filter ( ) is a Text Field filter that matches on the commit comment written when saving a model or file version.

SIZE

The **Size** filter ( ) is a File Size Field filter on the estimated disk space usage of a version when stored on the file system.

COMPUTED DATA

The **Computed Data** filter ( ) is a File Size Field filter on the estimated disk space usage of the built, computed, and plotted data of a model version when stored on the file system.

FILENAME

The **Filename** filter ( ) is a Keyword Field filter on the filename of a model version or the file resources of a file version. The filename includes the file type extension as a suffix.

FILE TYPE

The **File Type** filter ( ) is a combined Selection Field and Keyword Field filter on the file type extension of a file. Select a file type extension from the **Predefined** field, or manually type file type extensions, separated by spaces, in the **Manual** keyword field.

Unlike other Keyword Field filters in the **Filter** dialog, in which spaces are escaped with backslash, spaces between file extensions will instead be automatically replaced with Boolean `OR` operators.

OWNER

The **Owner** filter ( ) is a Selection Field filter on the current owner of a model or file that a version belongs to. Write the name or display name of a user in the **Name** field. Spaces in names are automatically escaped with backslashes.

Content Filters

Content filters match on values in The Contents Section of Model Settings — that is, on node properties, parameters, features, and other settings in the model tree of a model.

- Parameter
- Setting
- Part
- Space Dimension
- Physics
- Study Step
- Node Type
- Label
- Name
- Tag
- Created
- Author
- Version
- Comment
- Last Modified
- Last Modified By

Searching Nodes and Settings in the Model Tree

PARAMETER

The **Parameter** filter () is a combined Keyword Field, Text Field, and Numeric Field filter on parameters in a **Parameters** node. You can create a filter by combining:

- The parameter name as a Keyword Field in the **Name** field.
- The parameter description as a Text Field in the **Description** field.
- The parameter expression, including units, as a Keyword Field in the **Value** field.
- The parameter value, excluding units, as a real scalar Numeric Field in the **From** and **To** range fields. Leave one or both input fields empty to not set a corresponding bound.

The parameter value uses the unit system as set in the model when it was saved to the database.

Parameters in the *COMSOL Multiphysics Reference Manual*

SETTING

The **Setting** filter ( ) is a combined Keyword Field and Text Field filter on general settings found in the **Settings** window for a node. You can create a filter by combining:

- The name of a setting as a Keyword Field in the **Name** field. This is an identifier of the setting that is not visible in the Model Builder, Application Builder, or Physics Builder **Settings** windows. You can find the name of a particular setting in the **Name** column in the **Details** dialog for a node in The Contents Section.
- The description of a setting as a Text Field in the **Description** field.
- The value of a setting as a Keyword Field in the **Value** field.
- The value of a setting when accessed via the COMSOL API as a Keyword Field in the **API value** field.

!	Not all node settings can be searched in the database. You can see which settings are available by, for example, finding a model that has the node type you want to search, select the node in The Contents Section, click the **Details** button ( ), and examine the rows in the **Settings** table in the opened **Details** dialog.

PART

The **Part** filter ( ) is a Selection Field filter on model parts found inside the model tree that may be reused by other models as auxiliary data. The only supported part is **Geometry**.

	Geometry Parts Saved in Databases

SPACE DIMENSION

The **Space Dimension** filter ( ) is a Selection Field filter on the space dimension of a **Component** node. Select from **3D**, **Axial Symmetry (2D)**, **2D**, **Axial Symmetry (1D)**, **1D**, and **0D**.

	The Component Node in the *COMSOL Multiphysics Reference Manual*

PHYSICS

The **Physics** filter ( ) is a Selection Field filter on physics interfaces used by a model. You select physics interfaces you want to filter on from a tree. Select a space dimension in the **Show available physics for** field to filter the tree on physics interfaces available for that dimension. You can also filter the tree by writing the name of a physics interface in the search field above the tree.

	The Physics Nodes in the *COMSOL Multiphysics Reference Manual*

STUDY STEP

The **Study Step** filter ( ) is a Selection Field filter on study steps used by a model. Select the study steps you want to filter on from the shown list.

	Study and Study Step Types in the *COMSOL Multiphysics Reference Manual*

NODE TYPE

The **Node Type** filter ( ) is a combined Text Field and Keyword Field filter for finding models with nodes of a particular type. You can create a filter be combining:

- The node type as shown in the user interface as a Text Field in the **Type** field.
- The node type as used in the COMSOL API as a Keyword Field in the **API type** field.
- The node class as used in the COMSOL API as a Keyword Field in the **API class** field.

	Click the **Model Tree Node Text** button ( ) and select **Type** to see a node's type in the Model Builder tree.

You typically include a filter on the **API type** or **API class** fields to narrow the search when the **Type** field alone matches too wide.

NODE PROPERTIES

You can filter on node properties as shown in the **Properties** window for a node.

Label

The **Label** filter (abc) is a Text Field filter on a node's label.

Name

The **Name** filter (ABC) is a Keyword Field filter on a node's name.

Tag

The **Tag** filter (ABC) is a Keyword Field filter a node's tag.

Created

The **Created** filter ( ) is a Date Field filter for when the node was created.

Author

The **Author** filter ( ) is a Keyword Field filter on the node's **Author** field in the **Properties** window.

The value in a node's **Author** field is written as a general string value in the **Properties** window and need not correspond to any Users in a database.

Version

The **Version** filter (ABC) is a Keyword Field filter on the node's **Version** field in the **Properties** window.

The value in a node's **Version** field is written as a general string in the **Properties** window and does not correspond to any particular model version in a database.

Comment

The **Comment** filter ( ) is a Text Field filter on the node's **Comments** field in the **Properties** window.

Last Modified

The **Last Modified** filter ( ) is a Date Field filter for when the node was last modified. Except for the root node, this field is only available from the model object via the COMSOL API. Only a subset of all node types automatically update this value.

Last Modified By

The **Last Modified By** filter ( ) is a Keyword Field filter on the last modified by value accessible from the model object via the COMSOL API. Only a subset of all node types automatically update this value.

Applied Filter Pills

The currently applied filters in the **Open**, **Select File**, **Select Model**, and **Model Manager** windows are shown as *filter pills* above the search result.The pill's text is a short summary of the filter. Click a pill and select **Edit** () to edit an existing filter via The Filter Dialog. Select **Remove** () to remove a filter.

The filters are combined using `AND`-logic when searching, meaning that each item version found in the search result satisfy the conditions of all filters. Click a pill and select **Disable** () to temporarily disable the application of that filter in the search. Editing a disabled filter via The Filter Dialog will not automatically enable it. Click a disabled filter pill and select **Enable** () to enable it again.

Selecting **All Versions in Database** () when an unsupported filter for that search mode has already been applied will automatically disable the filter and show a warning triangle () on the pill. The same will happen if connecting to an older Model Manager server database that does not support the corresponding filter expression. It will not be possible to edit or enable the filter unless returning to a search target for which the filter is supported.

The Model Manager Search Syntax

The filter functionality in Model Manager is based on a tailor-made search syntax adapted for searching deep within a COMSOL Multiphysics simulation model. You may have seen examples of this syntax already in the **Filter query preview** field in The Filter Dialog. The syntax enables you to write filter queries that find models whose node properties, parameters, features, and other settings satisfy arbitrarily complex constraints. As an example, you can write a custom filter query that matches all models that have an axisymmetric 2D component, for which the component uses a particular physics interface and a particular material.

In this section, you will learn how you can formulate such custom filter queries. Via examples of gradually increasing complexity, you will see how to write simple expressions for filters on single fields, how you can combine such field expressions using Boolean logic, and finally how you can nest field expressions based on the nested node hierarchy in the model tree. The section ends with a complete listing of all available field expressions in the Model Manager search syntax.

- Basic Field Expressions
- Combining Expressions
- Searching Nodes and Settings in the Model Tree
- Search Syntax Completion
- Search Syntax Catalog

The Model Manager search syntax is only available when searching the latest versions in a branch or when searching all versions in the database. It is not available when searching the latest versions with respect to a snapshot or a commit. See also Searching Versions.

The examples in this section will generally refer to Models. You can write analogous filter queries for Files using the item field expressions in the Search Syntax Catalog.

Basic Field Expressions

A filter consists of one more *field expressions* combined with Boolean operators — for example, `AND`, `OR`, and `NOT` — and other grouping operators. Each field expression specifies which *field* is searched and the *field value* being searched on. The field also has a particular type which dictates how the field value will be interpreted by Model Manager — see Field Types.

You write a field expression using an `@`-notation of the general form:

```
@<field-name>:<field-value>
```

with `<field-name>` equal to the name of one of the available fields in Table 3-1, and `<field-value>` equal to the value being filtered on. Write, for example,

```
@title:busbar
```

to find all versions whose title contain the word `busbar`. To match on several search words, enclose the words with parentheses. Write

```
@title:(electrical busbar)
```

to find all versions whose title contain both the words `electrical` and `busbar`.

A space between two search words is automatically interpreted as a Boolean `AND` in the Model Manager search syntax. The previous expression is thus equivalent to:

```
@title:(electrical AND busbar)
```

Write

```
@title:(electrical OR busbar)
```

if you want to find all versions whose title contain either `electrical` *or* `busbar`.

You may have noticed that spaces in the input field for a Text Field in The Filter Dialog are automatically replaced by a Boolean `OR` in the corresponding **Filter query preview**. Select **Match all terms** under **Options** to change this to `AND` instead.

Field expressions can be written for other Field Types than a simple Text Field. Write

```
@saved:9/17/21
```

to find all versions saved on September 17, 2021 using a Date Field.

You can write the field name in a case-insensitive way. Thus `@saved:9/17/21` works fine.

You can enter a custom filter query either directly in the search field in the **Open**, **Select File**, **Select Model**, and **Model Manager** windows, or apply it as a separate filter in Applied Filter Pills by first clicking the **Customize filter query** button ( ) from The Filter Dialog.

If you want to combine full text search with a custom filter query, write the former first. The following is valid:

```
electrical busbar @description:example
```

and matches on all versions with a title, description, filename, or assigned tags containing the words `electrical` and `busbar`, and that has a description containing the word `example`. The following is not valid:

```
electrical @description:example busbar
```

and results in an error message.

Write all plain search words first, optionally followed by a custom filter query.

CONTROLLING PRECEDENCE USING PARENTHESES

A Boolean `AND` takes precedence over a Boolean `OR` in the Model Manager search syntax. The filter

```
@title:(electrical AND busbar OR tuning AND fork)
```

matches versions whose title either contains both `electrical` and `busbar`, or whose title contain both `tuning` and `fork`. You can override this operator precedence with parentheses. Write

```
@title:(electrical AND (busbar OR tuning AND fork))
```

to match all versions whose title contain `electrical`, and either the single word `busbar` or the two words `tuning` and `fork`.

WILDCARD MATCHING

You can use an asterisk character as a *wildcard* symbol that matches on zero or more arbitrary characters. The filter

```
@title:electric*
```

matches on all versions whose title begin with electric.

You can match on versions that have *any* value set for a field by using a single wildcard symbol. Write

```
@description:*
```

to find all versions with a nonempty description. This can also be written using the special `ANY` symbol:

```
@description:ANY
```

which may feel more intuitive.

You can use a wildcard symbol anywhere in a search word. Placing it at the start (that is, a *postfix* search) may, however, result in slow query times. The exception is when the field value only contains a single wildcard symbol, and nothing else.

Wildcard matching on search words that include punctuation markers — for example, periods, commas, colons, and hyphens — may lead to surprising results when used in a full text search or in a Text Field filter due to how the search splits text into word tokens. You are recommended to avoid using wildcards for such search words. A Keyword Field filter does not have this limitation.

PHRASE MATCHING

You can match on phrases — that is, multiple words in a sequence — by enclosing them in quotation marks. Write

```
@title:"electrical heating"
```

to match on versions whose title contains `electrical` followed by `heating`. You can also combine phrase search with ordinary search. Write

```
@title:("electrical heating" busbar)
```

to match on versions whose title contains `electrical` followed by `heating`, as well as the word `busbar`, for example, `Electrical Heating in a Busbar`.

Wildcard Matching inside a phrase is not supported.

NEGATED MATCHING

You can reverse the match logic using the special `NOT` symbol. Write

```
@title:(NOT busbar)
```

to find all versions whose title does *not* contain the word `busbar`. The `NOT` symbol takes precedence over both `AND` and `OR`, although you can override this precedence with parentheses. Write

```
@title:(NOT (electrical busbar))
```

to find all versions whose title does not contain the words `electrical` and `busbar`.

You can exclude the parentheses if the `NOT` is followed by a single word. Thus `@title:NOT busbar` is equivalent to the first example.

RANGE MATCHING

A filter on a Date Field, a Numeric Field, and a File Size Field can be written as inclusive ranges. Write

```
@saved:[9/1/21 TO 9/30/21]
```

to find all versions saved for the month of September in 2021. Use a wildcard symbol for unbounded ranges. The filter

```
@saved:[* TO 8/31/21]
```

matches on all versions that have not been saved after August 2021. You may also leave out the wildcard symbol altogether as long as you include a space before the `TO` symbol. The following is equivalent to the previous expression:

```
@saved:[ TO 8/31/21]
```

Ranges are also supported for a Text Field or Keyword Field filter and corresponds to matching in between two boundary words when ordering search words lexicographically. The usefulness of this, however, is rather limited.

ESCAPING RESERVED CHARACTERS

Some characters serve special purposes in the Model Manager search syntax and are therefore considered as *reserved characters*. If you want to search on words that contain such characters, precede them by a backslash. Write

```
@tag:(\[In Progress\])
```

to match all items assigned the tag with title `[In Progress]`. The enclosing parentheses are necessary as there are two search words, `[In` and `Progress]`.

The ten reserved characters are:

```
{ } ( ) [ ] " : \ SPACE
```

The last one is a common pitfall when writing a filter on a Keyword Field that matches a string containing a space character. Write

```
@filename:electrical\ heating\ busbar.mph
```

to match versions with filename `electrical heating busbar.mph`. Parentheses are not necessary here because the field value is considered as one search word as the two space characters have been escaped.

Combining Expressions

You can combine any number of field expressions to form more complicated filter queries. Write

```
@title:busbar @description:example
```

to find all versions whose title contain the word `busbar` and whose description contain the word `example`. As for search words in the field value of a single field expression, a space character between two field expressions is interpreted as a Boolean `AND`. Write

```
@title:busbar OR @description:example
```

to combine the two field expressions with a Boolean `OR`.

You can also use parentheses to control operator precedence. Write

```
@itemSaveType:draft AND (@owner:Alice OR @savedBy:Alice)
```

to find all drafts that are either owned *or* were saved by user Alice.

Searching Nodes and Settings in the Model Tree

You can find models by matching on their node properties, features, and settings in the model tree. You use a similar search syntax for field expressions involving these nodes as when searching on general item fields — see Basic Field Expressions.

In this section you will learn how to write basic filter queries that match on models with a particular node or setting, as well as advanced filter queries that match on models in which a particular node or setting is nested within other nodes.

Matching the model tree contents of models is only available when searching the latest versions on a branch and with **Item fields and content** configured — see Searching in Branches.

BASIC NODE FIELD EXPRESSIONS

You write a basic *node field expression* filter using an `@`-notation of the general form:

```
@node{@<field-name>:<field-value>}
```

with `<field-name>` equal to the name of one of the available fields in Table 3-5 and `<field-value>` equal to the value being filtered on. Write, for example,

```
@node{@type:rotor}
```

to find all models with nodes of a type that contains the word `rotor`. With this, you can, for example, find models that have a Frozen Rotor study step, or models that have a Beam Rotor interface. Write

```
@node{@type:(beam AND rotor)}
```

to narrow the search to the latter models.

You can leave out the Boolean `AND` as Model Manager automatically adds this whenever there is a space between two search words or two field expressions.

You can match a node on several properties, features, and settings by combining several expressions within `@node{…}`. Write

```
@node{@type:(beam rotor) @comment:"Use axial vibration"}
```

to find a model with a Beam Rotor interface for which a certain comment has been written in the **Comments** field in the **Properties** window for the node — see also Phrase Matching.

A node field expression can also be combined with item field expressions, as well as with other node field expressions. Write

```
@savedBy:Alice @node{@type:(beam rotor)}
```

to find models with a Beam Rotor interface that were saved by user Alice.

USING NAMED NODES

Oftentimes you want to find models with a node of a particular type. One challenge that you then face is that the **Type** field for a Node Type may match wider than you perhaps anticipated. Writing

```
@node{@type:(time dependent)}
```

will match on Time Dependent study steps. But it will also match on Time-Dependent Solvers, which may not be what you wanted. Write

```
@node{@apiClass:StudyFeature @type:(time dependent)}
```

to only match on the study step.

You can find the **API class**, as well as all other node fields, of a particular node in the **Node** table in the **Details** dialog opened from The Contents Section. These often have a technical name that may be challenging to remember. To help you with this, Model Manager comes with a set of predefined *named nodes* that work as aliases. You will match on only a Time Dependent study step by writing

```
@studyStep{@type:(time dependent)}
```

A complete listing of all named nodes is given in Table 3-4.

You use an **API type** field expression when a named node is not enough to distinguish types. Writing

```
@physics{@type:(electric currents)}
```

will match on models with an Electric Currents interface. But it will also match on models with an Electric Currents in Layered Shells interface. Write

```
@physics{@apiType:ConductiveMedia}
```

to only match the former physics interface.

BASIC SETTING FIELD EXPRESSIONS

You write a basic *setting field expression* filter using an `@`-notation of the general form:

```
@setting{@<field-name>:<field-value>}
```

with `<field-name>` equal to the name of one of the available fields in Table 3-3 and `<field-value>` equal to the value being filtered on. As for node field expressions, you can include several expressions within `@setting{…}`. Write, for example,

```
@setting{@description:Length @value:9\[cm\]}
```

to find all models with a setting **Length** having value `9[cm]` — see also Escaping Reserved Characters. When the setting is a scalar, you can also match on a range of values:

```
@setting{@description:Length @scalarReal:[0.05 TO 0.15]}
```

The scalar length values are written in the unit system of the model, unlike the textual length value in the previous example that matched on the exact string including the unit name.

You can find the available settings of a particular node under **Setting** in the **Details** dialog opened from The Contents Section. You may find it useful, for example, to include a filter on the **Name** setting field whenever the **Description** setting field matches too wide.

NESTED NODE AND SETTING MATCHING

You can write custom filter queries that match on settings for a particular node by nesting `@setting{…}` within `@node{…}`, or within any of the names nodes. Write

```
@parameters{@setting{@description:Length @scalarReal:0.09}}
```

to match a parameter setting on a **Parameters** node.

There is no limit on the number of setting field expressions you can include. Write, for example,

```
@parameters{@created:[9/1/21 TO 9/31/21]
@setting{@description:Length @scalarReal:0.09}
@setting{@description:Width @scalarReal:0.05}}
```

to find models with a **Parameters** node created in September 2021, such that the node has a **Length** parameter with value `9[cm]` and a **Width** parameter with value `5[cm]`.

You can also nest two or more node field expressions inside each other. Write, for example,

```
@component{@spaceDimension:2 @physics{@apiType:ConductiveMedia}}
```

to find all models with a **2D** component that contains an Electric Currents interface.

The Model Manager search syntax supports nesting node field expressions five levels deep in the model tree.

Search Syntax Completion

You can press Ctrl+Space when the cursor is placed in the search field in the **Open**, **Select File**, **Select Model**, and **Model Manager** windows to get completion assistance for the Model Manager search syntax. The opened completion menu contains one or more of the following submenus:

- **Field expressions**. The names of item and content filters for use in field expressions.
- **Node and setting expressions**. The names of nodes and settings for use in nested content filters.
- **Field values**. Available field values to filter on for a specific field expression.
- **Symbols**. The symbols used for range expressions, subexpressions, or boolean expressions.

The available submenus and their options depend on the text preceding the cursor location so that a valid syntax expression can still be formulated upon completion. While the menu is still open, you can continue to type in the search field to further narrow down the suggested completions. Once you select something in the menu, the current text in the search field will be adjusted accordingly.

While Model Manager will help you with completing the beginning of a node or setting expression, including appending the opening curly brace symbol ({), you must also remember to end the expression with a closing curly brace symbol (}).

If some parts of the text preceding the cursor location contain syntax errors, Model Manager will still attempt to complete a portion of the text “closest” to the cursor into a valid expression.

Many completion options support matching on both the actual value to be completed and the option's accompanying descriptive text while continuing to type with the menu open. This includes, for example, the **Field expressions** submenu for which you can match on either the field identifier or one of the words in the field description.

You can write `spaceDim` and press Ctrl+Space, for example, to complete the expression used to filter on all models with components having a specific space dimension:

1. Select the **Field expressions > @node{@spaceDimension:** option from the opened menu.
2. Write an integer space dimension and a closing curly brace, say `@node{@spaceDimension:2}`.
3. Press Enter to find all models with the sought-after space dimension.

Or, you can write `@tag:` and press Ctrl+Space to find models and data files by their tag assignments. Continue to type in the search field to narrow down to a specific tag title under **Field values** in the opened menu.

The `@tag:<field-value>` filter matches on the title of assigned tags as a Text Field filter. Use `@tagKey:<field-value>` if you have created multiple tags in the database having the same title and you only want results for one of them.

Completion of field values is only supported by a subset of all item and content filters. This includes filters involving tag assignments or users, file type extension filters, Selection Field filters of built-in type, Boolean Field filters, and date shorthands of Date Field filters.

Search Syntax Catalog

In this section, you will find a summary of all field expressions and symbols available in the Model Manager search syntax:

- Table 3-1 contains item field expressions that match on basic fields of models and files.
- Table 3-2 contains node field expressions that match on node types and properties.
- Table 3-3 contains setting field expressions that match on node settings.

- Table 3-4 contains the various expressions used to write plain and nested node and setting filters.
- Table 3-5 contains symbols available in the search syntax.

If you are uncertain of what field values are available for a Selection Field with built-in values, write any character and press Enter. Model Manager will show an error dialog containing supported values.

TABLE 3-1: FIELD EXPRESSIONS FOR ITEM FIELDS.

SYNTAX	TYPE	DESCRIPTION
@branch:...	Selection	The name of the branch that an item version is saved to. Escape spaces or other reserved characters in names with backslash.
@commitComment:	Text	The commit comment written when the item version was saved.
@computedData:...	File Size	The estimated disk space usage of all built, computed, and plotted data of a model version when stored on the file system.
@description:…	Text	The description of an item version.
@filename:…	Keyword	The filename used when exporting an item version to the file system.
@fileType:…	Keyword	The file type extensions of the file resources belonging to a file version.
@itemKey:...	Selection	The unique key of the item that an item version belongs to.
@itemSaveType:…	Selection	The item save type of the item that an item version belongs to.
@itemType:…	Selection	The item type of the item that an item version belongs to.
@itemVersionKey:...	Selection	The unique key of an item version.
@itemVersionType:...	Selection	The item version type of an item version.
@lastModified:…	Date	The instant in time when an item was last modified with respect to a branch. May be used in place of @saved:...
@lastModifiedBy:…	Selection	The name or display name of the user that last modified an item with respect to a branch. Escape spaces or other reserved characters in names with backslash. May be used in place of @savedBy:...

TABLE 3-1: FIELD EXPRESSIONS FOR ITEM FIELDS.

SYNTAX	TYPE	DESCRIPTION
@originItemKey:...	Selection	The unique key of the origin item to the item that an item version belongs to.
@owner:…	Selection	The name or display name of the user that owns the item. Escape spaces or other reserved characters in names with backslash.
@part:...	Selection	A reusable model part in a model version.
@repository:...	Selection	The name of the repository that an item version is saved to. Escape spaces or other reserved characters in names with backslash.
@saved:...	Date	The instant in time when an item version was saved.
@savedBy:...	Selection	The name or display name of the user that saved an item version. Escape spaces or other reserved characters in names with backslash.
@size:...	File size	The estimated disk space usage of an item version when stored on the file system.
@tag:…	Text	The titles of tags assigned to an item with respect to a branch.
@tagKey:…	Selection	The unique keys of tags assigned to an item with respect to a branch.
@title:…	Text	The title of an item version.

TABLE 3-2: FIELD EXPRESSIONS FOR NODE FIELDS.

SYNTAX	TYPE	DESCRIPTION
@apiClass:…	Keyword	The class of a node in the COMSOL API.
@apiType:…	Keyword	The node type as represented in the COMSOL API.
@author:…	Text	The author property of a node.
@axisymmetric:…	Boolean	The axisymmetric property of a **Component** or **Geometry** node.
@comment:…	Text	The comment property of a node.
@created:…	Date	The instant in time when a node was created.
@label:…	Text	The label of a node.
@lastComputationDate:...	Date	The instant in time when a **Study** node was last solved.
@lastComputationTime:…	Numeric	The computation time in seconds when a **Study** node was last solved.

TABLE 3-2: FIELD EXPRESSIONS FOR NODE FIELDS.

SYNTAX	TYPE	DESCRIPTION
@lastComputationVersion:...	Keyword	The COMSOL version that a **Study** node was last solved in.
@lastModified:…	Date	The instant in time when a node was last modified.
@lastModifiedBy:…	Text	The last modified by property of a node.
@name:…	Keyword	The name of a node.
@spaceDimension:…	Numeric	The spatial dimension of a **Component** or **Geometry** node.
@tag:…	Keyword	The tag of a node.
@type:…	Text	The node type as shown in the user interface.
@version:…	Text	The version property of a node.

TABLE 3-3: FIELD EXPRESSIONS FOR SETTING FIELDS.

SYNTAX	TYPE	DESCRIPTION
@apiValue:…	Keyword	The value of a setting as returned in the COMSOL API.
@description:…	Text	The description of a setting.
@name:…	Keyword	The identifier name of a setting.
@scalarImag:…	Numeric	The imaginary part of a scalar setting value.
@scalarReal:…	Numeric	The real part of a scalar setting value.
@value:…	Keyword	The value of a setting.

TABLE 3-4: NODE AND SETTING EXPRESSIONS.

SYNTAX	API CLASS	DESCRIPTION
@commonDefinition{…}	CommonFeature	A common definition node.
@component{…}	ModelNode	A **Component** node.
@geometry{…}	GeomSequence	A **Geometry** node.
@material{…}	Material	A **Material** node.
@mesh{…}	MeshSequence	A **Mesh** node.
@model{…}	Model	The root node of a model.
@multiphysicsCoupling{…}	MultiphysicsCoupling	A **Multiphysics** coupling node.
@node{…}	N/A	An arbitrary node.
@nodeGroup{…}	NodeGroup	A **Node Group**.
@parameters{…}	ModelParamGroup	A **Parameters** node.

TABLE 3-4: NODE AND SETTING EXPRESSIONS.

SYNTAX	API CLASS	DESCRIPTION
@physics{…}	Physics	A **Physics** node.
@results{…}	Results	The **Results** node.
@setting{…}	N/A	A node setting.
@study{…}	Study	A **Study** node.
@studyStep{…}	StudyFeature	A **Study Step** node.

TABLE 3-5: SYMBOLS IN THE MODEL MANAGER SEARCH SYNTAX.

SYNTAX	DESCRIPTION
*	A wildcard in a search word.
AND	Combines two field values or field expressions with a Boolean AND.
OR	Combines two field values or field expressions with a Boolean OR.
NOT	Negates a field value or field expression.
ANY	An alias for a field value containing a single wildcard.
(…)	Enclosing field values or field expressions.
{…}	Enclosing expressions inside a node or setting filter.
[… TO …]	A field range value.
"…"	Quotation marks enclosing a sequence of search words as a phrase.
\<c>	Backslash escaping one of the ten reserved search syntax characters.

TABLE 3-6: DATE SHORTHANDS IN THE MODEL MANAGER SEARCH SYNTAX.

SYNTAX	DESCRIPTION
TODAY	Shorthand for an interval covering the current day.
YESTERDAY	Shorthand for an interval covering yesterday.
CURRENTWEEK	Shorthand for an interval covering the current week.
CURRENTMONTH	Shorthand for an interval covering the current month.
CURRENTYEAR	Shorthand for an interval covering the current year.

4

Advanced Version Control

In this chapter, you will learn how you can experiment with a collection of models and data files in a Model Manager database by creating an alternative, perhaps private, commit history. If your experiments are successful, you can then incorporate the updated models and files as new versions in the main commit history. You will also learn how to undo changes in your commit history.

In this chapter:

Branching

In this section you will see how you can create an alternative commit history in a repository by creating a new branch from an existing branch, also known as *branching off* from the existing branch. You may, for example, find branching useful when:

- You and your team of coworkers save your ongoing work on models and data files on a branch private to the team, and then publish the finished work on a branch shared with your organization. Perhaps there are multiple teams publishing to this latter branch.
- You start a project involving one or more models and data files already present in the database, but you are uncertain what the end result is going to be. Perhaps the modifications you plan to make will result in new versions of the existing models, or perhaps you want to create entirely new models. Perhaps you decide to discard the modifications altogether. By branching, you can postpone these decisions until the project is finished.
- You want to experiment with a single model in a way that is hidden from other users. You therefore forgo the more lightweight, and recommended, way of Saving Drafts of Models — opting to create a branch that uses a **Private** permission template,

Prefer Repositories over Branches if you want to create independent silos of models and data files.

In this section:

The Branch as a Sequence of Commits

An initial branch is automatically created when you add a new repository in a database. This is especially true for the initial repository automatically added when you create a new database. Any set of changes to items — that is, model, files, and tags — are saved in commits on this branch. See Basic Version Control for more details.

You may think of a branch as a sequence, or history, of such commits. Each commit identifies a collection of versions that were the most recently saved, or latest, at the

time of the commit. Any particular version could have been saved in that particular commit, or in a previous commit. Each commit also identifies the tag assignments to items at the time of the commit. By comparing the collections of item versions, as well as the assigned tags of items, present in two different commits, Model Manager can infer all item changes done in-between the first commit and the second commit. See Figure 2-2 for a schematic representation of a branch containing three commits.

Use The Select Location Dialog, or click branch nodes ( ) in The Databases Tree, to switch between branches in The Model Manager Workspace Windows.

Creating a New Branch

You can select any commit on a branch to create a new branch from that *source commit*. This starts a new sequence of commits that runs in parallel with the first sequence. When the branch is created, the collections of versions and assigned tags will be identical in the source commit and on the new branch. But as soon as you start saving new versions and reassigning tags, the branches will diverge. See Figure 4-1, which is a continuation of Figure 2-2, for a schematic representation.

The Branch as a Sequence of Commits

- Prefer saving a draft over creating a new branch if all you want to do is experiment with a model, without necessarily affecting its version history. See Saving Drafts of Models.
- Prefer using **Save as New** ( ) if you want to create a copy of a model. Changes to the copy can later be merged to the original model via the **Comparison Result** window. See Saving Models to Databases and Comparing a Version With the Opened Model in the COMSOL Desktop.

A good mental picture is to think of a tree in which the initial branch is the trunk of the tree, with an initial commit saved at the base of the trunk (at the ground), and successive commits stacked on top of each other. Other branches created off the main

branch correspond to tree branches shooting out from the trunk. A repository can be thought of as the tree itself.

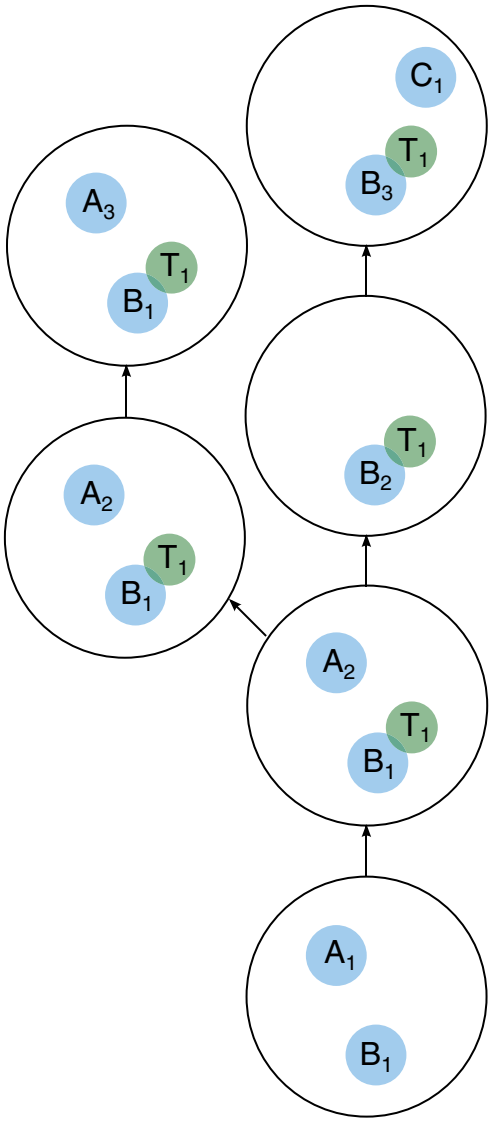


Figure 4-1: A schematic representation of a repository containing two branches. A second branch has been created off the main branch's second commit. In the third commit on the main branch, the model A was deleted. On the new branch's second commit, a new version of model A was instead saved. Browsing the latest versions on the main branch will return model versions B_3 and C_1. Browsing on the new branch will return A_3 and B_1.

To create a branch from a particular source commit, do one of the following:

- Select a branch node ( ) in The Databases Tree and click the **Branch** button ( ) in the **Repository** section of the **Database** toolbar. The source commit is the latest commit on the branch.
- Select a snapshot node ( ) in The Databases Tree and click the **Branch** button ( ) in the **Repository** section of the **Database** toolbar. The source commit is the commit that the snapshot references.
- Select a commit table row in The Commits Window and click the **Branch** button ( ) in the window's toolbar.

In all cases, The Create Branch Dialog is opened.

Creating a new branch is a fairly cheap operation in a Model Manager database in terms of actual data storage. No data is copied except for the small amount of metadata necessary for determining which items are initially present on the branch. There is, however, a cost in terms of disk space usage if a search index is created for the branch — see also Searching in Branches.

PARTIAL BRANCHES

You can create a *partial branch* that contains a subset of all items that were present in the source commit. Select models and files in The Databases Tree or The Model Manager Window to include those items in the new branch. Select tags to include all items that are tagged by the selected tag. All tag assignments present in the source commit will be mirrored on the new branch as well.

THE CREATE BRANCH DIALOG

You create the new branch from the **Create Branch** dialog. The **Database** field shows the database in which the branch is created, and the **Repository** field shows the repository that the source commit and new branch both belong to.

1. Write the name of the new branch in the **Name** field.
2. Write an optional comment for the initial commit that will be made for the new branch in the **Comments** field.
3. Select **Item fields and content** in the **Search** list for complete search and filter support on the new branch; otherwise, select **Only item fields** — see also Searching in Branches.

4 In the **Selection** list:

- Select **All** to include all items from the source commit.
- Select **Current selection** to only include the items whose versions were selected when the dialog was opened. The selected item versions are displayed in a table under the **Selection** field. Select a table row and click the **Exclude** button () to exclude the item from the new branch.
- Select **Empty** to create a branch that does not include any initial items at all.

5 You can set up permissions for the new branch in the **Permissions** field. This field is only shown if connected to a server database via a Model Manager server. See Granting Permissions.

6 Click **OK** to create the new branch in the database.

The created branch appears as a new child node to the **Branches** node in The Databases Tree.

An empty branch is, for example, useful if you only intend to create new models on the branch. You can later merge these models to the source branch.

An initial commit is always made on a new branch. If you open the Commit Settings for this commit you will see that the **Changes** table is, however, empty — no new versions or tag assignments are saved in the database in that initial commit.

Merging

After you have created a new branch and made changes to items on that branch, you will eventually want to transfer these changes back to its parent branch. In this section, you will learn how you can *merge* such changes to a branch.

- Merging Changes to a Target Branch
- The Merge Window

Merging Changes to a Target Branch

You merge item changes to a *target branch* by first selecting a source commit on a *source branch*. Changes made on the source branch up to and including the source commit will be made available for merging into the target branch. The merge itself will create a new commit on the target branch that includes all item changes you decided to merge.

To make two branches "equal" to each other in terms of their latest item versions, you first need to merge all changes from the latest commit on the first branch to the second branch, and then repeat this in the opposite direction.

To merge from a particular source commit, do one of the following:

- Select a branch node ( ) in The Databases Tree and click the **Merge** button ( ) in the **Repository** section of the **Database** toolbar. The source commit is the latest commit on the branch.
- Select a snapshot node ( ) in The Databases Tree and click the **Merge** button ( ) in the **Repository** section of the **Database** toolbar. The source commit is the commit that the snapshot references.
- Select a commit table row in The Commits Window and click the **Merge** button ( ) in the window's toolbar.

In all cases, The Merge Window is opened with a suggested target branch.

The Merge Window

You use the **Merge** window to select and merge item changes made on a source branch, up to a particular source commit, to a target branch.

Merging Changes to a Target Branch

The **Source** field shows the location corresponding to the source commit for which the window was opened. Click the link button to select a new branch or snapshot as source in The Select Location Dialog. The **Target** field shows the location corresponding to the target branch for the merge. Click the link button to select another branch.

The window contains a table with item changes made on the source branch that are not present in the target branch. The table columns are:

- The type column — the type of the changed item represented by an icon.
- The **Source Change** column — a description of the item change.
- The **Conflicting Target Changes** column — one or more changes on the target branch that are incompatible, or *in conflict*, with the source change.
- The **Selection** column — an icon representing whether to include the source change ( ), ignore the source change ( ), or if there is a conflict ( ).

Click the **Merge Changes** button ( ) to open The Merge Dialog after you have decided which changes to merge and resolved any merge conflicts.

Resolving Merge Conflicts

THE MERGE WINDOW TOOLBAR

The toolbar in the **Merge** window contains the following toolbar buttons:

- Click the **Refresh** button ( ) to refresh the table in case any new commits have been saved on the target branch.
- Click the **Take Source** button ( ) to include a source change in the merge. This is the default choice.

- Click the **Keep Target** button ( ) to ignore a source change, keeping the target as is.
- Click the **Merge Changes** button ( ) to open The Merge Dialog.

RESOLVING MERGE CONFLICTS

When you work with a collection of items with versions on multiple branches, you will inevitably encounter conflicting changes when merging from a source branch to a target branch. Such merge conflicts can arise, for example, when:

- Versions of the same item have been saved on both branches.
- An item has been saved on one branch but has been deleted on the other branch.
- An item has been assigned a tag on one branch, but that tag has been deleted on the other branch.

Merge conflicts are indicated in the **Selection** icon column by ( ). Select the table row and click the **Take Source** button ( ) to overwrite all conflicting changes on the target branch with the corresponding source change. Click the **Keep Target** button ( ) to keep the target unchanged by skipping the source change.

Merging Conflicting Model Updates

The all or nothing choice for including a source change may be too coarse when a model has been updated on both branches. The model version on the source branch and the model version on the target branch can contain independent updates to the model tree, and it would then makes sense to incorporate both updates in a *merged model version* on the target branch. You can proceed as follows:

1. Open the model version on the target branch in the COMSOL Desktop.
2. Select the model version on the source branch in The Model Manager Window and click the **Compare** button ( ) in the **Item** section of the **Home** toolbar.

   The **Comparison Result** window is opened with a comparison between the model in the COMSOL Desktop — that is, the target model version — and the selected source model version.
3. Merge the changes you want to keep from the source model version into the opened model using the merge functionality in the **Comparison Result** window.
4. From the **File** menu, select **Save as Version** ( ).
5. Save a new version of the model to the target branch from the **Save** window. This becomes the merged model version.

6 Click the **Refresh** button ( ) in the **Merge** window's toolbar to recompute source and target branch changes.

7 If there is still a conflict between the source branch and target branch for the model update, which is expected, select the row and click the **Keep Target** button ( ) to keep the merged model version on the target.

If there is more than one update conflict between a model version on the source branch and the target branch, repeat these steps for each one. Once finished, finish the merge by clicking **Merge Changes** ( ).

THE MERGE DIALOG

The **Merge** dialog gives you a final chance to either go through with the merge or cancel it (except for any manually merged model versions already saved on the target branch — see Merging Conflicting Model Updates). The **Source location** field shows the location of the source commit, and the **Target location** field shows the target branch. You can write an optional comment in the **Comments** field for the commit created by the merge.

The table shows all changes that will be applied to the target branch by the merge commit. These changes may differ from the original source changes shown in the **Merge** window depending on which changes were included, which were skipped, and potential merge conflict resolutions.

Click **OK** to merge the changes in the database.

	Once a source commit has been merged into a target branch, any changes on the source branch that were skipped will not show up in the **Merge** window the next time you open the window for a newer source commit. To include such older source changes, manually perform them on the target branch.

	You may find the number of source changes in the **Merge** window overwhelming if there has been many commits on the source branch. One solution is to first merge from an older commit on the source branch, and then progressively work yourself up to the latest commit. One drawback is that this requires more than one merge commit on the target branch, which may unnecessarily pollute the commit history on that branch.

Reverting

In this section, you will learn how to undo some or all of the changes saved in a commit. This is valuable, for example, when you want restore a previously deleted model or file, revert to the version of an item that existed before a commit, or even revert a merge.

- Reverting Changes on a Branch
- The Revert Window

Reverting Changes on a Branch

When you revert a commit on a branch, Model Manager takes the set of changes done in the commit and computes the corresponding reverse changes. These changes are then saved as a new commit on the branch. A created item will be deleted, a deleted item will be recreated, and an updated item will be replaced by its previous version. A tag added to an item will be removed, and a tag removed from an item will be added.

There is a subtle difference between restoring a version — see Restore Version — and reverting a commit in which a version was saved. When selecting and restoring a version, you save *that* version as the new latest version. When selecting and reverting a commit, you save the version that *preceded* the commit's version as the new latest version.

Select a commit table row in The Commits Window and click the **Revert** button () in the toolbar to open The Revert Window for the selected commit.

The Revert Window

You use the **Revert** window to select and apply changes to a branch such that they revert changes made in a commit.

Reverting Changes on a Branch

The window contains a table with the reverting item changes. The table columns are:

- The type column — the type of the item to change represented by an icon.
- The **Change to Apply** column — a description of the reverting item change.
- The **Conflicting Latest Changes** column — one or more later changes on the branch that are incompatible, or *in conflict*, with a reverting change.
- The **Selection** column — an icon representing whether to apply the reverting change ( ), ignore the change ( ), or if there is a conflict ( ).

Click the **Apply Revert** button ( ) to open The Apply Revert Dialog after you have decided which changes to apply and resolved any revert conflicts.

Resolving Revert Conflicts

THE REVERT WINDOW TOOLBAR

The toolbar in the **Revert** window contains the following toolbar buttons:

- Click the **Refresh** button ( ) to refresh the table in case any new commits have been saved on the branch.
- Click the **Take Change to Apply** button ( ) to apply a reverting change to the branch.
- Click the **Keep Latest Change** button ( ) to skip a reverting change, keeping the latest change as is.
- Click the **Apply Revert** button ( ) to open The Apply Revert Dialog.

RESOLVING REVERT CONFLICTS

When you revert a commit on a branch, you may discover that the reverse changes are in conflict with the current item versions on the branch. Such revert conflicts can arise, for example, when:

- Reverting a commit in which an item version was saved, but a newer version of the item exists on the branch.
- Reverting a commit in which an item version was saved, but the item has been deleted on the branch in a newer commit.
- Reverting a commit in which an item was assigned a tag, but the tag has itself been deleted on the branch in a newer commit.

Revert conflicts are indicated in the **Selection** icon column by ( ). Select the table row and click the **Take Change to Apply** button ( ) to overwrite all conflicting latest changes on the branch with the corresponding reverting change. Click the **Keep Latest Change** button ( ) to keep the target unchanged by skipping the reverting change.

Resolving Conflicting Model Versions

Unlike merging from a source branch to a target branch, you may see less need to manually merge the model trees of conflicting model versions when reverting a commit on a branch — see Merging Conflicting Model Updates. Nevertheless, you can proceed as follows:

1. Open the older model version you want to revert to in the COMSOL Desktop.
2. Select the latest model version on the branch in The Model Manager Window and click the **Compare** button ( ) in the toolbar.

   The **Comparison Result** window is opened with a comparison between the model in the COMSOL Desktop — that is, the older model version to revert to — and the latest model version.
3. Merge the changes you want to keep from the latest model version into the opened model using the merge functionality in the **Comparison Result** window.
4. From the **File** menu, select **Save as Version** ( ).
5. Save a new version of the model to the branch from the **Save** window — this becomes the reverted model version. There will inevitably be Save Conflicts with the latest model version. Choose to ignore them.
6. Click the **Refresh** button ( ) in the **Revert** window's toolbar to recompute changes.
7. If there is still a conflict between the version being reverted to and the latest version for the model, which is expected, select the row and click the **Keep Latest Change** button ( ) to keep the latest model version.

If there is more than one conflict between an older model version being reverted to and a latest model version, repeat these steps for each one. Once finished, finish the revert by clicking **Apply Revert** ( ).

THE APPLY REVERT DIALOG

The **Apply Revert** dialog gives you a final chance to either go through with the revert or cancel it (except for any manually reverted model versions already saved on the target branch — see Resolving Conflicting Model Versions). The **Location** field shows

the branch on which the revert is made. You can write an optional comment in the **Comments** field for the commit created by the revert.

The table shows all changes that will be applied to the branch by the revert commit. These changes may differ from the original changes shown in the **Revert** window depending on which changes were included, which were skipped, and potential revert conflict resolutions.

Click **OK** to apply the changes in the database.

5

Working with Models in Databases

This chapter showcases the Model Manager tools available in the COMSOL Desktop modeling environment by way of a few tutorials. You will, for example, learn how to open and save versions of models in a database, how to browse, organize, and search models in a database, and how to use advanced version control tools such as reverting commits and creating branches.

In this chapter:

Example: Modeling Using Version Control

This section introduces some of the Model Manager tools you will typically encounter when working with models and data files stored in databases. The section uses a streamlined version of the tutorial *Example 1: Structural Analysis of a Wrench* found in *Introduction to COMSOL Multiphysics*. You are encouraged to first work through that tutorial if you are new to the COMSOL Multiphysics software.

Creating the Database

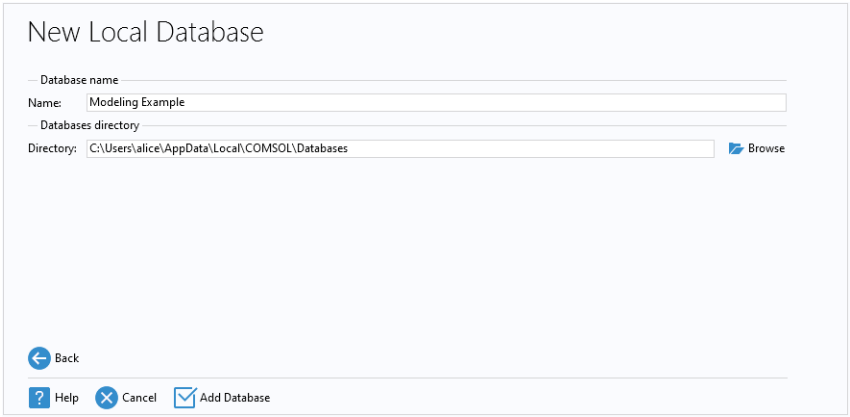
You are strongly recommended to create a new local database when working through the steps of this tutorial.

1 From the **File** menu, select **Open From** ().

2 In the **Open** window, choose **Add Database** () in the list of options.

3 In the **Add Database** window, choose **New Local Database** ().

4 In the **New Local Database** window, write `Modeling Example` as the name for the new database in the **Name** field.



5 Click **Add Database** ( ).

A progress window is displayed informing you that the database is being created on the file system. Once finished, the **Open** window is shown with the newly created database selected in the list.

From the **Open** window, you can search the database for models to open. At the moment your database is empty. Click **Cancel** ( ).

Adding Databases

Model Wizard Setup

You will use the **Model Wizard** to quickly get started with a new model for the structural integrity of a wrench.

1 From the **File** menu, select **New**. Choose **Model Wizard** in the **New** window.

2 In the **Select Space Dimension** window, choose **3D**.

3 In **Select Physics**, select **Structural Mechanics** > **Solid Mechanics (solid)** ( ). Click **Add**. Click **Study** ( ) to continue.

4 In **Select Study**, click **Stationary** ( ) under **General Studies**. Click **Done** ( ).

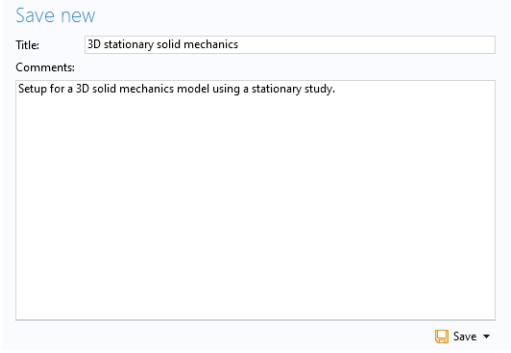
A new 3D solid mechanics model using a stationary study is initialized in the COMSOL Desktop.

Saving a First Version

Now is a good time to place your new model under *version control* by saving it to your database.

1. From the **File** menu, select **Save To** ( ).
2. In the **Save** window, choose your newly created database, `Modeling Example`, in the list of options.

   The **Save** window shows the selected database set as the target for the save. The header reads **Save new** as the model is not yet present in the database.
3. Write `3D stationary solid mechanics` in the empty **Title** field.
4. You can write an optional comment describing what you are saving in the **Comments** field. Write `Setup for a 3D solid mechanics model using a stationary study.`



5. Click the **Save** button ( ).

A first version of the model is now saved in the database.

Saving Models to Databases

Click the root node in the **Model Builder** window. The **Title** field in the **Presentation** section of the **Settings** window has been updated with the title you gave when saving.

Saving More Versions

The model uses a geometry that was previously created and stored in the COMSOL native CAD format `mphbin`.

1 In the **Model Builder** window, right-click **Geometry 1** ( ) and select **Import** ( ).

2 In the **Settings** window for **Import**, from the **Source list**, select **COMSOL Multiphysics file**.

3 Click **Browse** ( ) and locate the file `wrench.mphbin` in the application library folder of the COMSOL installation folder. Its default location in Windows® is

```
C:\Program Files\COMSOL\COMSOL63\Multiphysics\applications\
COMSOL_Multiphysics\Structural_Mechanics\wrench.mphbin
```

Double-click to add, or click **Open**.

4 Click **Import** ( ) to display the geometry in the **Graphics** window.

5 Select **Geometry 1** ( ) and click the **Build All** ( ) button in the **Settings** window.

COMSOL Multiphysics supports automatically detecting and cleaning up small details in an imported geometry that may prevent successful meshing and simulation. To analyze the imported wrench geometry for such details:

1 Select **Solid Mechanics (solid)** ( ).

2 In the opened **Geometry Cleanup** dialog, select **Clean up Automatically**.

3 Once the cleanup finishes, you may see a warning that the geometry still has a few issues in the **Settings** window. For the purpose of this tutorial, you can ignore this.

Geometry Cleanup in the *COMSOL Multiphysics Reference Manual.*

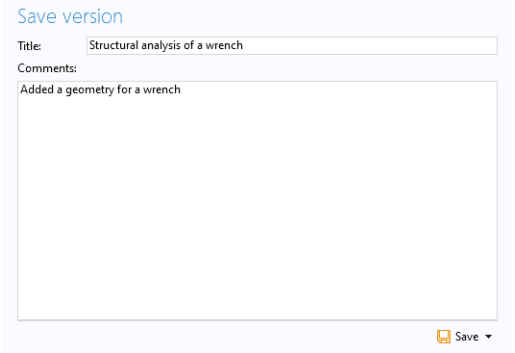
With a geometry added to the model, save a second version:

1 From the **File** menu, select **Save as Version** ( ).

The **Save** window opens with your database preselected in the list of options. The header reads **Save version** as the model already exists in the database. The same title you gave when saving the first version is suggested also for this second version.

2 In the **Title** field, change the title to `Structural analysis of a wrench`.

3 Write `Added a geometry for a wrench` in the **Comments** field.



4 Click the **Save** button ( ).

You now have two versions of the model saved in your database.

Add a description to your model and save a third version:

1 Click the root node in the Model Builder window.

2 In the **Description** field under the **Presentation** section, write `Analysis of the mechanical stress level in a wrench`.

3 From the **File** menu, select **Save as Version** ( ).

4 Expand the **Description** section on the right in **Save** window. The text area is prefilled with the same description you wrote in the **Presentation** section. You could have also written your new description directly here.

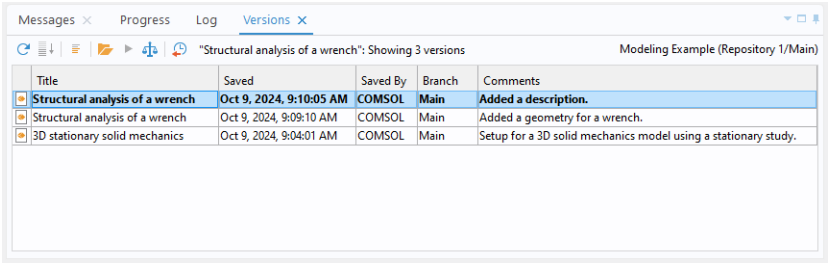
5 Write `Added a description` in the **Comments** field. Click **Save** ( ).

THE VERSIONS WINDOW

You have, up to this point, saved three versions of your model in the database.

From the **Windows** menu ( ) in the **Layout** section of Model Builder's **Home** toolbar, select **Versions** ( ) to open the **Versions** window. You will see your three versions in

a table. The top table row is highlighted in bold as it is opened in the COMSOL Desktop.



You can open an older version from the **Versions** window:

1. Select the bottom row in the table and click the **Open** button ( ) in the toolbar. You can also double-click the row. Select **No** if you are asked to save any unsaved changes.

   The first version is opened in the COMSOL Desktop. There is no **Import** ( ) node under the **Geometry 1** ( ) node, and the **Description** field is empty.

2. Select the middle row in the table and click **Open** ( ).

   The second version is opened in the COMSOL Desktop — the import node is now present under the geometry node, but the **Description** field is still empty.

3. Select the top row in the table and click **Open** ( ).

   The third, and latest, version is opened in the COMSOL Desktop.

The Versions Window for the COMSOL Desktop Model

Working With a Draft of the Model

It may have crossed your mind that saving a new model version requires several steps — especially as compared to just pressing Ctrl+S for a model opened from the file system. You need to open the **Save** window, perhaps think of a comment describing your changes (although the comment is not required), and then click the **Save** button ( ). You might even realize after saving multiple versions that your modeling work has gone in the wrong direction. You would then have a *version history* cluttered with unwanted versions.

A more lightweight option when working on a model is to save a *draft* of the model. You can save versions of this draft without affecting the original model. Once you are happy with your draft, you can save it back as a new version of the original model. You may of course choose to discard your draft altogether — instead opening the original model and, perhaps, starting a new draft.

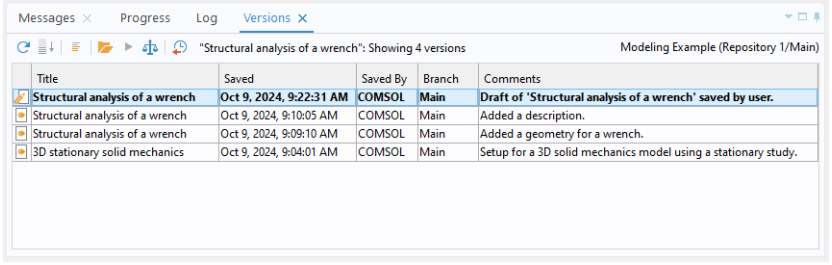
STARTING A DRAFT

You are going to continue the modeling of the wrench using a draft. Make sure that you have opened the latest (top) version in the **Versions** window.

Add a generic steel material for the wrench and save your work as a new draft.

1. Right-click **Component 1** > **Materials** ( ) and select **Add Material from Library** ( ).
2. In the **Add Material** window, click to expand the **Built-In** tree node. Scroll down to find **Structural steel**, right-click, and select **Add to Component 1**.
3. Close the **Add Material** window.
4. From the **File** menu, select **Save Draft** ( ). You can also press the keyboard shortcut Ctrl+S.

You have created a first version of a draft of the model. You can see this *draft version* as a new row in the **Versions** window on top of the three versions of the original model. The draft version uses a separate pen icon ( ) to distinguish it from the *regular versions* ( ). Note that the regular versions belong to the original model, not the draft itself — they are included in the table to make it easier for you to track where the draft originated from.



Saving Drafts of Models

SAVING ADDITIONAL DRAFT VERSIONS

Specify the load applied to the wrench and save your draft changes.

1 Select **Parameters 1** (P_1) in the **Model Builder** window.

2 In the **Settings** window's **Parameters** table, enter these settings:

- In the **Name** column, enter `F`.
- In the **Expression** column, enter `150[N]`.
- In the **Description** column, enter `Applied Force`.

3 From the **File** menu, select **Save Draft** ( ).

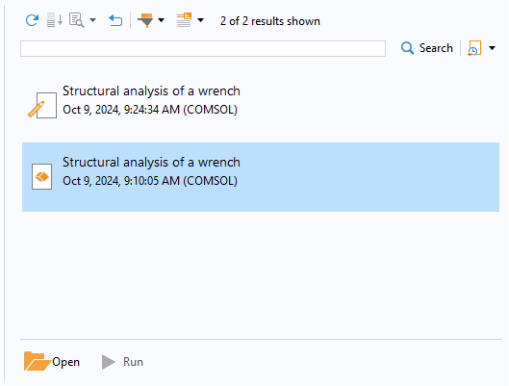
Selecting **Save Draft** a second time creates a second version of your draft — the **Versions** window now shows two draft versions and three regular versions. As for the regular versions of the original model, you can inspect an older draft version by selecting the row in the table and clicking **Open** ( ).

A draft is a model in its own right in the database — existing side by side with the original model. You can switch back and forth between them in the COMSOL Desktop simply by opening one or the other. To demonstrate this:

1 From the **File** menu, select **Open From** ( ).

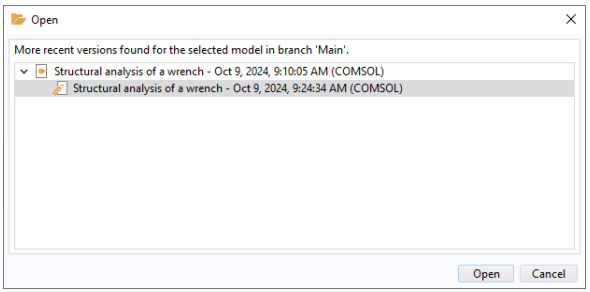
2 In the **Open** window, choose your database, `Modeling Example`, in the list of options.

The **Open** window shows the latest version of the draft ( ) and the latest version of the original model ( ) in the search result.



3 Select the version of the original model ( ) and click the **Open** button ( ).

Model Manager detects that there is an ongoing draft of the original model with a draft version newer than the latest version of the model. A dialog is shown in which you can choose to open that draft version instead.



4 Select the top node in the tree in the dialog and click **Open** to open the original model. The latest version of the model is opened in the COMSOL Desktop — neither the **Structural steel** material node nor the **Parameters** setting is present in the model tree, as expected.

Opening Models from Databases

You could at this point continue working with the original model, thereby implicitly discarding your draft work. The draft itself can be manually deleted from the database at some later time.

Choosing instead to continue with your draft, open its latest version again:

1 From the **File** menu, select **Open From** ( ).

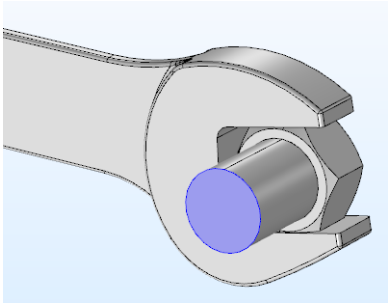
2 In the **Open** window, choose your database in the list of options.

3 Select the draft version ( ) and click **Open** ( ).

Finish the component setup by defining boundary conditions and mesh settings:

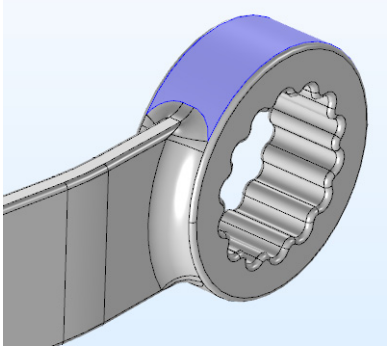
1 Right-click **Solid Mechanics (solid)** ( ) and select **Fixed Constraints** ( ).

2 In the **Graphics** window, rotate the geometry and select the front surface of the partially modeled bolt. The **Boundary** number in the **Selection** list is **35**.



3 Right-click **Solid Mechanics (solid)** ( ) once more and select **Boundary Load** ( ).

4 Select the top socket face (boundary 111) in the **Graphics** window.

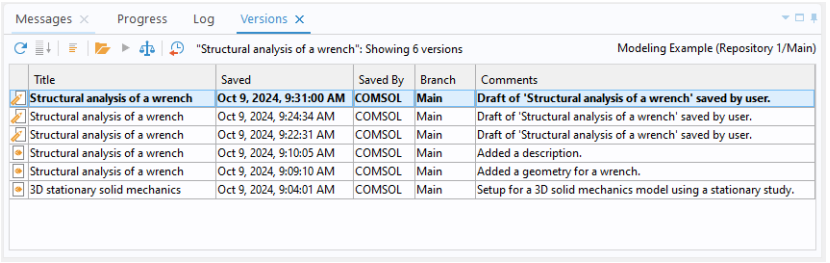


5 In the **Settings** window for **Boundary Load**, under **Force**, select **Total force** as the **Load type** and enter `-F` in the text field for the **z** component.

6 Select **Mesh 1** ( ). In the **Settings** window for **Mesh**, under **Physics-Controlled Mesh**, select **Finer** from the **Element size** list.

7 Click the **Build All** ( ) button in the **Settings** window.

With the basic setup finished, select **Save Draft** ( ) in the **File** menu to save a third draft version.



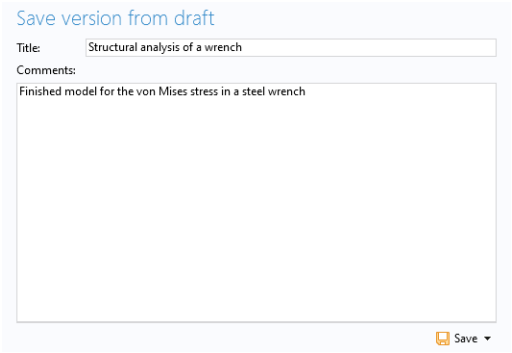
FINISHING YOUR DRAFT

With the component setup finished, it is time to save your draft work back to the original model:

1. From the **File** menu, select **Save as Version** ( ).

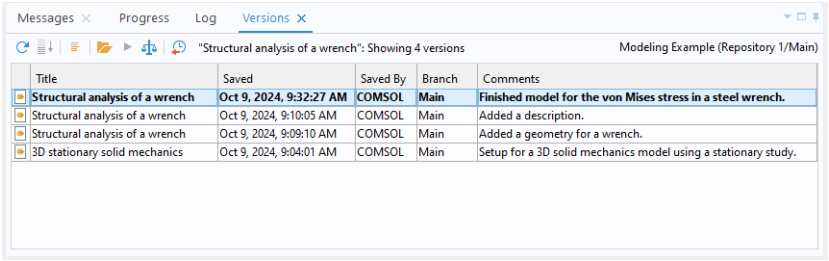
   The **Save** window opens with your database preselected in the list. The header reads **Save version from draft** as a new version of the original model will be saved from the draft.

2. Write `Finished model for the von Mises stress in a steel wrench` in the **Comments** field.



3. Click the **Save** button ( ).

Open the **Versions** window to see that all draft versions are now gone and replaced by your new, fourth, version of the original model.



The draft is automatically deleted when you save it back to the original model. This deletion is not permanent though — see Deleting Items and references therein to learn how you may recover your draft.

Comparing Versions

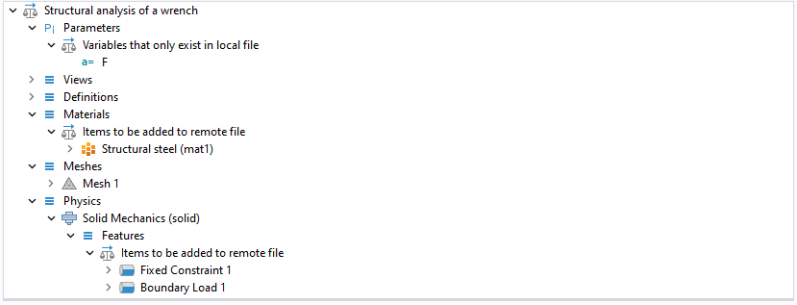
You can see all changes made to the model when you saved it from your draft.

1 Right-click the second row from the top in the **Versions** window and select **Compare** ( ).

The **Comparison Result** window is opened with a comparison between the current model in the COMSOL Desktop and the selected version.

2 In the **Comparison Result** window, click the **Expand All** button ( ) in the toolbar.

3 The expanded tree shows, for example, the force parameter, the steel material, the mesh settings, and the two boundary conditions, added from your draft.

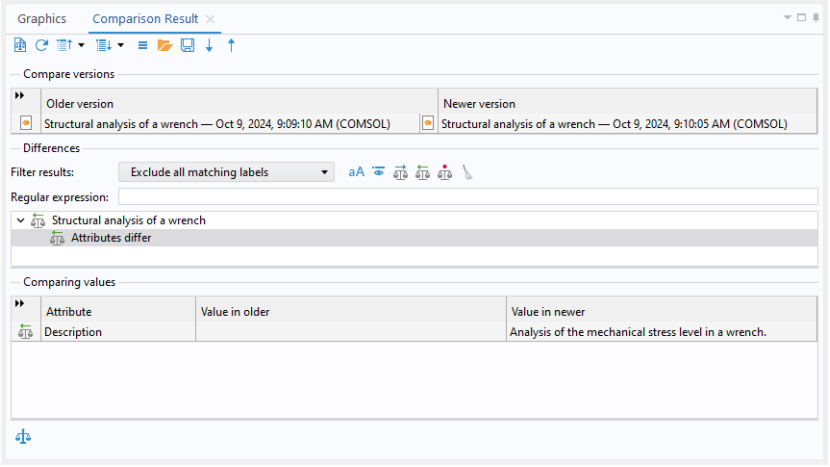


You can also compare two versions with each other:

1 Select the second and third rows in the **Versions** window, right-click either one, and select **Compare** ( ).

The **Comparison Result** window is updated with a comparison between these two versions.

2 Expand the tree and select the **Attributes differ** child node. In the **Comparing values** table below the tree you will find the description you added in the third model version.



3 Close the **Comparison Result** window.

Comparing Models Saved in Databases

Excluding Built, Computed, and Plotted Data

Right-click **Study 1** ( ) and select **Compute** ( ) to solve the model. When the computation finishes (it may take a few minutes depending on the performance of your computer), the von Mises stress is displayed in a default Volume plot in the **Graphics** window.

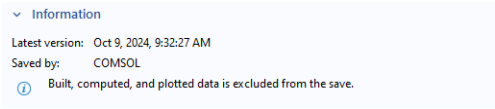
Storing simulation data generated by a model can require a large amount of disk space usage. For such data that is reproducible — such as *built, computed, and plotted data* — it may be undesirable, or even impossible due to sheer size, to save it in the database.

1 Select the root node in the **Model Builder** window.

2 In the **Settings** window, under **Built, computed, and plotted data** in the **Save** section, select **Exclude** in the **In database** list. You can leave the **On file** list as is.

3 From the **File** menu, select **Save as Version** ( ).

The **Save** window opens for your database with the message `Built, computed, and plotted data is excluded from the save` shown in the **Information** section.



4 Write `Saved without generated simulation data` in the **Comments** field.

5 Click **Save** ( ).

The model remains in its solved state in the COMSOL Desktop after the save. Go to the **File** menu, select **Revert to Saved** ( ), and click **Yes** in the dialog that appears. The latest saved version is opened. You may reproduce the, by now, lost solution by right-clicking **Study 1** ( ) and selecting **Compute** ( ).

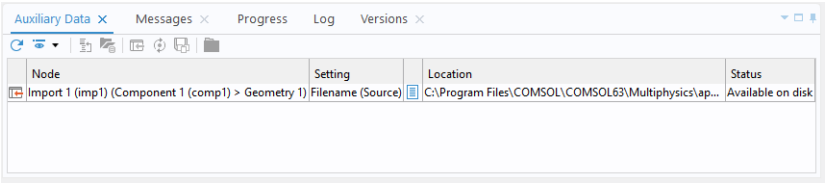
Built, Computed, and Plotted Data

Importing Auxiliary Data to the Database

You may have noticed that, while the model is version controlled in the database, the same is not true for the CAD *input file*. You can import the file to the database as follows:

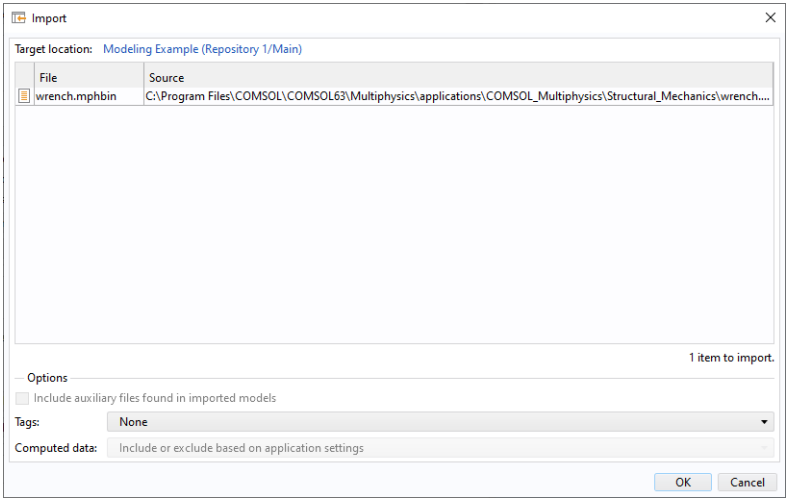
1 From the **Windows** menu ( ) in the **Layout** section in the **Home** toolbar, select **Auxiliary Data** to open the **Auxiliary Data** window.

The **Auxiliary Data** window shows a table with input and output data referenced by nodes in the model tree. In this case, a single row for the CAD file used by the **Import** ( ) node is shown.



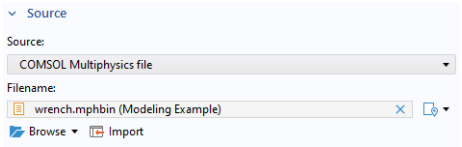
2 Select the table row, right-click, and select **Import to Database** ( ).

3 The **Import** dialog shows the file `wrench.mphbin` in a table.



4 Click **OK** to import the file into the database.

Select the **Import** ( ) node in the **Model Builder** window. The **Filename** field in the **Import** section in the **Settings** window now shows a reference to the file imported into the database.



Finish by saving the model to the database:

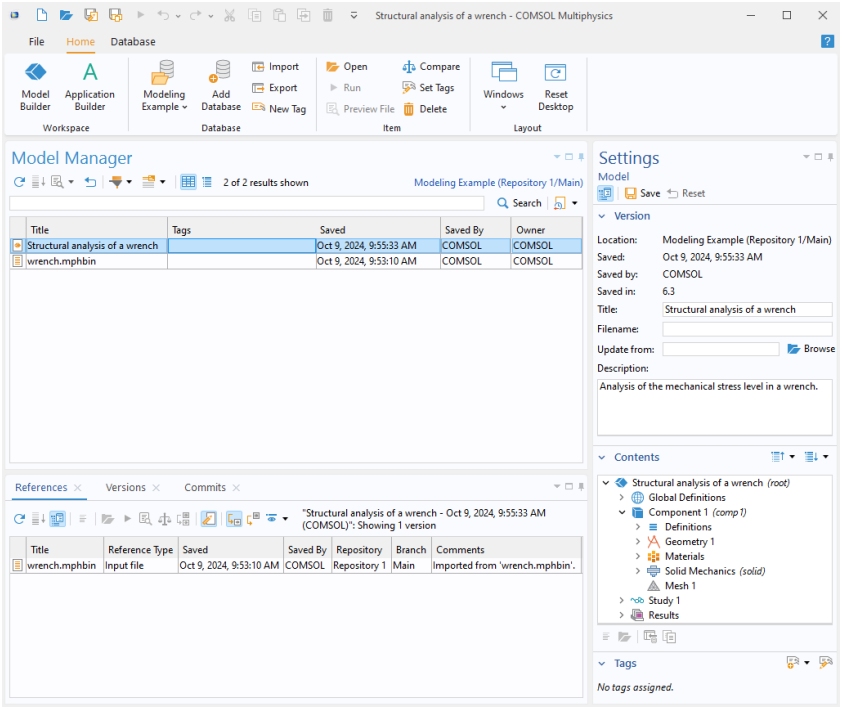
1 From the **File** menu, select **Save as Version** ( ).

2 Write `Referenced CAD file from database` in the **Comments** field. Click **Save** ( ).

The Auxiliary Data Window for Database Input and Output

The Model Manager Workspace

Click the **Model Manager** ( ) button in the **Workspace** section of the **Home** toolbar in Model Builder to open The Model Manager Workspace — a workspace in the COMSOL Desktop dedicated to database-specific tasks. The COMSOL Desktop switches to display the toolbar for the Model Manager, as well as opens windows belonging to the workspace. You will find the latest versions of your model and CAD file in The Model Manager Window.



To learn more on what you can do in the workspace, you can, for example:

- Select the model and expand the model tree in The Contents Section of The Settings Window. You will find that you can browse the content of a model without opening it.
- Search for your model by applying various Item and Content Filters, for example, a Physics filter on a Solid Mechanics interface ( ) or a Parameter filter for the applied force of `150[N]`.
- Right-click the model and select **References** ( ) to see the database relationship between the model and the CAD file in The References Window.
- Right-click the CAD file and select **Versions** ( ) to see all versions of the file — currently only one — in The Versions Window.
- Right-click the model and select **Commits** ( ) to open The Commits Window. Select the third table row from the top. In the **Settings** window, you will see details on the commit in which a new version of the original model was saved from your draft, and the draft itself was deleted.

This concludes this introductory tutorial. You are encouraged to further explore the Model Manager workspace by working through the remaining example tutorials in this chapter.

Example: Browsing, Organizing, and Searching Models and Data Files

In this tutorial section, you will learn how you can browse and search for models and data files stored in a database using the Model Manager search functionality. You will also learn how you can assign tags to your models and data files to achieve better organization in the database, thereby making it easier to retrieve these items in the future. The tutorial uses a demo database containing models and data files imported from the COMSOL Application Libraries. You can download and open this local database from within the COMSOL Desktop environment.

- Downloading the Demo Database for Model Manager
- Searching and Browsing the Demo Database
- Using a Tag Tree for Organization and Retrieval in a Database
- Creating and Assigning Tags
- Searching on Model Contents
- Using the Model Manager Search Syntax

Downloading the Demo Database for Model Manager

To add the demo database for Model Manager to the COMSOL Desktop, select **Download Demo Database for Model Manager** ( ) in the **File** > **Help** menu. The database is downloaded as a compressed archive from the COMSOL web site and unpacked to the default **Directory for local databases** as set in the **Preferences** window — see also New Local Database. Once finished, The Model Manager Workspace is automatically opened with the demo database preselected in the **Database** section in the **Home** toolbar.

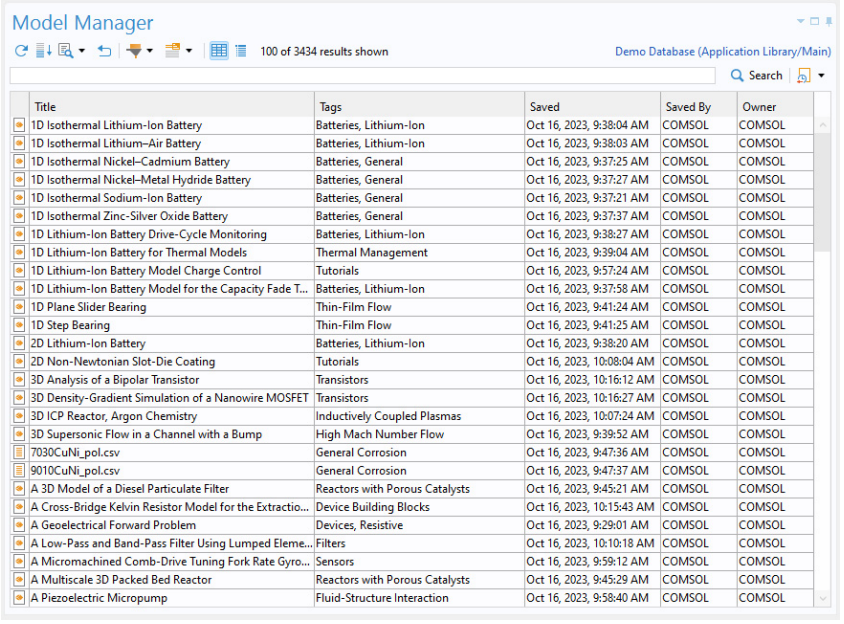
You can also manually download the demo database directly from the COMSOL web site — see the instructions on https://www.comsol.com/model/demo-database-for-model-manager-104691. This is useful if you, for example, want to keep a pristine backup of the database for future reference once you have finished the tutorials in this chapter.

Searching and Browsing the Demo Database

The demo database contains a few thousand items that have been imported to the database from files found in the COMSOL Application Libraries. These are files you can use, for example, to learn how to build models in the COMSOL Desktop modeling environment or use as templates for your own simulation models and applications. This tutorial shows how to use these models and data files as an example of a larger database in which the Model Manager tools for browsing, searching, and organizing items become useful.

The Application Libraries Window in the *COMSOL Multiphysics Reference Manual*

The **Model Manager** window lists these models and data files in a table. If you scroll down the table, you will notice that the table only contains the first 100 items in the database sorted alphabetically on their titles. This cutoff is an optimization to not cause unnecessary data traffic between the COMSOL Desktop and the database — something that may be especially important when connected to a Model Manager server over the network.



Title	Tags	Saved	Saved By	Owner
1D Isothermal Lithium-Ion Battery	Batteries, Lithium-Ion	Oct 16, 2023, 9:38:04 AM	COMSOL	COMSOL
1D Isothermal Lithium–Air Battery	Batteries, Lithium-Ion	Oct 16, 2023, 9:38:03 AM	COMSOL	COMSOL
1D Isothermal Nickel–Cadmium Battery	Batteries, General	Oct 16, 2023, 9:37:25 AM	COMSOL	COMSOL
1D Isothermal Nickel–Metal Hydride Battery	Batteries, General	Oct 16, 2023, 9:37:27 AM	COMSOL	COMSOL
1D Isothermal Sodium-Ion Battery	Batteries, General	Oct 16, 2023, 9:37:21 AM	COMSOL	COMSOL
1D Isothermal Zinc-Silver Oxide Battery	Batteries, General	Oct 16, 2023, 9:37:37 AM	COMSOL	COMSOL
1D Lithium-Ion Battery Drive-Cycle Monitoring	Batteries, Lithium-Ion	Oct 16, 2023, 9:38:27 AM	COMSOL	COMSOL
1D Lithium-Ion Battery for Thermal Models	Thermal Management	Oct 16, 2023, 9:39:04 AM	COMSOL	COMSOL
1D Lithium-Ion Battery Model Charge Control	Tutorials	Oct 16, 2023, 9:57:24 AM	COMSOL	COMSOL
1D Lithium-Ion Battery Model for the Capacity Fade T...	Batteries, Lithium-Ion	Oct 16, 2023, 9:37:58 AM	COMSOL	COMSOL
1D Plane Slider Bearing	Thin-Film Flow	Oct 16, 2023, 9:41:24 AM	COMSOL	COMSOL
1D Step Bearing	Thin-Film Flow	Oct 16, 2023, 9:41:25 AM	COMSOL	COMSOL
2D Lithium-Ion Battery	Batteries, Lithium-Ion	Oct 16, 2023, 9:38:20 AM	COMSOL	COMSOL
2D Non-Newtonian Slot-Die Coating	Tutorials	Oct 16, 2023, 10:08:04 AM	COMSOL	COMSOL
3D Analysis of a Bipolar Transistor	Transistors	Oct 16, 2023, 10:16:12 AM	COMSOL	COMSOL
3D Density-Gradient Simulation of a Nanowire MOSFET	Transistors	Oct 16, 2023, 10:16:27 AM	COMSOL	COMSOL
3D ICP Reactor, Argon Chemistry	Inductively Coupled Plasmas	Oct 16, 2023, 10:07:24 AM	COMSOL	COMSOL
3D Supersonic Flow in a Channel with a Bump	High Mach Number Flow	Oct 16, 2023, 9:39:52 AM	COMSOL	COMSOL
7030CuNi_pol.csv	General Corrosion	Oct 16, 2023, 9:47:36 AM	COMSOL	COMSOL
9010CuNi_pol.csv	General Corrosion	Oct 16, 2023, 9:47:37 AM	COMSOL	COMSOL
A 3D Model of a Diesel Particulate Filter	Reactors with Porous Catalysts	Oct 16, 2023, 9:45:21 AM	COMSOL	COMSOL
A Cross-Bridge Kelvin Resistor Model for the Extractio...	Device Building Blocks	Oct 16, 2023, 10:15:43 AM	COMSOL	COMSOL
A Geoelectrical Forward Problem	Devices, Resistive	Oct 16, 2023, 9:29:01 AM	COMSOL	COMSOL
A Low-Pass and Band-Pass Filter Using Lumped Eleme...	Filters	Oct 16, 2023, 10:10:18 AM	COMSOL	COMSOL
A Micromachined Comb-Drive Tuning Fork Rate Gyro...	Sensors	Oct 16, 2023, 9:59:12 AM	COMSOL	COMSOL
A Multiscale 3D Packed Bed Reactor	Reactors with Porous Catalysts	Oct 16, 2023, 9:45:29 AM	COMSOL	COMSOL
A Piezoelectric Micropump	Fluid-Structure Interaction	Oct 16, 2023, 9:58:40 AM	COMSOL	COMSOL

Click the **Show More** button ( ) in the toolbar to append the next 100 items to the bottom of the table. You can repeat this as many times as you like until all items found in the database have been appended to the table.

You use the **Model Manager** window to find models and data files in the database by writing search terms in the text field above the table.

1 Write `busbar` and click **Search**. You can also press Enter.

   The table is updated to only show the items whose title, description, assigned tags, or filename fields match the word busbar. In this case, the number of search results is less than 100 so the **Show More** button ( ) is disabled.

2 Write `bus` and click **Search**.

   The search terms you write in the text field will only match complete words by default. In this case, you get zero results as there are no items with the word bus in the searched item fields. You can append a *wildcard* asterisk character to match words that start with a search term. Write for example `bus*` to get search results.

3 Write `electrical heating` and click **Search**.

   When you write multiple search terms separated by spaces, the search will match on all terms using Boolean AND-logic. In this case, you find all items for which both the word electrical and the word heating is found in the searched item fields.

Select the **Thermal Analysis of a Bipolar Transistor** model ( ) in the table. The **Settings** window updates to show various metadata fields for this model. You will find that the **Description** field for the model contains the two search words electrical and heating, although not next to each other.

You can require that all search terms must match as a sentence by enclosing the search terms with quotations. Write `"electrical heating"` in the text field and click **Search**. The **Thermal Analysis of a Bipolar Transistor** model ( ) is now no longer present in the search result.

Click the **Reset** button ( ) to clear the text field and restore the search to match all models and data files in the database.

The Model Manager Window

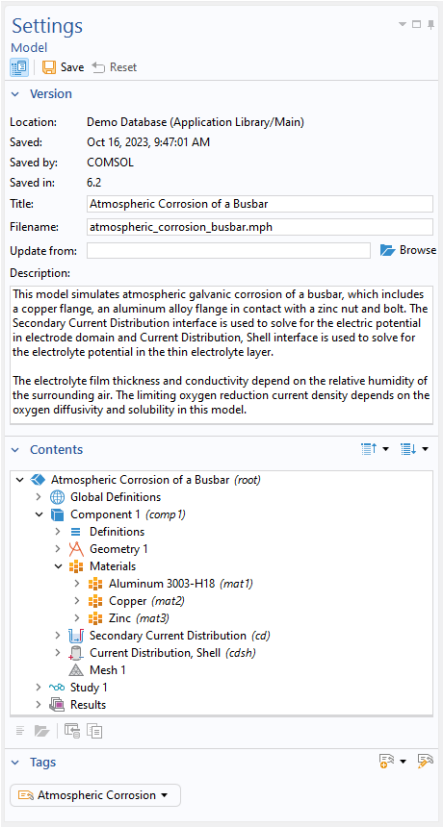
The default behavior in the **Model Manager** window is to search the latest versions of items. This is also the version shown in the **Settings** window when you select a model

or data file in the search result table. From the **Settings** window, you can, for example, see the point in time when the version was saved in the **Saved** field, the name of the user that saved the version in the **Saved by** field, and the title of the version in the **Title** field.

1 Write `busbar` in the text field again and click **Search**.

2 Select the **Atmospheric Corrosion of a Busbar** model ( ) in the search result table.

In the **Contents** section in the **Settings** window, you can explore the model tree of the model as it looked when the version was saved to the database. Expand some of the nodes in the tree to discover that this is a model with a `3D` component node containing three material nodes and two physics interface nodes.

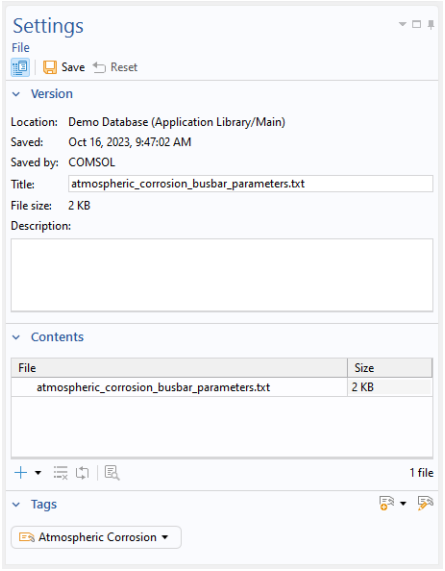


From the **Contents** tree, you can insert components, geometry parts, materials, parameters, and many other nodes into the model currently opened in the COMSOL Desktop using the **Insert into Model** button ( ). You can also select a

node and click **Open Node** ( ) to open the model with the node selected in the **Model Builder** window.

3 Select the **atmospheric_corrosion_busbar_parameters.txt** file ( ) in the search result table.

This is a file containing parameters that can be used in a **Parameters** node in the **Model Builder** window. You can double-click the table row in the **Model Manager** window to preview the data file with the default application used for a text file on your computer. The **Settings** window shows, for example, the file size when the file is stored on a file system.



4 In the **Description** field in the **Settings** window, write `Parameters for modeling atmospheric corrosion of a busbar`. Click the **Save** button ( ).

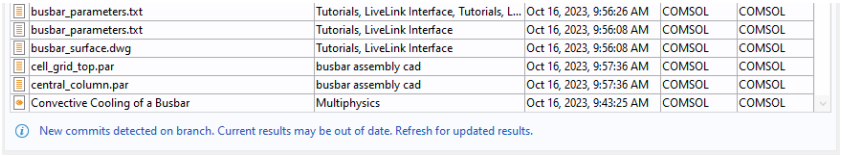
The **Save File** dialog is opened with a suggested save comment.

5 Change the suggested save comment to `Updated the file description`.

6 Click **OK** to save the changed description in a new version of the data file.

The **Saved** and **Saved by** fields are updated in the **Settings** window to reflect the new latest version. The **Model Manager** window is, however, not automatically updated.

Click the link text informing you that the current search result is out of date at the bottom of the window to refresh the search result.



7 Select **busbar_assembly.iam** ( ) in the search result table.

This is a *fileset*, an item containing multiple data files that are version controlled as a collective whole. In this case, the data is a CAD assembly containing a main assembly file and several component files. You can see the file contents of the fileset in the **Contents** section in the **Settings** window.

8 Click the **Reset** button ( ) to reset the search result.

The **Settings** window is where you primarily view and update items in a Model Manager database. When you click the **Save** button ( ) in the top toolbar, a new version is saved using the current values found in the various fields in the window. You can, for example, change the title or description of an item, update a model from an MPH-file on your computer, or update the contents of a file or fileset using data files on your computer. The **Settings** window is also where you update other types of objects found in a database.

The Settings Window

APPLYING BASIC FILTERS

Writing search terms in the text field in the **Model Manager** window is often the quickest way of finding a particular model or data file given that you know, for example, its title. As a complement to this, you can apply separate search filters.

1 Click the **Add Filter** button ( ) in the toolbar of the **Model Manager** window and select **Filename** ( ).

The **Filter** dialog is opened enabling you to apply a filter on the filename used by a model or data file when exported to the file system.

2 In the **Keyword** field, write `busbar.mph`. Click **OK**.

The search result contains a single model. Select the model and verify that the **Filename** field in the **Settings** window indeed has the expected value.

A filter on a filename must include the file extension to return any matches in the search result. Alternatively, you can use a wildcard in the filter:

1 Click on the **busbar.mph** filter pill ( ) and select **Edit** ( ).

2 In the **Keyword** field, change to `busbar`.

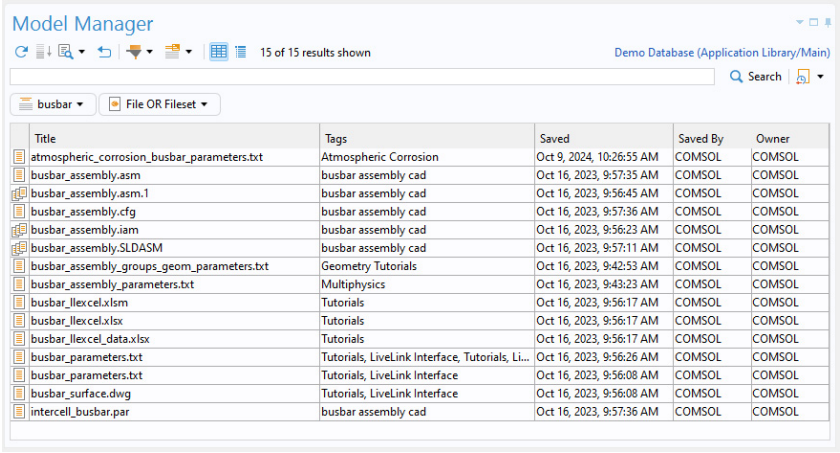
3 Select **Prefix Match** under **Options** to automatically append a wildcard asterisk character to the searched value. Click **OK**.

The search result contains multiple models and data files, all with a filename starting with busbar.

4 Click on the **busbar*** filter pill ( ) and select **Remove** ( ) to remove the filter.

As you may have previously noticed, writing `busbar` in the text field and clicking **Search** ( ) led to a search result containing items that did not have that particular word in their titles. Instead the word busbar was found in the description or assigned tags of the items. You can address this by applying an explicit filter on the title field. Click the **Add Filter** button ( ) and select **Title** ( ). In the **Filter** dialog, write `busbar` in the **Text** field. Click **OK**. The search result now only contains models and data files with the word busbar in their titles.

You can add as many filters as you like. Click the **Add Filter** button ( ) in the toolbar again and select **Item Version Type** ( ). Select the **File** and **Fileset** checkboxes. Click **OK**. The search result is further reduced to only contain data files.



Title	Tags	Saved	Saved By	Owner
atmospheric_corrosion_busbar_parameters.txt	Atmospheric Corrosion	Oct 9, 2024, 10:26:55 AM	COMSOL	COMSOL
busbar_assembly.asm	busbar assembly cad	Oct 16, 2023, 9:57:35 AM	COMSOL	COMSOL
busbar_assembly.asm.1	busbar assembly cad	Oct 16, 2023, 9:56:45 AM	COMSOL	COMSOL
busbar_assembly.cfg	busbar assembly cad	Oct 16, 2023, 9:57:36 AM	COMSOL	COMSOL
busbar_assembly.iam	busbar assembly cad	Oct 16, 2023, 9:56:23 AM	COMSOL	COMSOL
busbar_assembly.SLDASM	busbar assembly cad	Oct 16, 2023, 9:57:11 AM	COMSOL	COMSOL
busbar_assembly_groups_geom_parameters.txt	Geometry Tutorials	Oct 16, 2023, 9:42:53 AM	COMSOL	COMSOL
busbar_assembly_parameters.txt	Multiphysics	Oct 16, 2023, 9:43:23 AM	COMSOL	COMSOL
busbar_llexcel.xlsm	Tutorials	Oct 16, 2023, 9:56:17 AM	COMSOL	COMSOL
busbar_llexcel.xlsx	Tutorials	Oct 16, 2023, 9:56:17 AM	COMSOL	COMSOL
busbar_llexcel_data.xlsx	Tutorials	Oct 16, 2023, 9:56:17 AM	COMSOL	COMSOL
busbar_parameters.txt	Tutorials, LiveLink Interface, Tutorials, Li...	Oct 16, 2023, 9:56:26 AM	COMSOL	COMSOL
busbar_parameters.txt	Tutorials, LiveLink Interface	Oct 16, 2023, 9:56:08 AM	COMSOL	COMSOL
busbar_surface.dwg	Tutorials, LiveLink Interface	Oct 16, 2023, 9:56:08 AM	COMSOL	COMSOL
intercell_busbar.par	busbar assembly cad	Oct 16, 2023, 9:57:36 AM	COMSOL	COMSOL

Click the **Reset** button ( ) to reset the search result.

Using a Tag Tree for Organization and Retrieval in a Database

There are various ways a user of the COMSOL Multiphysics software may organize their simulation work when relying solely on the file system for storage. Perhaps they have set up a folder structure containing their MPH-files and associated data files. The files are maybe placed in various subfolders corresponding to different projects (in a generic sense). Perhaps there is a shared folder with data files used as input in several models and in different projects. There could also be a template folder with a few models containing some common setups and to be used as starting points for new projects. Version control of files could be achieved via a strict naming convention: say `model_v1.mph`, `model_v2.mph`, `model_v3.mph`, and so on and `data_file_v1.txt`, `data_file_v2.txt`, `data_file_v3.txt`, and so on. To keep track of the ongoing work, each series of version-controlled MPH-files could perhaps have an associated work log file that describes what was changed between each version. Each data file might have an associated *used-by* text file, containing a list of MPH-files known to use this particular data file as input (or there could a be single spreadsheet document containing a table with all such relations).

The Model Manager tools in the COMSOL Desktop modeling environment helps you with these organizational and version-control aspects of your simulation work by, for example, keeping a version history of items, automatically keeping track of references between items, and enabling you to save comments, descriptions, and other metadata with your items. One such organizational metadata is the concept of assigning *tags* to items, which is the Model Manager analogue to placing such items in folders on your computer. Much like folders on the file system, tags can be assigned to other tags, resulting in a *tag tree* structure of tags, models, and data files.

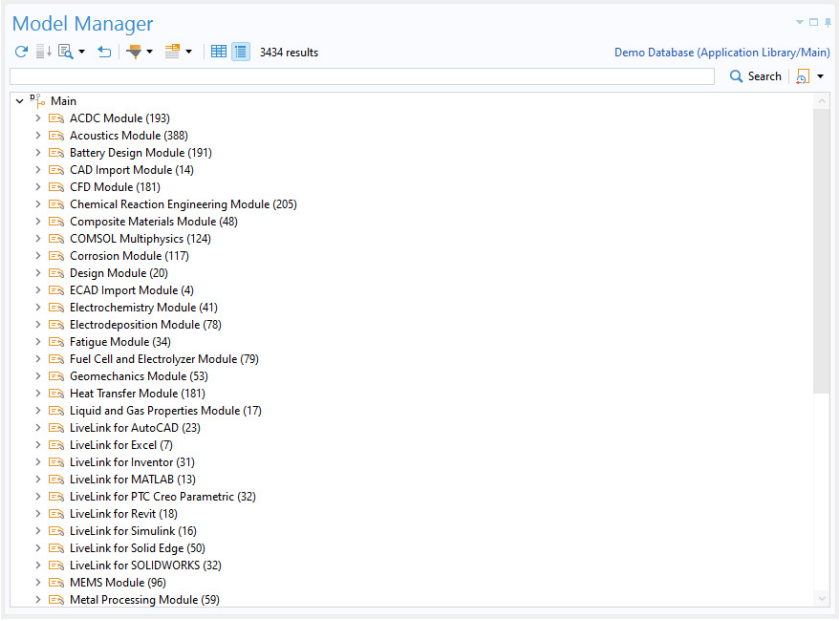
The demo database for Model Manager already contains such a tag tree. These tags were automatically created from the corresponding folders that contained the models and data files in the COMSOL Application Libraries when these files were imported into the database.

The folder structure in the COMSOL Application Libraries, and thus the resulting tag tree in the demo database for Model Manager, is heavily influenced by the COMSOL product suite itself — with emphasis on the available add-ons and interfacing products, and their associated model tutorials. The tag structure you create in your own Model Manager databases are likely to show little resemblance to this tree.

You can view the tag tree from the **Model Manager** window by switching from the Table View to the Tree View. While the former view is useful when you quickly want to find models and data files given a specific set of search and filter criteria, the latter is more suited to browsing the items in the database without having a particular model or data file in mind.

1 In the **Model Manager** window, click the **Tree** button ( ) in the toolbar.

The **Model Manager** window switches to show a tree of tags ( ). There are about 50 tags at the top level, with several hundred more tags found under these top tags.



2 Expand the tags **ACDC Module** > **Tutorials, Cables**. You will find about 40 items under the last tag tree node. In the Model Manager database, these items are all assigned a tag with title **Tutorials, Cables**. The **Tutorials, Cables** tag is itself assigned the **ACDC Module** tag. The number of items found under each tag tree node in the search result is written within parentheses next to the tag's title.

3 Write `tutorials, cables` in the text field and click **Search**. The tree only shows the items assigned the tag, the **Tutorials, Cables** tag itself, and the **ACDC Module** tag.

Punctuation characters, such as commas, are ignored when searching in a Model Manager database. Write `tutorials cables` to get the same search result.

4 Write `acdc module` and click **Search**. The search result contains all items found under the **ACDC Module** tag. You may be surprised, however, to discover a few other tags on the top level in the tag tree.

5 Expand **ACDC Module** > **Electromagnetics and Optimization**. One of the *tagged* items is the **Topology Optimization of a Magnetic Circuit** model ( ). Also expand **Acoustics Module** > **Optimization** and notice that the same model is in fact also found under these latter tags.

You can assign multiple tags to the same item in a Model Manager database. In this case, Model Manager discovered during the import that the model **Topology Optimization of a Magnetic Circuit** was stored as identical MPH-files in multiple folders in the COMSOL Application Libraries. Rather than importing duplicate models, a single model was imported and assigned multiple tags.

6 Write `topology optimization magnetic circuit` and click **Search**. You will discover that the model, together with a few other items, was assigned three different tags during the import.

7 Click the **Reset** button ( ) to clear the current search terms and restore the tag tree. Click the **Table** button ( ) to switch back to the Table View.

BROWSING THE TAG TREES OF MULTIPLE DATABASES

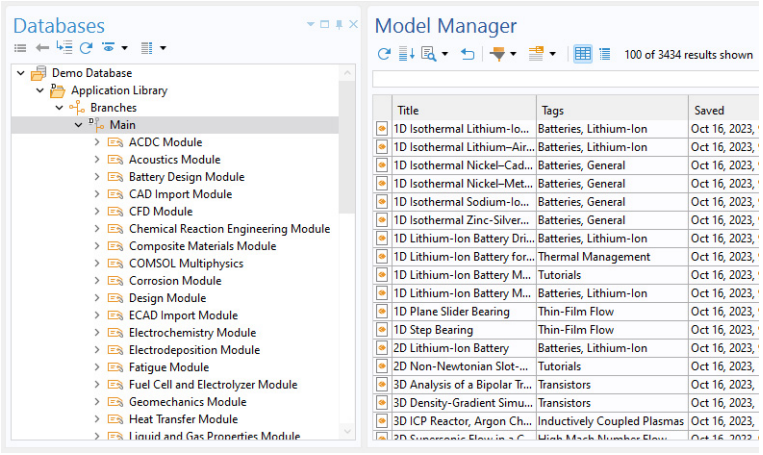
The **Model Manager** window enables you to search and browse a specific branch in a specific repository in a Model Manager database. If you would prefer to browse multiple branches in different repositories, and perhaps in different databases, at once, you can use the **Databases** window.

1 In the **Database** toolbar, in the **Database** section, click the **Databases** button ( ).

The **Databases** window opens next to the **Model Manager** window. The window shows all databases that you have added to the COMSOL Desktop as a tree.

2 Expand the **Main** ( ) branch node for the demo database.

The same tag tree available in the **Model Manager** window is shown as a subtree to the branch node. Unlike the **Model Manager** window, however, there is no search functionality available in the window.



3 If you have added any other databases than the demo database, you are encouraged to explore their database subtrees as well. This includes, for example, the database created in the tutorial Example: Modeling Using Version Control.

4 Click the **Databases** button ( ) again to close the **Databases** window.

The **Databases** window is also where you perform more advanced database operations including, for example, creating and merging branches or managing users and groups in a Model Manager server database.

The Databases Window

Creating and Assigning Tags

The tags in the demo database were all created from folders on the file system. You can also create new tags directly in the database and assign them to your models and data files.

1 Write `tutorials` in the text field in the **Model Manager** window and click **Search**.

   The search result contains about 500 items, all somehow related to various modeling tutorials for the COMSOL Desktop modeling environment

2 Click the **Tree** button () in the toolbar to show the tag tree.

3 Browse the tree by expanding some of the tag tree nodes. You will quickly discover that the various tutorial models are spread out over a multitude of tags with various tutorial-related titles.

You are going to create a new tag in the database that you will assign to all models used in tutorials. Having such a tag will make it easier to quickly find all tutorials in the future.

1 Click the **Table** button () to switch back to the Table View.

2 Click the **Add Filter** button () in the toolbar and select **Item Version Type** ().

   The **Filter** dialog is opened enabling you to apply a separate filter on the types of item versions to include in the search result.

3 In the **Filter** dialog, select the **Model** checkbox. Click **OK**.

The search result is reduced to about 200 models.

1 Repeatedly click the **Show More** button () in the toolbar until all models have been appended to the table.

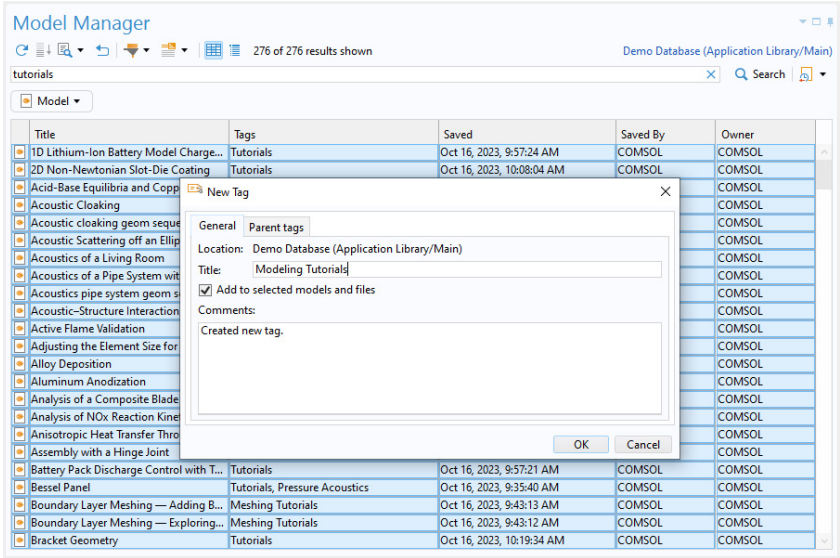
2 Select one of the table rows and press Ctrl+A to subsequently select all table rows.

3 In the **Home** toolbar, in the **Database** section, click the **New Tag** button ().

   The **New Tag** dialog is opened with the checkbox **Add to selected models and files** already selected. The dialog enables you to create a new tag and simultaneously assign the tag to the current selection of models and data files in the **Model Manager** window.

4 Write `Modeling Tutorials` in the **Title** field. Click **OK**.

The save takes as few seconds as Model Manager makes the new tag assignment available for searching and filtering.



5 At the bottom of the **Model Manager** window, click the link text informing you that the current search result is out of date.

The current search is reloaded. Notice that the **Tags** column contains your new tag assigned to all items in the search result.

6 Click the **Reset** button ( ) to clear the current search term and filter. Click the **Tree** button ( ) in the toolbar to switch to the Tree View. Expand the **Modeling Tutorials** tag tree node to find all tutorial models.

The **Model Manager** window shows the first 100 items under a tag tree node. Expand the **Show More** tree node ( ) to reveal the next 100 items.

7 Click the **Table** button ( ) to return to the Table View.

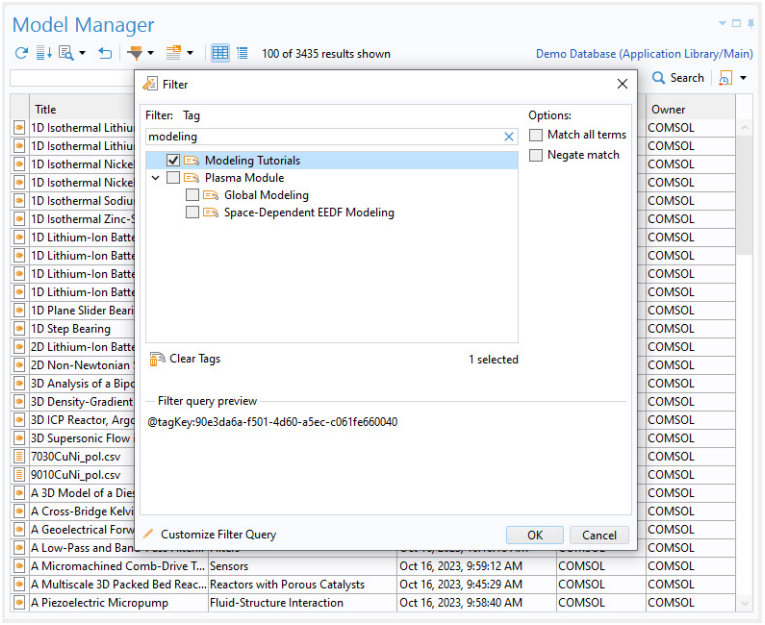
Assigning Tags to Items

With the new tag for modeling tutorials in the database, it is a simple matter of quickly finding such models by applying a tag filter.

1 Click the **Add Filter** button ( ) in the toolbar and select **Tag** ( ).

The **Filter** dialog is opened enabling you to apply a separate filter on the assigned tags of items to include in the search result.

2 Write `modeling` in the text field above the tag tree to filter the available tags on their titles.



3 Select the **Modeling Tutorials** checkbox and click **OK**.

The search result only includes the models that are assigned the **Modeling Tutorial** tag ( ). From here one can continue to drill down in the search result by writing search terms or applying additional filters.

4 Click on the **Modeling Tutorials** filter pill ( ) and select **Remove** ( ) to remove the filter.

In this case, you could have also found all tutorial models by simply writing `modeling tutorials` in the text field and clicking **Search**. But for tags whose titles are likely to also appear in the titles or descriptions of model and data files, using an explicit **Tag** filter ( ) is a better strategy.

ASSIGNING EXISTING TAGS TO ITEMS

The tags in the demo database are all of a purely descriptive nature related to what the models and data files contain. You can also use tags for simple project and workflow management.

1. Click the **New Tag** button () in the **Database** section in the **Home** toolbar.
2. In the **Title** field, write `Projects`.
3. Make sure that the checkbox **Add to selected models and files** is cleared. Click **OK**.

A new tag that can be used to group models and data files involved in various simulation projects is created in the database.

1. Click the **New Tag** button () again.
2. In the **Title** field, write `In Progress`.
3. Click the **Parent tags** tab.

   A tag tree is shown enabling you place the new tag under an existing tag.
4. Find and select the **Projects** tag () — for example by typing `projects` in the text field above the tree.
5. Click **OK** to create the tag.

For the purpose of this part of the tutorial, assign the **In Progress** tag to a few select models:

1. Write `electrical heating` in the **Model Manager** window and click **Search**.
2. Select one of the table rows and press Ctrl+A to subsequently select all table rows.
3. In the **Home** toolbar, in the **Database** section, click the **Set Tags** button ().

   The **Set Tags** dialog is opened enabling you to set the assigned tags of one or more items. You will notice that some of the checkboxes in the tag tree have an *indeterminate* selection. These are tags that are assigned to some, but not all, of the items. Leaving these indeterminate checkboxes as-is means that these assignments will not be modified when you click **OK**.
4. Find and select the **In Progress** tag — for example by typing `progress` in the text field above the tree.
5. Click **OK** to assign the **In Progress** tag to the items.

Much like the modeling tutorials, you can now find these models and data files by searching on the tag title or by applying the tag as a filter. Perhaps you use the **In Progress** tag (), for example, as a way of signaling to your coworkers that you are

currently working on updates to these models, which could be a more lightweight alternative to setting restrictive permissions for the models.

Adding and Removing Tag Assignments

ASSIGNING TAGS WHEN SAVING MODELS

So far you have worked exclusively with tags from the Model Manager workspace. You can also assign tags directly when saving a model version from the **Save** window. To this end, create a new blank model in the COMSOL Desktop to act as a placeholder for a model tutorial.

1 From the **File** menu, select **New**. Click **Blank Model** ( ).

2 From the **File** menu, select **Save To** ( ).

3 In the **Save** window, choose the demo database in the list of options.

4 In the **Title** field, write `My Tutorial`.

5 In the **Comments** field, write `A new model tutorial`.

The **Tags** section in the **Save** window shows the tags assigned to the model, which in the present case are none.

1 Click the **Add Tag** button ( ).

2 Write `Modeling Tutorials` and press Enter.

The **Modeling Tutorials** tag is added as a *tag pill* in the **Tags** section.

3 Repeat this also for the **In Progress** tag.

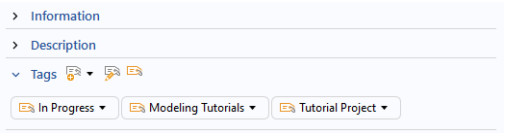
You can also create a new tag in the database from the **Save** window.

1 Click the **New Tag** button ( ) to open the **New Tag** dialog.

2 Write `Tutorial Project` in the **Title** field.

3 On the **Parent tags** tab, select the **Projects** tag in the tag tree.

4 Click **OK**.



With the three tags added to the **Tags** section, you are ready to save the new model in the database. Click **Save** ( ).

Return to the Model Manager workspace. Find and select the new model in the **Model Manager** window — for example by first searching on `my tutorial`. The assigned tags are all shown as tag pills in the **Tags** section in the **Settings** window. You can also verify that you can find the model under these three tags in the Tree View of the **Model Manager** window.

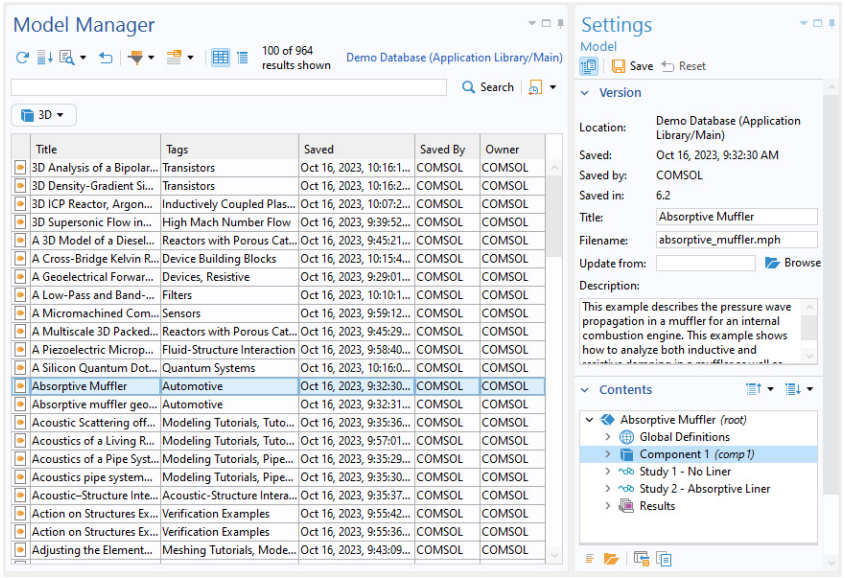
Searching on Model Contents

Applying a **Tag** filter ( ) in the **Model Manager** window is a quick and easy way of finding models and data files in the database. This requires, however, that you have already assigned tags to your items based on some preplanned organizational structure. As you have already seen, you can also apply other types of search filters — including, for example, when an item version was last saved or the user that saved that version. You can even search on the node properties, parameters, settings, and other contents found "deep" inside models.

1. Click the **Reset** button ( ) in the **Model Manager** window to reset the search result. Click the **Table** button ( ) to see the Table View if not already shown.
2. Click the **Add Filter** button ( ) in the toolbar and select **Space Dimension** ( ).

   The **Filter** dialog is opened enabling you to apply a filter on the space dimension of components found inside models.
3. Select the **3D** checkbox in the list of space dimensions.
4. Click **OK** to apply the filter.

The search result includes about 900 models. Select any one of them and verify that the model indeed has a 3D component in the **Contents** section in the **Settings** window.



You can extend your filter to also match models with 2D components.

1 Click the **3D** filter pill in the Model Manager window and select **Edit** ( ).

2 In the **Filter** dialog, select the **2D** checkbox.

3 Click **OK** to apply the updated filter.

The search result count increases with a few hundred models having a 2D component.

The default behavior in the **Filter** dialog is to combine filter options with Boolean `OR`-logic. You can update our filter to use Boolean `AND`-logic instead.

1 Click the space dimension filter and select **Edit** ( ) again.

2 In the **Options** list in the **Filter** dialog, select the **Match all terms** checkbox.

3 Click **OK**.

Select any model in the search result and verify that the model indeed has both a 2D component and a 3D component in the **Contents** section in the **Settings** window.

You can combine the space dimension filter with a separate filter on a physics interface.

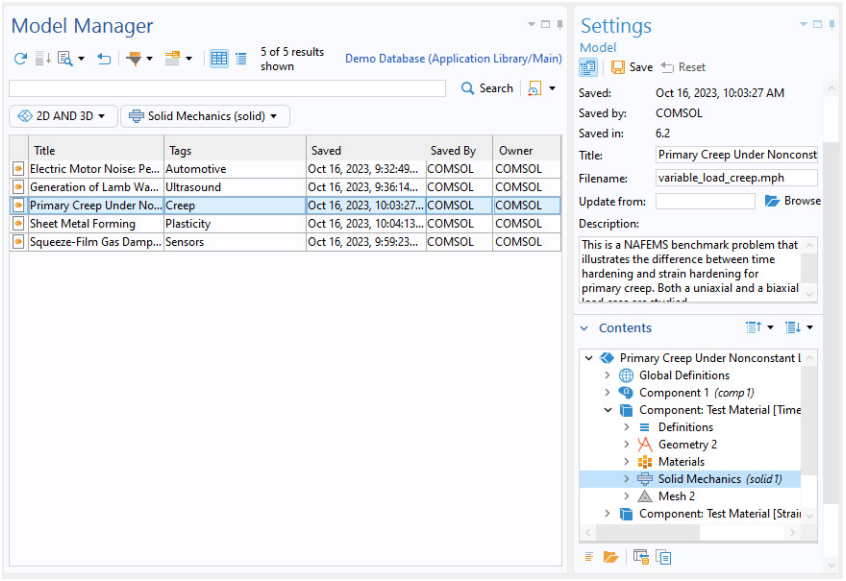
1 Click the **Add Filter** button ( ) and select **Physics** ( ).

2 In the text field above the tree, write `solid mechanics`. Click **Search**.

3 Select the **Structural Mechanics** > **Solid Mechanics (solid)** checkbox.

4 Click **OK**.

The search result only contains three models. All of them use the Solid Mechanics interface in at least one component.



Item and Content Filters

SEARCHING NODE PROPERTIES

You can find models based on the properties of specific nodes in the model tree.

1 Click the **Reset** button (⮌) to remove all filters.

2 Write `wrench` and click **Search**.

3 Double-click the **Stresses and Strains in a Wrench** model (◈) in the search result.

The model version is loaded from the database and opened in the COMSOL Desktop. This is the same model you built in the tutorial Example: Modeling Using Version Control.

4 Right-click **Component 1** ( ) and select **Properties and Comments** ( ) to open the **Properties** window for the node.

5 In the **Comments** field in the **Properties** window, write `A 3D component for the structural integrity of a wrench`.

6 From the **File** menu, select **Save To** ( ).

7 In the **Comments** field in the **Save** window, write `Added a node comment for the component for easier future retrieval`.

8 Click **Save** ( ) to save a new version of the model.

Return to the Model Manager workspace. You can now find this model by searching on the node comment you wrote in the **Properties** window.

1 Click the **Reset** button ( ) to remove the search term.

2 Click the **Add Filter** button ( ) and select **Node Properties** > **Comment** ( ).

3 In the **Filter** dialog, write `component structural integrity wrench` in the **Text** field.

4 Select the **Match all terms** checkbox under **Options**. This step is done to require that all four words appear in the matched node comment.

5 Click **OK**.

The wrench model is indeed the sole search result.

Double-click the component node in the **Contents** section of the **Settings** window for the **Stresses and Strains in a Wrench** model ( ). A **Details** dialog is shown containing node properties and settings for the component. You can find your comment in the **Comment** row of the **Node** table. Click **OK** to close the dialog.

All node and setting values shown in the **Details** dialog for a node in the **Contents** section can be applied as a filter in the Model Manager search functionality. See Content Filters for further details.

Click the **Reset** button ( ) to remove the comment filter.

SEARCHING PARAMETER SETTINGS

The model for the structural integrity of a wrench uses an **Applied force** parameter for the total load on the wrench, in this case set to 150N. You can search for other models using the same parameter and force range.

1 Click the **Add Filter** button ( ) and select **Parameter** ( ).

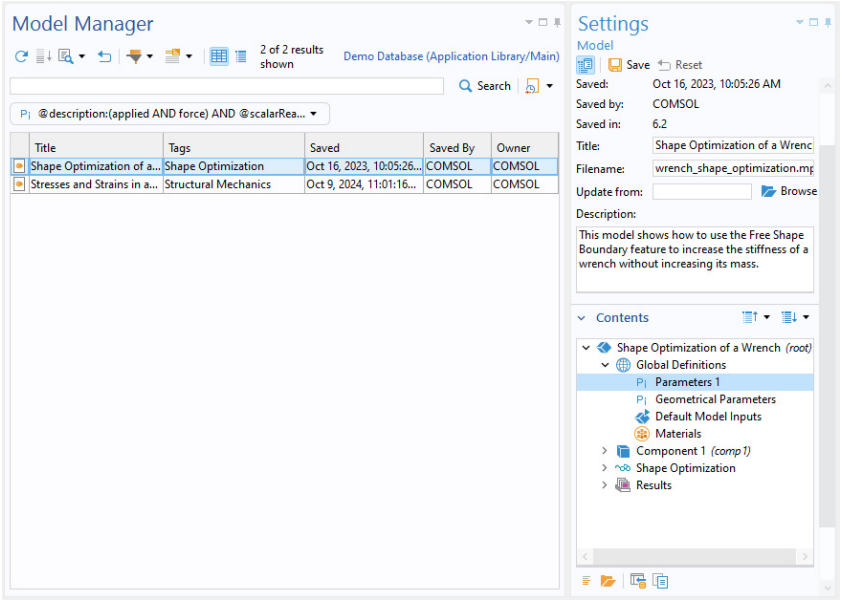
2 In the **Description** field, write `applied force`.

3 Select the **Match all terms** checkbox under **Options**.

4 Write `100` in the **From** field and `200` in the **To** field.

5 Click **OK**.

There are two models in the search result. Select the **Shape Optimization of a Wrench** model ( ) and double-click a parameters node under **Global Definitions** ( ) in the **Contents** section. You will find that there is indeed a row for the **Applied force** with a force value within the searched range in the **Settings** table.



Using the Model Manager Search Syntax

All filters that you can apply via the **Filter** dialog have an equivalent search syntax that you can write directly in the text field in the **Model Manager** window. This syntax is displayed under **Filter query preview** in the **Filter** dialog.

1 Click the **Reset** button ( ) to reset the search result.

2 Click the **Add Filter** button ( ) and select **Item Version Type** ( ).

3 In the **Filter** dialog, select the **Model** checkbox.

   The expression under **Filter query preview** reads `@itemVersionType:model`. The part before the colon identifies the item field to filter on; the part after the colon is the value to filter on.

4 Select the **Application** checkbox.

   The expression now reads `@itemVersionType:(application OR model)`. The expression will match any version that is either a model *or* an application.

5 Select the **Negate match** checkbox under **Options**.

   The expression reads `NOT @itemVersionType:(application OR model)`. The expression will match any version that is *neither* a model nor an application.

6 Click **Cancel** to close the dialog without applying the filter.

You can try out the Model Manager search syntax in the text field.

1 Write `@itemVersionType:fileset` and click **Search**.

   The search result only contains filesets. The part before the colon is case insensitive — writing `@itemversiontype:fileset` yields the same result.

2 Write `@itemversiontype:fileset AND @filetype:asm` and click **Search**.

   The search result contains all filesets containing a data file with a CAD assembly file extension. Spaces are automatically interpreted as Boolean `AND` so you can leave out the `AND` symbol in this case.

3 Write `@itemversiontype:fileset @filetype:asm @title:busbar` and click **Search**.

   The search result only contains CAD assemblies with `busbar` in their title.

The Model Manager search tools can also assist you with search syntax suggestions:

1 Write `@item` and press Ctrl+Space.

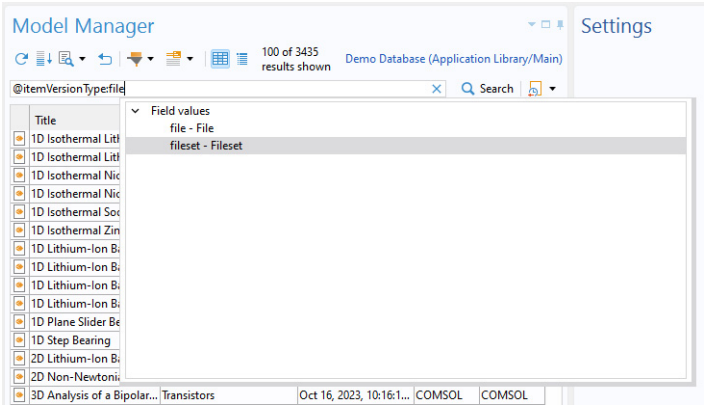
2 In the opened dialog, under **Field expressions**, select the **@itemVersionType: - Item version type** option.

   The begun field identifier is replaced with the complete `@itemVersionType:` syntax.

3 Continue the syntax expression by writing `file` after the colon. Press Ctrl+Space again.

4 In the opened dialog, under **Field values**, select the **fileset - Fileset** option.

The text completes to the proper filter syntax `@itemVersionType:fileset` used to find all filesets.



The search syntax can also be used for searching model contents. You can, for example, find the wrench model with the node comment. Write

```
@node{@comment:(component structural integrity wrench)}
```

and click **Search**. The **Stresses and Strains in a Wrench** model ( ) is again the sole match in the search result.

You may have noticed that these steps did not specify that the comment should appear in a 3D component node. Write

```
@component{@spacedimension:3 @comment:(component structural
integrity wrench)}
```

and click **Search**. The result is the same. Change the space dimension value to `2` and verify that you no longer get any matches.

A full treatise of the capabilities of the Model Manager search syntax is well beyond the scope of this tutorial. You can, for example, write nested search expressions that specify that only models with a specific component space dimension, such that the component itself contains a particular material and physics interface, should match.

The Model Manager Search Syntax

Example: Using Advanced Version Control Tools in the Model Manager

The Model Manager introduces many concepts and terms related to version control that are new to the COMSOL Desktop modeling environment. This includes, for example, versions, drafts, commits, branches, and snapshots. In this section, you will become more acquainted with these concepts and terms by working through an example tutorial. You are strongly recommended to first go through the tutorial Example: Modeling Using Version Control if you have not already done so.

The tutorial makes use of the demo database for Model Manager provided via the COMSOL web site. If you have not already added this database to the COMSOL Desktop, follow the instructions in Downloading the Demo Database for Model Manager.

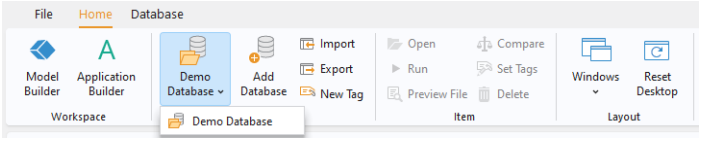
- Using a Draft to Update a Single Model
- Working with Commits
- Using a Branch to Update Many Models

Using a Draft to Update a Single Model

In the tutorial Example: Modeling Using Version Control, you built a model for the structural integrity of a wrench. That model can also be found in the demo database for Model Manager.

1 Click the **Model Manager** ( ) button in the **Workspace** section of the **Home** toolbar in Model Builder to open the Model Manager workspace.

2 In the **Database** section of the **Home** toolbar, select the demo database for Model Manager via the database selector expand button.



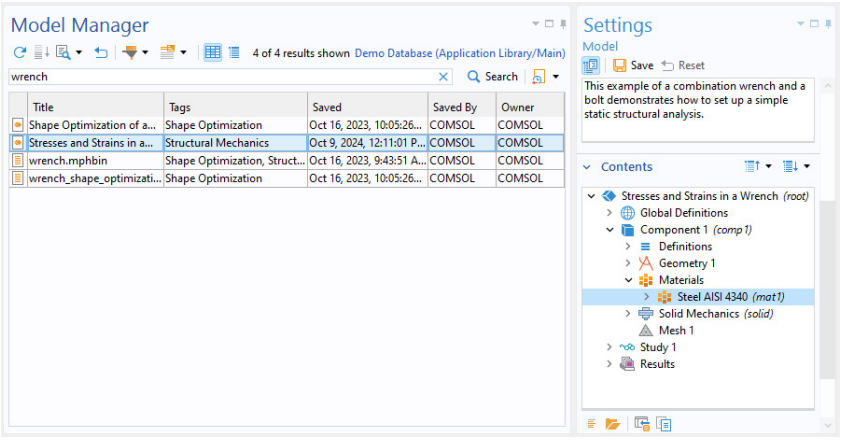
The **Model Manager** window is updated with a search result for the demo database.

3 Write `wrench` in the text field and click **Search**.

4 Select the **Stresses and Strains in a Wrench** model ( ) in the table.

The **Settings** window is updated to show the latest version of the model. If you have already worked through the tutorial Example: Browsing, Organizing, and Searching Models and Data Files, this is the version in which you added a node comment to the component node. Otherwise, it is the first version that was imported from the COMSOL Application Libraries.

5 Expand **Component 1** > **Materials** ( ) in the **Contents** section of the **Settings** window to find that the latest version of the model uses a Structural steel material.



You will make a small update to this model by changing to the low alloy steel material Steel AISI 4340. In the process, you will encounter some new version control tools that go beyond what you learned in the tutorial Example: Browsing, Organizing, and Searching Models and Data Files.

1 Select **Structural steel** ( ) in the **Contents** section and click the **Open Node** button ( ) in the toolbar below the tree.

The model version is loaded from the database and opened in the Model Builder workspace with the **Component 1** > **Materials** > **Structural steel** node ( ) selected.

2 Right-click **Component 1** > **Materials** > **Structural steel** ( ) and select **Delete** ( ). Click **Yes** in the **Confirm Delete** dialog.

The material node is removed from the model tree.

3 Right-click **Component 1** > **Materials** ( ) and select **Add Material from Library** ( ).

4 In the **Add Material** window, click to expand the **Built-In** tree node. Scroll down to find **Steel AISI 4340**, right-click, and select **Add to Component 1**.

5 From the **File** menu, select **Save To** ( ).

6 In the **Comments** field, write `Changed the material to a low alloy steel`.

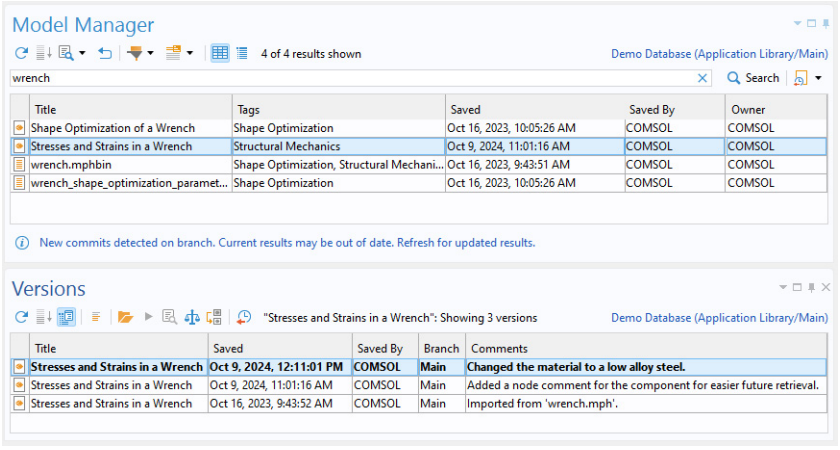
7 Click **Save** ( ).

A new version of the model has been saved to the database. In the tutorial Example: Modeling Using Version Control, you saw that you could view the current version history of the model opened in the COMSOL Desktop from the **Versions** window in the Model Builder workspace. You can also view this version history from the Model Manager workspace.

1 Open the Model Manager workspace.

2 Right-click **Stresses and Strains in a Wrench** in the **Model Manager** window and select **Versions** ( ).

The **Versions** window is opened with the first table row highlighted in bold indicating that the version is currently opened in the COMSOL Desktop. There are three versions in total (or two if you have not gone through the tutorial Example: Browsing, Organizing, and Searching Models and Data Files).



The Versions Window

Imagine now that you regret saving a new version of the model with the changed material — you would rather have kept the structural steel material. One solution is that you open the previous version in the COMSOL Desktop and immediately save

that as a new latest version. Yet a simpler solution is to *restore* the previous version directly in the database.

1 Right-click the second table row from the top in the **Versions** window and select **Restore Version** ( ).

2 In the **Restore Version** dialog, click **OK**.

The selected version is automatically restored as a new latest version in the database.

3 Click **Yes** when asked if you also want to open the restored version.

The selected version is opened in the Model Builder workspace.

An added benefit to using the **Restore Version** ( ) functionality as opposed to manually opening and saving the model in the COMSOL Desktop is that you avoid any model migrations in case the model was originally saved in an older version of the COMSOL Multiphysics software.

Restore Version

You can verify that the opened model indeed uses the old structural steel material by expanding **Component 1** > **Materials** ( ).

You already learned in the tutorial Example: Modeling Using Version Control that a better strategy to making changes to a model is by first creating a draft of the model. A draft is tailor-made for the use case of performing updates to a model without deciding upfront whether the changes are worth keeping.

1 Select **Component 1** > **Materials** > **Structural steel** ( ) and press Del. Click **Yes** to delete the Structural steel material.

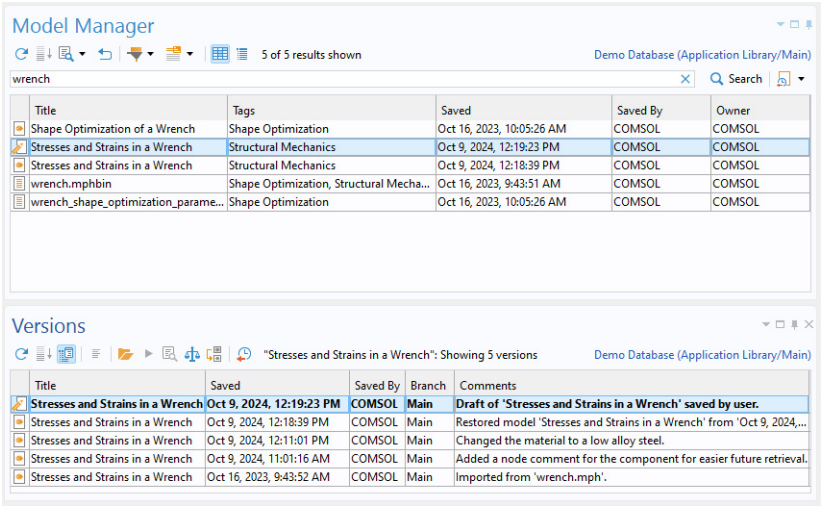
2 In the **Add Material** window, right-click **Steel AISI 4340** and select **Add to Component 1** to add the low alloy steel material once more.

3 Press Ctrl+S to save a draft of the model.

Open the Model Manager workspace. Click the **Refresh** button ( ) in the **Model Manager** window to refresh the table. You will find both the **Stresses and Strains in a Wrench** regular model ( ) and a new **Stresses and Strains in a Wrench** draft model ( ) in the search result.

1 Select the **Stresses and Strains in a Wrench** draft model ( ) in the **Model Manager** window.

The **Versions** window shows the single version of the draft model in the top table row. For convenience, the table also contains the four versions of the regular model that the draft originated from.



2 Select the **Stresses and Strains in a Wrench** regular model ( ) in the **Model Manager** window.

The **Versions** window shows the four versions of the regular model.

You can finish the draft work by saving the draft back as a new version of the regular model.

1 From the **File** menu, select **Save To** ( ).

2 In the **Comments** field, write `Changed the material to a low alloy steel via a draft`.

3 Click **Save** ( ).

The **Stresses and Strains in a Wrench** draft model ( ) is automatically deleted and no longer visible in the **Model Manager** window. Select the **Stresses and Strains in a Wrench** regular model ( ). The **Versions** window shows five versions for the model.

Working with Commits

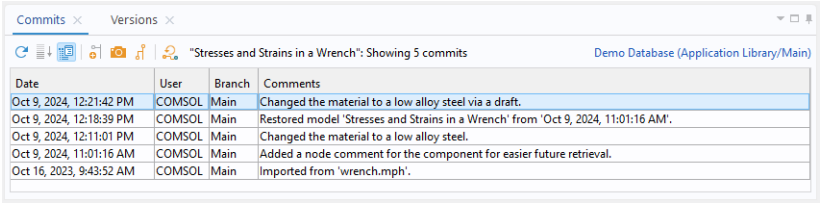
At this point, you may be satisfied with the new low alloy steel material for the wrench. But what if you realize that you wanted to perform further changes to your draft before those changes were saved back to the regular model? You could create a new draft and continue your work, but in the meantime the regular model would contain changes that you are perhaps not yet prepared to share with your coworkers. You could restore the previous version of the regular model using **Restore Version** ( ) from the **Versions** window, but then you would have to redo the work you did in the first draft.

Model Manager solves this issue via the concept of *commits*. Every time you perform changes that involve items in a Model Manager database, all those changes are saved as a *collective whole* in a *commit*. Each commit identifies what was changed, when those changes were performed, and the user that performed the changes. Saving a new version of an item is a special case of such a change. Other examples include deleting items and assigning tags to items.

A useful property of commits is that they can be *reverted*. Reverting a commit means to save a new commit in which the opposite changes to those in the reverted commit are performed.

1 Right-click the **Stresses and Strains in a Wrench** regular model ( ) in the **Model Manager** window and select **Commits** ( ).

The **Commits** window is opened in the Model Manager workspace. The window shows all commits in which the selected model was changed. The commits are sorted in chronological order with the latest commit at the top.



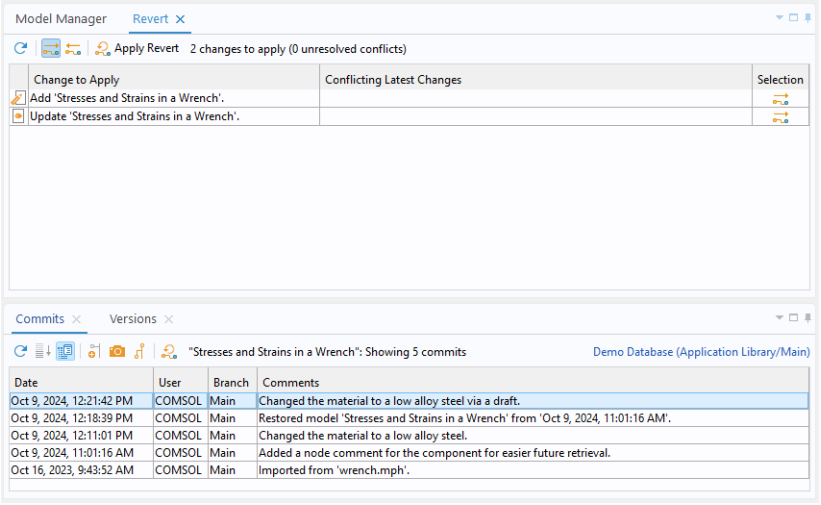
2 Select the top table row.

The **Settings** window shows details for the commit in which the draft model was saved back to the regular model. As seen in the **Changes** section, two item changes were performed in this commit: a new version of the regular model ( ) was saved and the draft model ( ) was deleted.

You will revert the changes made in the latest commit. This will restore the regular model to its previous version *and* recover the deleted draft.

1 Select the top table row and click the **Revert** button ( ) in the toolbar.

The **Revert** window is opened in the Model Manager workspace. The table contains the changes that reverts the selected commit.



2 Click the **Apply Revert** button ( ) in the toolbar.

The **Apply Revert** dialog is opened with a suggested commit comment.

3 Change to `Reverted updates to model made from draft` in the **Comments** field. Click **OK**.

The reverting commit is saved to the database and the **Revert** window is automatically closed.

4 Click the **Refresh** button ( ) in the **Model Manager** window. Both the **Stresses and Strains in a Wrench** regular model ( ) and the **Stresses and Strains in a Wrench** draft model ( ) are shown in the search result.

5 Right-click the regular model and select **Versions** ( ). You can see that the **Versions** window now contains a later version of the regular model than the bold-highlighted

one opened in the COMSOL Desktop. Select the draft model and verify that there are two versions of the draft — these two versions are identical in content.

Reverting

At this point, you could open the **Stresses and Strains in a Wrench** draft model ( ) and continue your draft work. For the conclusion of this part of the tutorial, instead delete the draft again.

1. Right-click the **Stresses and Strains in a Wrench** draft model ( ) in the **Model Manager** window and select **Delete** ( ).

   The **Delete Draft** dialog is opened. The **Item** table contains the draft model, the sole item that will be deleted.

2. Click **OK** to delete the draft.

You should view the list of commits in the **Commits** window as a history of changes made in the database. To see the complete list of all changes going back to when the database was created:

1. Click the **Tree** button ( ) in the **Model Manager** window to switch to the Tree View.
2. Right-click the **Main** root node ( ) in the tree and select **Commits** ( ).

   The **Commits** window shows the 100 most recent commits saved in the demo database. You can select some of the table rows to see which items were involved in each commit in the **Changes** section of the **Settings** window.
3. Click the **Show More** button ( ) a few times to append older commits to the bottom of the table — there are about 4 000 commits in total.
4. Click the **Table** button ( ) in the **Model Manager** window to switch back to the Table View.

The Commits Window

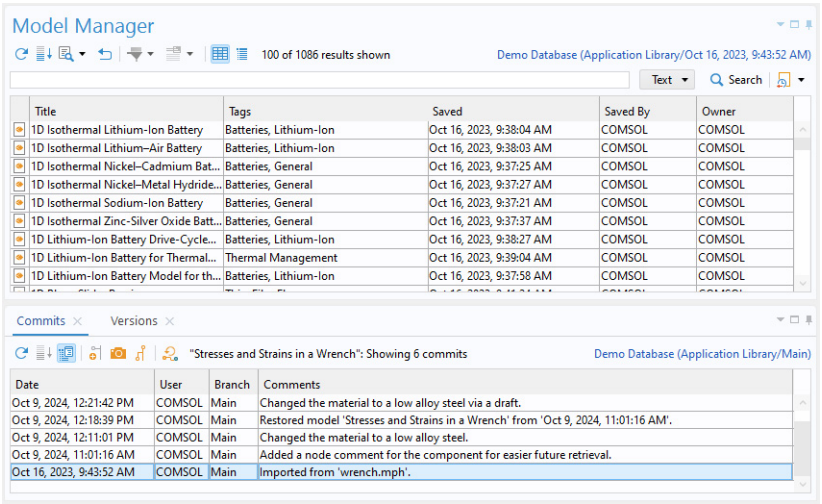
BROWSING AND SEARCHING WITH RESPECT TO COMMITS

Commits serve a secondary purpose in a Model Manager database beyond grouping a collection of changes: a commit can be used to browse and search the particular state of your database at the time when the commit was saved.

1 Right-click the **Stresses and Strains in a Wrench** regular model ( ) and select **Commits** ( ).

2 In the **Commits** window, right-click the last table row at the bottom and select **Search in Commit** ( ).

The **Model Manager** window is updated to show the latest item versions *at the time* the wrench model was first imported in the database. Click the **Reset** button ( ). You will notice that the database had a few thousand less items at that time than when the original import completed.



3 Right-click the table row with the comment **Changed the material to a low alloy steel** in the **Commits** window and select **Search in Commit** ( ).

The **Model Manager** window is updated to show the latest item versions at the time when you replaced the material in the wrench model for the first time in this tutorial.

The most useful commit to search in the **Model Manager** window is the latest one, which is also the default behavior. To switch back to searching with respect to this commit:

1 Click the link button in the upper-right corner in the **Model Manager** window — you will find it above the **Search** button ( ).

The **Select Location** dialog is opened enabling you to select the *commit location* to search with respect to.

2 Select the **Main** branch node ( ) in the tree. The branch serves as an implicit stand-in for the latest commit saved in the database.

You will return to the concept of branches later in this tutorial.

3 Click **OK**.

Locations

You can record a *snapshot* that references a particular commit in case the state of the database holds a special meaning at the point in time when the commit was saved. One such commit for the demo database is the last one in which a file was imported from the COMSOL Application Libraries.

1 Click the link button in the upper-right corner in the Model Manager window.

2 Under **Snapshots** ( ), select the single leaf snapshot node ( ) named after the version number of the COMSOL Multiphysics software used to import the COMSOL Application Libraries.

3 Click **OK**.

The Model Manager window shows the model and data file versions that were the latest versions in the database when the demo database was initially prepared.

Recording Snapshots

Restore the search in the **Model Manager** window by clicking the link button in the upper-right corner in the **Model Manager** window and selecting the **Main** branch node ( ) in the **Select Location** dialog. You can also restore the search by selecting the demo database via the database selector expand button in the **Database** section of the **Home** toolbar.

Using a Branch to Update Many Models

In the previous example, you used a draft to replace a material in a model. This enabled you to do the change in isolation and at your own pace — all while leaving the main model untouched in the interim. This strategy is the recommended approach when working on changes to a *single* model, but how should you proceed if there is more than one model you need to update? You could create drafts of all the models, do your updates to each draft, and then save each draft back to its original model. This strategy works in principle but is rather tedious and risks leaving the models in a half-finished state while your work is progressing. If you later want to undo your changes, there would potentially be a lot of cleaning up to do in the form of reverting commits.

As a concrete example, say you have a few models that all share the same CAD geometry. Imagine that your CAD engineer colleague has sent you a new geometry that you would like to update your models with. If the models are stored on a file system shared with other simulation engineers, you perhaps begin by copying all the MPH-files to a new folder on the file system. You replace the geometry in the models using the new CAD data file, test out your changes, and then overwrite the original MPH-files when you are satisfied with the update.

Model Manager enables you to solve such a multi-update problem for models and data files using the concept of *branches*. A branch is another name for the sequence, or *history*, of commits shown in the **Commits** window. You can create a new branch from the initial **Main** branch, thereby creating an *alternative history* of commits. You can think of the commits on the **Main** branch as the trunk of a growing tree, with the oldest commit found at the base of the tree. The alternative commit sequences are tree branches growing out from this tree trunk. When you save commits on such a new branch, any changes made to models and data files are invisible to users working on the **Main** branch — thereby enabling you to do your changes in isolation until you are ready to share your work by *merging* the changes to the **Main** branch.

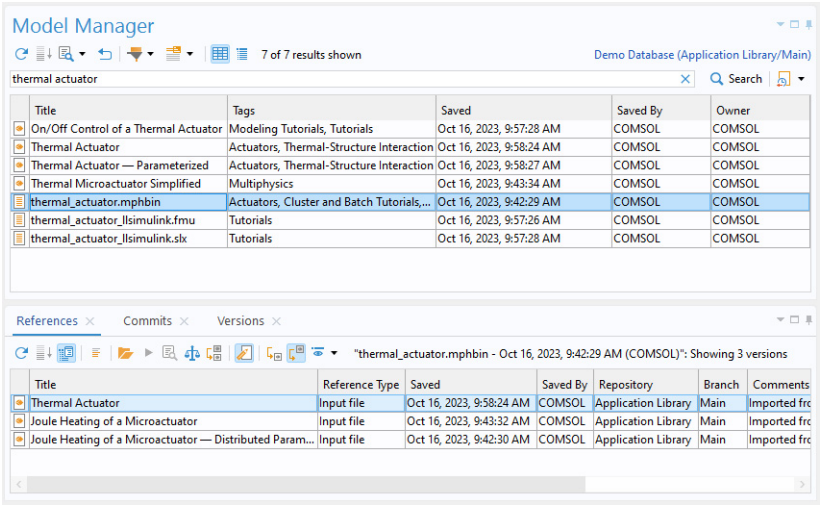
CREATING THE NEW BRANCH

As an example of working simultaneously with many model changes, you will replace a CAD geometry used as an input file by a few models in the demo database for Model Manager.

1 Write `thermal actuator` in the text field in the **Model Manager** window and click **Search**.

2 Right-click the **thermal_actuator.mphbin** data file ( ) and select **References** ( ).

The **References** window is opened in the Model Manager workspace showing three models using the selected data file as an input file. The selected file is a CAD geometry stored in the COMSOL native CAD format `mphbin`, which the models all reference via a geometry **Import** node ( ).



You will assign a tag to the three models and the data file to collect them together in the database.

1 Press Ctrl and select the **thermal_actuator.mphbin** data file ( ) and the **Thermal actuator** model ( ) in the **Model Manager** window.

2 In the **Home** toolbar, in the **Database** section, click the **New Tag** button ( ).

3 In the **Title** field, write `CAD Update Project`. Leave the **Add to selected models and files** checkbox selected.

4 Click **OK**.

A new tag is created in the database. The tag is also assigned to the CAD data file and one of the models referencing the file.

1 Write `joule heating` and click **Search**.

2 Press Ctrl and select the **Joule Heating of a Microactuator** model ( ) and the **Joule Heating of a Microactuator — Distributed Parameter Version** model ( ).

3 In the **Home** toolbar, in the **Database** section, click the **Set Tags** button ( ).

4 In the **Set Tags** dialog, select the **CAD Update Project** tag tree node ( ).

5 Click **OK**.

You are now ready to create a new branch in your database that contains the three models and the data file.

1 Write `cad update project` and click **Search**.

2 Press Ctrl and select the three models and the **thermal_actuator.mphbin** data file ( ) in the **Model Manager** window.

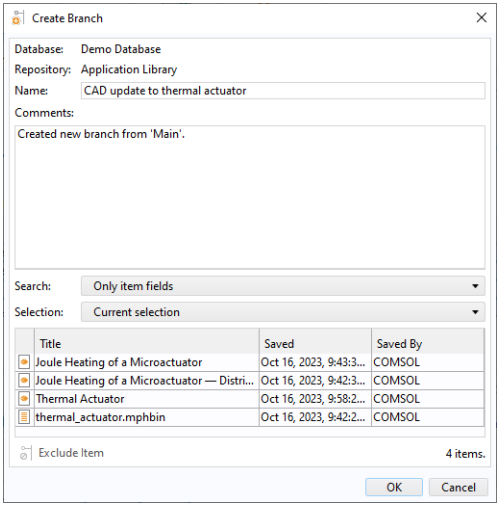
3 Right-click any one of the selected items and select **Branch** ( ).

The **Create Branch** dialog is opened with the three models and the single data file listed in a table at the bottom of the dialog.

4 In the **Name** field, write `CAD update to thermal actuator`.

5 In the **Search** list, select `Only item fields`. This reduces the disk space used by indexed search data at the expense of less filter capabilities on the new branch. This is a reasonable tradeoff as the new branch will only contain four, easy-to-find, items.

6 Click **OK**.



You could have also right-clicked the **CAD Update Project** tag tree node ( ) in the Tree View of the **Model Manager** window and selected **Branch** ( ).

Branching

A branch containing the three models and the CAD geometry data file has been created in the database. Click the **Reset** button ( ) in the Model Manager window — you will indeed only see four items in the search result.

An initial commit was saved to the database when the **CAD update to thermal actuator** branch was created. You can see this commit as the top table row in the **Commits** window. The table also includes the commits on the **Main** branch up to the point where the new branch was created.

1. Click the link button in the upper-right corner in the **Model Manager** window — you will find it above the **Search** button ( ).
2. In the **Select Location** dialog, select the **Main** branch node ( ) in the tree.
3. Click **OK**.

The **Model Manager** window is updated to show the latest versions of items on the **Main** branch. Also, the **Commits** window only shows the commits on the **Main** branch, not on the **CAD update to thermal actuator** branch. To return to your new branch, click the link button in the **Model Manager** window, select the **CAD update to thermal actuator** branch node ( ) in the tree, and click **OK**.

Select the top table row in the **Commits** window. The **Settings** window shows the initial commit on the **CAD update to thermal actuator** branch. You will find that the **Changes** section is empty as no items were changed when the new branch was created — the latest item versions and the tag assignments on the new branch are, at least initially, *identical* to those on the **Main** branch. You can compare this with the file system analogy of copying MPH-files to a new location on the file system.

While the **CAD update to thermal actuator** branch contains copies of the four versions selected in the **Main** branch, the new branch uses little additional disk space for these copies thanks to data deduplication in Model Manager.

UPDATING THE CAD INPUT FILE

You will update the CAD data file in the database with a file found in the COMSOL installation folder.

1 Select the **thermal_actuator.mphbin** data file ( ) in the **Model Manager** window.

2 In the **Description** field in the **Settings** window, write `CAD geometry for a thermal actuator`.

3 In the **Contents** section in the **Settings** window, select **thermal_actuator.mphbin**.

4 Click the **Replace** button ( ) and locate the file `thermal_actuator.mphbin` in the application library folder of the COMSOL installation folder. Its default location in Windows® is

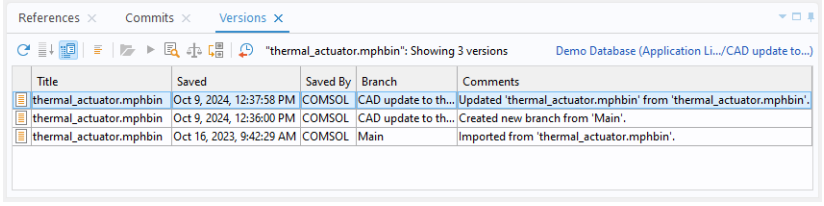
```
C:\Program Files\COMSOL\COMSOL63\Multiphysics\applications\
COMSOL_Multiphysics\Multiphysics\thermal_actuator.mphbin
```

Double-click to replace, or click **Open**.

5 Click **Save** ( ) in the **Settings** window.

6 Click **OK** in the **Save File** dialog.

The CAD data file is now updated with a new version on the **CAD update to thermal actuator** branch. Right-click the **thermal_actuator.mphbin** data file ( ) and select **Versions** ( ) to open the **Versions** window for the file. The window shows all versions of the file *with respect to* the **CAD update to thermal actuator** branch. The top table row is the latest version on the branch, the middle row is an identical copy of the version found on the **Main** branch. For convenience, the original version on the **Main** branch is also appended at the bottom of the table. This last version is also the only version that users currently browsing the **Main** branch sees.



UPDATING THE MODELS

You will continue your work on the branch by using the new CAD geometry in your models.

1 Double-click the **Thermal actuator** model ( ) in the **Model Manager** window.

The model is opened in the Model Builder workspace.

2 In the **Model Builder** window, select **Geometry 1** > **Import** ( ).

3 In the **Settings** window, click the expand button next to **Browse** ( ) and select **Browse From** ( ).

4 In the **Select File** window, select the demo database in the list of options.

5 Click the **Main** link button — you will find it above the **Search** button ( ).

6 In the **Select Location** dialog, select the **CAD update to thermal actuator** branch node ( ). Click **OK**.

7 Double-click the **thermal_actuator.mphbin** data file ( ) to select it as a new input file.

The model now references the updated CAD geometry in the new branch. Click the **Import** button ( ) to import the geometry into the model. Finish by saving a new version of the model.

1 From the **File** menu, select **Save To** ( ).

   The **Save** window is opened for the demo database with the new branch automatically selected in the **Location** field.

2 In the **Comments** field, write `Updated geometry using new CAD data`.

3 Click **Save** ( ).

A new version of the model is saved to the **CAD update to thermal actuator** branch. Much like the updated data file, this model version is not visible on the **Main** branch.

Open the Model Manager workspace and repeat the update for the remaining two models, **Joule Heating of a Microactuator** and **Joule Heating of a Microactuator — Distributed Parameter Version**. In the **Select File** window, you may use the **Recent** ( ) list option to quickly select the **thermal_actuator.mphbin** data file ( ).

MERGING THE CHANGES

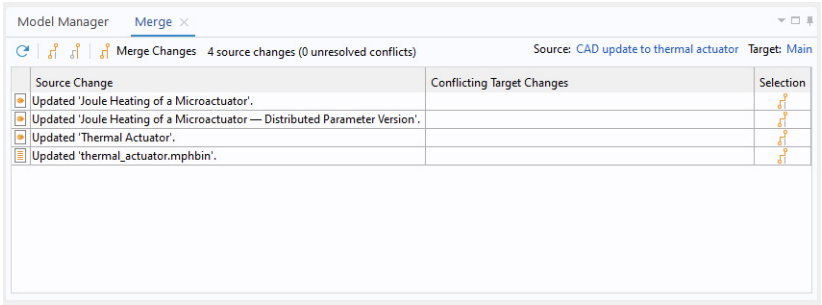
With the model updates completed, the final step is to *merge* the changes from the **CAD update to thermal actuator** branch to the **Main** branch. This will save the latest versions of items on the *source* branch as new latest versions on the *target* branch.

1 Open the Model Manager workspace.

2 In the **Database** toolbar, in the **Database** section, click the **Databases** button ( ) to open the **Databases** window.

3 Right-click the **CAD update to thermal actuator** branch node ( ) and select **Merge** ( ).

The **Merge** window is opened in the Model Manager workspace. The window lists all changes that will be merged from the source branch to the target branch — in this case there are four changes corresponding to the updated items.



4 Click the **Merge Changes** button ( ) in the toolbar.

5 In the **Merge** dialog, click **OK**.

The **Merge** window is closed and the four changes are saved as a single commit to the **Main** branch.

6 Select the **Main** branch node ( ) in the **Databases** window.

The **Model Manager** window is automatically updated to show the search result for **Main** branch node.

Merging

The three models and the data file are now updated on the **Main** branch. At this point, you could continue your work on the **CAD update to thermal actuator** branch, save versions of your models, and merge any new changes to the **Main** branch.

Opting instead to delete the branch.

1 Right-click the **CAD update to thermal actuator** branch node ( ) and select **Delete** ( ).

2 Click **Yes** in the **Delete Branch** dialog.

This concludes this tutorial on the version control tools available in the Model Manager. While creating branches is useful when you want to make sweeping changes

involving many models and data files, it comes at the cost of added complexity. You should always consider if a draft can solve your particular version control problem before you create a new branch.

6

Model Manager API

This chapter shows how you can access Model Manager databases via the Model Manager application programming interface (API) for use with the Java® programming language. The Model Manager API can be used in the Application Builder's Method Editor, in the Java Shell window, in model files for Java®, and from the LiveLink™ for MATLAB® interface. If you are using the Model Manager API from the Application Builder, see also the *Application Programming Guide* for useful information when creating methods for applications. See the *Programming Reference Manual* to learn how to use the COMSOL API to access models through the *model object*.

This chapter is organized into sections that cover various parts of the Model Manager API. Each part typically consists of a short introduction together with one or more examples formulated as a programming task and a code sample that solves that specific task.

In this chapter:

Getting Started with the API

The Model Manager API for use with the Java® programming language enables you to perform various tasks in a Model Manager database by writing and running custom code. Many of the Model Manager Tools available from the COMSOL Desktop have direct counterparts in the form of methods in the API. There is even functionality in the API that enables you to perform tasks that are not possible in the COMSOL Desktop. You can use the Model Manager API in the Application Builder's Method Editor, in the Java Shell window, in model files for Java®, and from the LiveLink™ for MATLAB® interface.

	The Model Manager API is *not* available when running an application via COMSOL Server™ or in a standalone application compiled using COMSOL Compiler™.

	If the **Enforce security restrictions** checkbox is selected on the **Security>Methods and Java Libraries** page in the **Preferences** window, you must select **Allow access to Model Manager databases** to allow the use of the Model Manager API.

	The Java documentation for all types and methods in the Model Manager API is available at https://doc.comsol.com — see **COMSOL API for use with Java®>Java Documentation** at the bottom of the page.

	All code samples in this and other sections are written in a method editor window in the Application Builder workspace. You may need to adapt the code if used in another context.

In this section:

- Accessing the Model Manager API
- Connecting to Model Manager Databases
- Navigating a Model Manager Database
- Reading Settings

- Creating and Updating Database Objects
- Advanced Database Operations Using Parameter Objects
- Querying Database Objects

Accessing the Model Manager API

You access the Model Manager API using the utility class `DatabaseApiUtil` and its static method `api`. This method returns a `DatabaseApi` instance that is used, for example, to list all databases configured in the COMSOL Desktop, connect to a configured database, or to directly obtain the model version corresponding to the model opened in the COMSOL Desktop.

The Model Manager API is defined in the `com.comsol.api.database` package and its subpackages. You will find the complete Javadoc for all types and methods in these packages on the **COMSOL API for use with Java®>Java Documentation** pages at the COMSOL documentation website https://doc.comsol.com.

EXAMPLES

Get the model version that the model opened in the COMSOL Desktop was loaded from. The opened model is identified by its model tag string, which should not be confused with a tag item in a Model Manager database.

```
String tag = model.tag();
DatabaseApi api = DatabaseApiUtil.api();
ModelItemVersion version = api.modelVersionByModelTag(tag);
```

A model version contains, for example, methods for reading basic Model Settings.

Items and Versions

Get the model version corresponding to a model location URI string. Such a string can be obtained, for example, by selecting **Copy Location** () in the context menu for a model version in the Model Manager workspace.

```
DatabaseApi api = DatabaseApiUtil.api();

// The dots represent further characters specific to the selected
// model version.
```

```
String location = "dbmodel:///?..."
ModelItemVersion v = api.modelVersionByLocationUri(location);
```

Get the file version corresponding to a file location URI string. Such a string can be obtained, for example, by selecting **Copy Location** () in the context menu for a file version in the Model Manager workspace. You can also obtain it by selecting **Copy Location** () in the **Location** menu() for a model tree node with a **Filename** field in the Model Builder or Application Builder workspaces.

```
DatabaseApi api = DatabaseApiUtil.api();

// The dots represent further characters specific to the selected
// file version.
String location = "dbfile:///my_interpolation_function.txt?...";
FileItemVersion version = api.fileVersionByLocationUri(location);
```

A file version contains, for example, methods for reading basic File Settings, including listing all its file resources.

Copying Model and File Locations

Write the label of all configured databases to the **Debug Log** window in the Application Builder workspace using the `debugLog` method from the Method Editor's built-in method library.

```
DatabaseApi api = DatabaseApiUtil.api();

QueryDatabaseConfigurationResultStream configurations = api
  .queryDatabaseConfigurations();

for (QueryDatabaseConfigurationResult config : configurations) {
  debugLog(config.label());
}
```

Configurations for both local databases and server databases accessed via a Model Manager server are included in the for-loop.

Creating and Running Methods in Models in the *COMSOL Multiphysics Reference Manual*

Save a new version of the model opened in the COMSOL Desktop using its model tag identifier and without providing a commit comment.

```
String tag = model.tag();
DatabaseApi api = DatabaseApiUtil.api();
ModelItemVersion version = api.saveRegularModel(tag, null);
```

Basic Loading and Saving of Models

Connecting to Model Manager Databases

You can connect to a Model Manager database, for example, by first saving an alias for the database via the **Alias** field in the **Settings** window and then using this alias as an identifier in the API.

Database Settings

Only databases that have already been added to the COMSOL Desktop are available from the API. For a Model Manager server database, you must also have selected the **Remember Password** checkbox — either when connecting to the server database via the **Add Database** window or from the **Settings** window for the corresponding database configuration.

Adding Databases

EXAMPLES

Connect to the database with alias `my-database`.

```
DatabaseApi api = DatabaseApiUtil.api();
Database database = api.databaseByAlias("my-database");
```

Connect to databases by looping over all database configurations.

```
DatabaseApi api = DatabaseApiUtil.api();

QueryDatabaseConfigurationResultStream configurations = api
  .queryDatabaseConfigurations();

for (QueryDatabaseConfigurationResult config: configurations) {
```

```
    DatabaseConfigurationKey configurationKey = config
      .databaseConfigurationKey();
    Database database = api
      .databaseByConfigurationKey(configurationKey)
}
```

An exception is thrown if Model Manager is unable to connect to the database. This includes, for example, if the password used to login to a Model Manager server is not available for Model Manager.

Navigating a Model Manager Database

Starting from a connected database, you can navigate to repositories, branches, items, and other objects stored in the database using the objects' *identifying keys*. Some common objects are also available via shortcut methods. The keys themselves are typically obtained from individual elements in query and search results — see Querying Database Objects — or from other database objects.

In the remainder of this section, it is assumed that all code example snippets are preceded by the statements:

```
DatabaseApi api = DatabaseApiUtil.api();
Database database = api.databaseByAlias(<alias>);
```

with `<alias>` some alias for a Model Manager database.

Connecting to Model Manager Databases

EXAMPLES

Get the latest versions of 100 models and files in the default repository and in the default branch. This is the same search result you would obtain in the **Model Manager** window when not writing a search term in the search field or applying any search filters.

```
Repository repository = database.defaultRepository();
Branch branch = repository.defaultBranch();
SearchBranchItemResult[] result = branch
  .searchItems("")
  .toArray(100);
```

If you want to access the default branch in some other repository, for example, you will first need its repository key. One way, which is rather cumbersome, is to pairwise write

the repository keys and repository names of all available repositories in the database to the **Debug Log** window in the Application Builder workspace.

```
for (QueryRepositoryResult r : database.queryRepositories()) {
  debugLog(r.name());
  debugLog(r.repositoryKey().value());
}
```

Select the key string of the desired repository in the window and press CTRL+C, or right-click and select **Copy**. Paste the key string into your code by pressing CTRL+V, or right-click and select **Paste**.

```
RepositoryKey key = RepositoryKey.of(<keyString>);

Repository repository = database.repositoryByKey(key);
Branch branch = repository.defaultBranch();

SearchBranchItemResult[] result = branch
  .searchItems("")
  .toArray(100);
```

If you already have a reference to some other database object in the repository, for example a model version, you can often navigate to the repository from that object directly.

```
// The modelLocationUri string could have been obtained, for
// example, using Copy Location in the Model Manager workspace.
ModelItemVersion version =
  api.modelVersionByLocationUri(modelLocationUri);

Repository repository = version.repository();
```

The same is true for other objects related to the model version.

```
// The model item that the version belongs to.
ModelItem item = version.item();

// The branch that the version was saved in.
Branch branch = version.branch();

// The latest version of the item on the branch.
BranchModelItem latest = version.branchItem();

// The commit that the version was saved in.
Commit commit = version.commit();

// The database that the version was saved in.
Database database = version.database();
```

You may think of the model version as providing a stable "anchor" in your code from which you can quickly navigate to other objects. This will hold true as long as the model version is not permanently deleted in the Model Manager database.

Search for the latest version of a model item in the default repository and in the default branch. Query for the most recently saved version in the database — in any repository and in any branch — of the corresponding model item. Write the names of the corresponding repository and branch the version belongs to in the **Debug Log** window.

```
Branch branch = database.defaultRepository().defaultBranch();

SearchBranchItemResult searchResult = branch
  .searchItems("My Model")
  .single();
ItemKey itemKey = searchResult.itemKey();
ModelItem modelItem = database.modelByKey(itemKey);

QueryItemVersionResult queryResult = modelItem
  .queryVersions()
  .firstOrNull();
ItemVersionKey versionKey = queryResult.itemVersionKey();
ModelItemVersion version = database.modelVersionByKey(versionKey);

debugLog(version.repository().name());
debugLog(version.branch().name());
```

Querying Database Objects

Reading Settings

You can read individual settings values for database objects corresponding to the fields shown in the **Settings** window in the Model Manager workspace. You can also retrieve all settings values via an optimized `get` method on a database object — thereby reducing network latency overhead when, for example, reading settings values from a Model Manager server database.

EXAMPLES

Get settings values for the model version currently opened in the COMSOL Desktop and write them to the **Debug Log** window.

```
String tag = model.tag();
ModelItemVersion version = api.modelVersionByModelTag(tag);
```

```
debugLog(version.title());
debugLog(version.filename());
debugLog(version.description());
debugLog(formattedDateTime(version.saved()));
debugLog(version.savedBy());
debugLog(version.savedIn());
debugLog(version.owner());

// The settings values can also be retrieved collectively from
// the database via the optimized get method.
GetModelItemVersionResult result = version.get();

debugLog(result.title());
debugLog(result.filename());
// ...
```

The `formattedDateTime` method from the Method Editor's built-in method library converts the UNIX epoch milliseconds returned by `saved` to a read-friendly date and time string.

Get settings values for a file version referenced as an output file by a table feature in the model object.

```
ExportFeature f = model.result().export("table1");
String fileLocationUri = f.getString("filename");

FileItemVersion version = api
  .fileVersionByLocationUri(fileLocationUri);

debugLog(version.title());
debugLog(version.description());
debugLog(formattedDateTime(version.saved()));
debugLog(version.savedBy());
debugLog(version.savedIn());
debugLog(version.owner());

FileResourceResult fileResource = version
  .firstFileResourceOrNull();

debugLog(fileResource.path().filename());
debugLog(fileResource.size());
debugLog(fileResource.lastModified());
```

The previous code snippet assumed that the file version had a single file resource, that is, it is not a fileset.

Creating and Updating Database Objects

You can create new, or update existing, database objects via methods in the Model Manager API. Such methods typically come in pairs: one simple variant that takes a few mandatory arguments and that relies on defaults for other settings, and one advanced variant that enables you to fully specify all settings — see Advanced Database Operations Using Parameter Objects for further details.

EXAMPLES

Create a new repository in the database with name `My Repository`. The second argument is the comment for the first commit automatically saved to the new repository's default branch.

```
Repository repository = database
  .createRepository("My Repository", "Created a new repository.");
```

The comment can always be skipped when saving a commit — in the previous example done by passing null for the second argument.

Create a new branch with name `My Branch` from the default branch. The third argument is the comment for the first commit automatically saved to the new branch.

```
BranchKey branchKey = repository.defaultBranch().branchKey();
Branch branch = repository
  .createBranch("My Branch", branchKey, "Created a new branch.");
```

Create a tag in the new branch with title `My Tag`. Assign the new tag to a model on the branch.

```
TagItemVersion version = branch
  .saveNewTag("My Tag", "Created a new tag.");

// The modelItemKey variable identifying the model item is
// assumed to be initialized elsewhere.
branch
  .modelByKey(modelItemKey)
  .assignTag(version.itemKey(), "Assigned a tag to the model.");
```

Update the title of the tag.

```
BranchTagItem branchTagItem = version.branchItem();
```

```
TagItemVersion secondVersion = branchTagItem
  .updateTitle("My Renamed Tag", "Renamed the newly created tag.");
```

Tags are version controlled in exactly the same way as models and files. Updating the tag is done on a specific branch and results in a new version of the tag.

Advanced Database Operations Using Parameter Objects

The Model Manager API enables you to take further control over database operations using so-called *parameter objects*. These are objects that specify *all* available input options governing the corresponding database operation. This is useful, for example, when you want to stray from default settings when creating or updating database objects, or when you need to perform other advanced operations that would be unwieldy to code by providing a long list of disparate arguments to an API method.

The parameter objects are created using a utility returned from the method `param` on `DatabaseApiUtil`. For each advanced database operation in the Model Manager API, there is a corresponding method on this utility that returns a so-called *parameter generator* for the parameter object. This generator has a dual purpose: it is used for specifying all input options and for being passed as argument to the corresponding API method as the parameter object itself.

Accessing the Model Manager API

SYNTAX

```
DatabaseApiParamGenerators param = DatabaseApiUtil.param();

<method>ParamGenerator generator = param.for<method>();

<method>ParamGenerator parameterObject = generator
  .with<parameter-option1>(<parameter-value1>)
  .with<parameter-option2>(<parameter-value2>)
  .with<parameter-option3>(<parameter-value3>);
```

with `<method>` corresponding to a method in the Model Manager API, `<parameter-optionX>` the name of a parameter option, and `<parameter-valueX>` the value for the parameter option. The parameter generator interface extends the parameter object

interface, which means that `parameterObject` can be passed directly to a method in the API expecting the latter interface.

EXAMPLES

Create a new repository with a custom name for the repository's initial, default, branch.

```
CreateRepositoryParamGenerator p = DatabaseApiUtil.param()
  .forCreateRepository()
  .withName("My Repository")
  .withDefaultBranchName("My Main Branch")
  .withCommitComment("Created a new repository.");

Repository repository = database.createRepository(p);
```

Create a new branch from the specific commit that a model version was saved in.

```
// The modelLocationUri string could have been obtained, for
// example, using Copy Location in the Model Manager workspace.
ModelItemVersion version = api
  .modelVersionByLocationUri(modelLocationUri);

CreateBranchParamGenerator p = DatabaseApiUtil.param()
  .forCreateBranch()
  .withName("My Branch")
  .withSourceCommitKey(version.commitKey());

Branch branch = version.repository().createBranch(p);
```

The model version itself is used as an anchor to access the repository in which to create the new branch — see also Navigating a Model Manager Database.

Update settings for the latest version of a model on a branch.

```
BranchModelItem branchModelItem = version.branchItem();
UpdateModelItemParamGenerator p = DatabaseApiUtil.param()
  .forUpdateModel()
  .withDescription("My model description.")
  .withFilename("my_model.mph");

// Passing null for the optional commit comment is allowed.
ModelItemVersion newVersion = branchModelItem.update(p, null);
```

Querying Database Objects

You can read data values for multiple database objects using *query* methods in the Model Manager API. You have already seen examples of this for database configurations and repositories earlier in the section. As the number of objects in a

database can potentially be large, these query methods return so-called *result streams* from which individual elements can be retrieved *on demand*. You may, for example, build an application in the Application Builder that shows the latest couple of versions of a model in a suitable form object. In that case, it would typically be wasteful to first fetch data for all versions from the database in the app and then immediately discard all but a few to show in the form object.

All result streams in the Model Manager API extend `DatabaseApiResultStream`. You can read the first element from the result stream, read a fixed number of elements from the result stream as a list or an array, or you can traverse the elements in the result stream using loop constructs.

The elements in the result stream only contain data values — they are not a representation of the database objects themselves. You may, for example, obtain a result stream whose elements correspond to a *mix* of model versions and file versions when querying items in the database. You can always obtain the corresponding database object from an element using its identifying key — see also Navigating a Model Manager Database.

EXAMPLES

Get the first branch created in a repository by querying all branches sorted on created date and then using the `firstOrNull` method on the result stream.

```
QueryBranchesParamGenerator p = DatabaseApiUtil.param()
  .forQueryBranches()
  .withSortByCreated();
QueryBranchResultStream resultStream = repository
  .queryBranches(p);
QueryBranchResult result = resultStream.firstOrNull();
Branch branch = repository.branchByKey(result.branchKey());
```

When you expect that a query should only return a single element — anything else to be considered an error — you can use the `single` method on the result stream. Find the latest version of a model with a specific filename, with the filename expected to be unique for models on the branch.

```
Branch branch = database.defaultRepository().defaultBranch();

// Filters can be specified using the Model Manager search syntax.
SearchBranchItemResultStream resultStream = branch
  .searchItems("@filename:my_model.mph");

// The single-method throws if the result stream does not contain
// exactly one element.
SearchBranchItemResult r = resultStream.single();
```

```
BranchModelItem branchModelItem = branch.modelByKey(r.itemKey());
```

Searching

You can read any number of elements from the stream into a list or an array. Read data values for the ten most recently saved versions of a file item in a branch.

```
// The itemKey variable is assumed to be initialized elsewhere.
BranchFileItem branchFileItem = branch.fileByKey(itemKey);
QueryItemVersionHistoryResultStream resultStream = branchFileItem
  .queryVersionHistory();

// You can use either a list or an array depending on preference.
java.util.List<QueryItemVersionHistoryResult> list = resultStream
  .toList(10);
QueryItemVersionHistoryResult[] array = resultStream
  .toArray(10);
```

When you need to traverse a large number of database objects from a result stream, it is better to use a *for-each loop* statement. Save the latest versions of all models from one *source* database to another *target* database.

```
SearchBranchItemResultStream resultStream = sourceDatabase
  .defaultRepository()
  .defaultBranch()
  .searchItems("@itemType:model");

for (SearchBranchItemResult r : resultStream) {
  SaveModelItemParamGenerator p = DatabaseApiUtil.param()
    .forSaveModel()
    .withSourceDatabaseKey(database1.databaseKey())
    .withSourceItemVersionKey(r.itemVersionKey());

  ModelItemVersion version = targetDatabase
    .defaultRepository()
    .defaultBranch()
    .saveModel(p, null);
}
```

You can also use the `iterator` method on `DatabaseApiResultStream` to take explicit control of the element traversal. With an iterator, you can, for example, fetch the next 100 versions when building an application user interface that shows a version history with *show more* functionality.

```
// The branch and itemKey variables are initialized elsewhere.
```

```
QueryItemVersionHistoryResultStream resultStream = branch
  .modelByKey(itemKey)
  .queryVersionHistory();

// An iterator obtained when initializing some form object that
// will show the version history.
java.util.Iterator<QueryItemVersionHistoryResult> iterator =
  resultStream.iterator();
```

The iterator may be used in method code that runs when, for example, clicking a **Show More** button in the application user interface.

```
int i = 0;
while (iterator.hasNext() && i < 100) {
  QueryItemVersionHistoryResult result = iterator.next();
  i++;
  // Add data to the form object...
}
```

You can also use the `stream` method on `DatabaseApiResultStream` if you prefer to work directly with the Java® Stream API.

Version Control Management of Models and Files

The Model Manager API supports the same version control functionality for items that is available using the Model Manager tools in the COMSOL Desktop. By combining options on parameter objects, it is even possible to perform advanced operations that are not easily done via the COMSOL Desktop — see Advanced Database Operations Using Parameter Objects. This includes, for example, importing files on the file system as new versions of existing items, importing item versions from another database, or updating an item from a version of another, possibly unrelated, item.

In this section:

Items and Versions

When working with the Model Manager API, it is beneficial to internalize the distinction between items and the items' versions. An *item* — that is, a tag, model, or file — can have one or more *item versions* saved in different branches, repositories, or even databases. The item is uniquely identified by its `ItemKey`. This key is preserved by default when saving an item from one database to another database. A version of an item belongs to a specific branch, repository, and database. Each item version is uniquely identified by its `ItemVersionKey`.

The items and their versions are represented by different interfaces in the Model Manager API. For a model (with analogous interfaces for tags and files):

- `ModelItem` — the model item itself. May be used, for example, to query for *all* versions of the model in the database, irrespective of repository and branch.
- `ModelItemVersion` — a specific version of the model item. Can be used to read data values and other information related to the version, but cannot be used to save a new version of the model item.
- `BranchModelItem` — a dynamic representation of the latest version of the model item on a branch. May be used, for example, to update the model item on the branch by saving a new latest version.
- `CommitModelItem` — a representation of the latest version of the model item at the time when a specific commit was saved in a branch. The version itself may have been saved in that same commit or in a previous commit on the branch. Can be used to read data values and other information related to the version, but cannot be used to save a new version of the model item.

Navigating a Model Manager Database

Searching

You can search for item versions using either the **Latest Versions for Location** search mode or the **All Versions in Database** search mode — see Searching Versions. The search term can contain both plain search words and any number of filter expressions written using The Model Manager Search Syntax.

To quickly see how to express a particular filter using the Model Manager search syntax, specify values in the **Filter** dialog for that filter and inspect the corresponding text shown under **Filter query preview**. See also The Filter Dialog.

EXAMPLES

Search for the latest versions of all drafts on a branch that are assigned the tag `My Project`.

```
SearchBranchItemResultStream resultStream = branch
  .searchItems("@itemSaveType:draft @tag:\"my project\"");
```

Field expressions in the Model Manager search syntax are automatically combined with AND-logic unless an explicit boolean operator is provided — see Table 3-5.

You can also write the previous code using a parameter object for which the two filters are provided as separate string arguments to the withSearchFilters method.

```
SearchItemsForBranchParamGenerator p = DatabaseApiUtil.param()
  .forSearchItemsForBranch()
  .withSearchFilters("@itemSaveType:draft",
                     "@tag:\"my project\"");
SearchBranchItemResultStream resultStream = branch.searchItems(p);
```

Search for all versions in the database that were saved with a commit comment containing the word solved. Sort the search result on the size of Built, Computed, and Plotted Data.

```
SearchItemVersionsParamGenerator p = DatabaseApiUtil.param()
  .forSearchItemVersions()
  .withSearchFilters("@commitComment:solved")
  .withSortByComputedData()
  .withSortDescending();
SearchItemVersionResultStream resultStream = database
  .searchItemVersions(p);
```

Advanced Database Operations Using Parameter Objects

Basic Loading and Saving of Models

You can load a model version from a Model Manager database using its model location URI. A loaded model can then, for example, be updated using the COMSOL API for the model object and subsequently be saved as a new version in the database.

EXAMPLES

Load the latest version of a model on the same branch as a fixed version using the loadModel method from the Method Editor's built-in method library.

```
// The modelLocationUri variable could have been obtained via
// Copy Location in the Model Manager workspace.
ModelItemVersion version = api
  .modelVersionByLocationUri(modelLocationUri);

BranchModelItem branchModelItem = version.branchItem();
Model m = loadModel(branchModelItem.modelLocationUri());
```

Save a regular version of a model originally loaded from the database.

```
ModelItemVersion version = api
  .saveRegularModel(model.tag(), "Saved a regular version.");
```

Save a draft version of a model originally loaded from the database.

```
ModelItemVersion version = api
  .saveDraftModel(model.tag(), "Saved a draft version.");

// Calling save directly yields the same result, except that
// you cannot specify a commit comment.
model.save();
```

Save a previously unsaved model as a new model in the database. The first version of the model is saved to the default branch in the default repository.

```
ModelItemVersion version = database
  .defaultRepository()
  .defaultBranch()
  .saveRegularModel(model.tag(), "Saved a new model.");
```

Force the creation of a new model in the database even if the model already exists in the database.

```
SaveModelItemParamGenerator p = DatabaseApiUtil.param()
  .forSaveModel()
  .withSourceModel(model.tag())
  .withTargetAsNew();

ModelItemVersion version = database
  .defaultRepository()
  .defaultBranch()
  .saveModel(p, null);
```

Importing Items

You can import models and files into a Model Manager database from either the file system or another database. When importing a file from the file system, the source location can be specified using either a regular file system path or a file scheme path. You can also choose to either create a new item or update an existing item in the target database.

File Schemes and File Handling in the *Application Builder Reference Manual*

EXAMPLES

Import an MPH file as a new model in the database.

```
String filePath = "C:\\My Models\\my_model.mph";
String title = "My Model";
ModelItemVersion version = database
  .defaultRepository()
  .defaultBranch()
  .saveNewModel(filePath, title, "Imported a new model");
```

Update an existing model on some branch from an MPH file.

```
String filePath = "C:\\My Models\\my_model.mph";
UpdateModelItemParamGenerator p = DatabaseApiUtil.param()
  .forUpdateModel()
  .withSourceLocation(filePath);

// The itemKey variable is initialized elsewhere.
ModelItemVersion version = branch
  .modelByKey(itemKey)
  .update(p, "Updated model from MPH file.");
```

Import a whole sequence of MPH files as versions of the same model.

```
String[] paths = {"C:\\my-model\\v1.mph", "C:\\my-model\\v2.mph",
  "C:\\my-model\\v3.mph", "C:\\my-model\\v4.mph"};

ModelItemVersion firstVersion = branch
  .saveNewModel(paths[0], "My Model", null);
ItemKey itemKey = firstVersion.itemKey();

// Update the model item from the remaining MPH files.
for (int i = 1; i < paths.length; i++) {
  UpdateModelItemParamGenerator p = DatabaseApiUtil.param()
    .forUpdateModel()
    .withSourceLocation(paths[i])
    .withIgnoreConflicts();
  branch.modelByKey(itemKey).update(p, null);
}
```

If the source MPH files were exported from model versions in some other database, you can use the `withIgnoreConflicts` method on the parameter generator interface to ignore the inevitable save conflict with the latest version in the database.

Import a geometry stored in the COMSOL native CAD format `mphbin`.

```
String filePath = "C:\\my_geometry.mphbin";

// Pass null for the title argument to automatically infer the title
// from the filename.
```

```
FileItemVersion version = branch
  .saveNewFile(filePath, null, "Imported CAD data.");
```

Import a CAD assembly with external component files as a fileset by specifying a directory as the source.

```
String filepath = "C:\\My CAD files";

// The title is required since we are importing multiple file
// resources, in which case the choice of filename is ambiguous.
FileItemVersion version = branch
  .saveNewFile(filePath, "My CAD Assembly", null);
```

Import a file version from a source database to a target database.

```
// The sourceDatabaseKey and sourceItemVersionKey variables are
// initialized elsewhere.
SaveFileItemParamGenerator p = DatabaseApiUtil.param()
  .forSaveFile()
  .withSourceDatabaseKey(sourceDatabaseKey)
  .withSourceItemVersionKey(sourceItemVersionKey);
FileItemVersion version = branch.saveFile(p, null);
```

Save a file version using the `temp` file scheme.

```
DatabaseApiParamGenerators param = DatabaseApiUtil.param();
SourceFileParamGenerator p1 = param
  .forSourceFile()
  .withLocationModel(model.tag())
  .withLocation("temp:///my_table_data.txt");
SaveFileItemParamGenerator p2 = param
  .forSaveFile()
  .withSourceFiles(p1);
FileItemVersion version = branch.saveFile(p2, null);
```

The `temp` file scheme requires that the model tag identifier of a model is provided. That model implicitly specifies the root directory path for the temporary files on the file system.

Advanced Save Operations for Items

Exporting Items

You can export model and file versions from the database to the file system. When exporting a file to the file system, the target location can be specified using either a regular file system path or a file scheme path.

EXAMPLES

Export a model version as an MPH file to the directory `C:\\My Models`.

```
// The itemVersionKey variable is initialized elsewhere. The
// filename saved with the model version is automatically used.
ExportModelItemVersionResult result = database
  .modelVersionByKey(itemVersionKey)
  .exportToDirectory("C:\\My Models");

// The file system path to the exported MPH file.
String filePath = result.exportedLocation();
```

Export a model version using a custom filename and without its Built, Computed, and Plotted Data.

```
// The modelLocationUri variable is initialized elsewhere.
ModelItemVersion version = DatabaseApiUtil
  .api()
  .modelVersionByLocationUri(modelLocationUri);

ExportModelItemVersionParamGenerator p = DatabaseApiUtil.param()
  .forExportModelVersion()
  .withTargetFilename("my_model_no_computed_data.mph")
  .withTargetDirectoryLocation("C:\\My Models")
  .withSourceComputedDataExcluded();
ExportModelItemVersionResult result = version.export(p);
String filePath = result.exportedLocation();
```

Export a file version to the root directory identified by the `common` file scheme.

```
SearchItemVersionResult result = database
  .searchItemVersions("my_geometry.mphbin")
  .firstOrNull();
database
  .fileVersionByKey(result.itemVersionKey())
  .exportToDirectory("common:///");
```

Unlike for example the `temp` file scheme, the `common` file scheme does not require an associated model.

Querying Versions

Given an item in a Model Manager database, you can query different subsets of versions associated with the item. You can, for example, query its version history with respect to the latest version on a branch, query all versions of the item ever saved in the database, or query versions of other items that are referenced by one of the item's versions.

EXAMPLES

Query the version history of a file item on the same branch as one of its file versions.

```
// The fileLocationUri variable is initialized elsewhere.
FileItemVersion version = DatabaseApiUtil
  .api()
  .fileVersionByLocationUri(fileLocationUri);
QueryItemVersionHistoryResultStream resultStream = version
  .branchItem()
  .queryVersionHistory();
```

Query versions associated with a model by including all versions of the model itself as well as all versions of drafts created from the model.

```
QueryItemVersionParamGenerator p = DatabaseApiUtil
  .param()
  .forQueryItemVersions()
  .withItemVersionInclusionOptions(
    QueryItemVersionInclusionOption.INCLUDE_ANCILLARY);

// The itemKey variable is initialized elsewhere.
QueryItemVersionResultStream resultStream = database
  .modelByKey(itemKey)
  .queryVersions(p);
```

Search for the versions of all drafts created from a model.

```
// The itemKey variable for the model is initialized elsewhere.
SearchItemVersionsParamGenerator p = DatabaseApiUtil.param()
  .forSearchItemVersions()
  .withSearchFilters("@itemSaveType:draft")
  .withSearchFilters("@originItemKey:" + itemKey.value());
SearchItemVersionResultStream resultStream = database
  .searchItemVersions(p);
```

Query all geometry parts referenced by a model version.

```
QueryItemVersionReferencesParamGenerator p = DatabaseApiUtil
  .param()
  .forQueryItemVersionReferences()
  .withItemVersionReferenceTypes(
    ItemVersionReferenceType.GEOMETRYPART);

// The itemVersionKey variable is initialized elsewhere.
QueryItemVersionReferenceResultStream resultStream = database
  .modelVersionByKey(itemVersionKey)
  .queryVersionReferences(p);
```

Search for the latest versions of models containing reusable geometry parts.

```
SearchBranchItemResultStream resultStream = branch
  .searchItems("@part:geometry");
```

Working with Input and Output

You can reference a version in the database as an input source or an output target from a model. The referenced version itself is identified by an item version location URI.

A referenced file version is loaded on demand from the database to a temporary working copy directory located on the computer running COMSOL Multiphysics (the server computer when running COMSOL Multiphysics in client–server mode). Any input read by the model, and any output written by the model, goes via files in this directory. You save the contents of the working copy directory as a new file version in the database by specifying the model tag identifier of the model and the location URI of the file version.

A location URI for a file version is written using the so-called `dbfile` file scheme. A location URI for a model version does not use a file scheme.

- Copying Model and File Locations
- Loading and Saving Auxiliary Data Files Stored in Databases

EXAMPLES

Set a file version containing a single file resource as an input source for an interpolation function feature.

```
SearchBranchItemResult result = branch
  .searchItems("my_interpolation.txt")
  .single();
FileItemVersion version = database
  .fileVersionByKey(result.itemVersionKey());
String fileLocationUri = version
  .firstFileResourceOrNull()
  .fileLocationUri();

FunctionFeature f = model.func("int1");
f.set("source", "file").set("filename", fileLocationUri);
f.importData();
```

Set a fileset version containing a CAD assembly as an input source for a geometry feature.

```
SearchBranchItemResult result = branch
  .searchItems("My Cad Assembly")
  .single();
FileItemVersion version = database
  .fileVersionByKey(result.itemVersionKey());

// Find the file resource corresponding to the main assembly file
// in the fileset.
String fileLocationUri = "";
for (FileResourceResult resource : version.fileResources()) {
  if (resource.path().filename().endsWith(".asm")) {
    fileLocationUri = resource.fileLocationUri();
    break;
  }
}

GeomSequence gs = model.component("comp1").geom("geom1");
GeomFeature f = gs.create("imp1", "Import");
f.set("filename", fileLocationUri);
f.importData();
```

Set a model version as an input source for a geometry part.

```
SearchBranchItemResult result = branch
  .searchItems("My Geometry Part @part:geometry").single();
ModelItemVersion version = database
  .modelVersionByKey(result.itemVersionKey());
String modelLocationUri = version.modelLocationUri();

// We assume that the geometry part has the tag identifier "part1".
String[] partTags = new String[]{"part1"};
model.geom().load(partTags, modelLocationUri, partTags);
```

Set a new file as the output target for a mesh export.

```
// Declare a file location URI for a new, but not yet created, file.
String locationUri = branch.newFileLocationUri("my_mesh.mphbin");

// Export the mesh as a working copy.
MeshSequence m = model.component("comp1").mesh("mesh1");
m.export().set("filename", locationUri);
m.export(location);

// Save the working copy to the branch as a new file version.
SaveFileItemParamGenerator p = DatabaseApiUtil.param()
  .forSaveFile()
  .withSourceItemVersionWorkingCopy(model.tag(), locationUri);
FileItemVersion version = branch
  .saveFile(p, "Saved an exported mesh.");
```

Set an existing file as the output target for an HTML report.

```
SearchBranchItemResult result = branch
  .searchItems("my_report.html")
  .single();
FileItemVersion version1 = database
  .fileVersionByKey(result.itemVersionKey());
String location = version1
  .firstFileResourceOrNull()
  .fileLocationUri();

ReportFeature r = model.result().report("rpt1");
r.set("filename", location);
r.run();

SaveFileItemParamGenerator p = DatabaseApiUtil.param()
  .forSaveFile()
  .withSourceItemVersionWorkingCopy(model.tag(), location);
FileItemVersion version2 = branch
  .saveFile(p, "Updated a report.");
```

Advanced Save Operations for Items

The options available for the parameter objects used when saving items enables you to perform a variety of advanced operations that go beyond what you have seen so far. A few examples:

- You can save a new version of an item using the version of some other item as a source. This is useful, for example, if you have previously created a new regular model via **Save as New** () in the **Save** window but now would like to save the new model back to its origin model.
- You can combine file resources belonging to different file versions into a new file version. You may, for example, want to combine external components files from different CAD assemblies into a new fileset version.
- You can save a model and multiple data files as new versions in a single commit. The data files could, for example, be working copies containing output files exported by the model. Saving the items collectively in a single commit enables you to easily trace the source model version of the output files via the **References** window.

EXAMPLES

Save a model loaded into COMSOL Multiphysics as a version of another model in the database.

```
// The targetItemKey variable is initialized elsewhere as the
// identifying key of an existing model in the database.
```

```
SaveModelItemParamGenerator p = DatabaseApiUtil.param()
  .forSaveModel()
  .withSourceModel(model.getTag())
  .withTargetItemKey(targetItemKey);
ModelItemVersion version = branch
  .saveModel(p, "Saved a model as a version of another model.");
```

Combine three file resources from two separate file versions into a new file.

```
DatabaseApiParamGenerators param = DatabaseApiUtil.param();

// The itemVersionKey1 and itemVersionKey2 variables are obtained
// elsewhere as the identifying keys of two existing file versions.
SourceFileParamGenerator p1 = param
  .forSourceFile()
  .withItemVersionKey(itemVersionKey1)
  .withFileResourcePath("casing.prt");
SourceFileParamGenerator p2 = param
  .forSourceFile()
  .withItemVersionKey(itemVersionKey1)
  .withFileResourcePath("cover.prt");
SourceFileParamGenerator p3 = param
  .forSourceFile()
  .withItemVersionKey(itemVersionKey2)
  .withFileResourcePath("stator_coil.prt");

SaveFileItemParamGenerator p = param
  .forSaveFile()
  .withSourceFiles(p1, p2, p3)
  .withTitle("My Induction Motor Assembly")
  .withTargetAsNew();
FileItemVersion version = branch.saveFile(p, "Saved a new file.");
```

Export a file version containing table data to a temporary file using the `temp` file scheme. Append a table row to the temporary file using the `writeFile` method from the Method Editor's built-in method library. Save the updated table data as a new file version.

```
String tempFilePath = "temp:///table.tmp";

// The location variable identifying the file version is
// initialized elsewhere.
FileItemVersion fileVersion = api
  .fileVersionByLocationUri(location);

FileResourcePath fileResourcePath = fileVersion
  .firstFileResourceOrNull()
  .path();

DatabaseApiParamGenerators param = DatabaseApiUtil.param();
```

```
ExportFileItemVersionParamGenerator p1 = param
  .forExportFileVersion()
  .withSourceFileResourcePath(fileResourcePath)
  .withTargetLocationModel(model.tag())
  .withTargetLocation(tempFilePath)
  .withTargetWriteOptions(
     ExportFileItemVersionTargetWriteOption.REPLACE_EXISTING);
fileVersion.export(p1);

// Append a table row to the exported file.
writeFile(tempFilePath, new String[][]{{"1", "2", "3"}}, true);

// Save the updated table data as a new file version.
SourceFileParamGenerator p2 = param
  .forSourceFile()
  .withLocationModel(model.tag())
  .withLocation(tempFilePath);
UpdateFileItemParamGenerator p3 = param
  .forUpdateFile()
  .withSourceFiles(p2);

fileVersion
  .branchItem()
  .update(p3, "Appended a table row.");
```

Export table data from a model and then save both the model and the table data as new versions in a single commit.

```
String location = branch.newFileLocationUri("my_table.txt");

ExportFeature e = model.result().export("table1");
e.set("filename", location);
e.run();

DatabaseApiParamGenerators param = DatabaseApiUtil.param();

SaveModelItemParamGenerator p1 = param
  .forSaveModel()
  .withSourceModel(model.tag());
SaveFileItemParamGenerator p2 = param
  .forSaveFile()
  .withSourceItemVersionWorkingCopy(model.tag(), location);
SaveCommitParamGenerator p = param
  .forSaveCommit()
  .withModelsToSave(p1)
  .withFilesToSave(p2);

Commit commit = branch.saveCommit(p);
```

7

Glossary

This Glossary of Terms contains terms related to databases and version control as they relate to the Model Manager tools in the COMSOL Multiphysics® software and documentation. For references to further information about a term, see the index.

Glossary of Terms

access-control list A list of granted permissions and their grantees — that is, users and groups — assigned to a database object.

administrator A privileged account with a Model Manager server that always passes permission requirement checks.

auxiliary data Data that is referenced as input or output by a model in the COMSOL Desktop environment. The data may, for example, be a version-controlled model or data file in a Model Manager database or a file on the file system.

boolean field The type for a field whose value is either `true` or `false`.

branch A sequence of commits in a database, ordered chronologically by their commit date to form a *commit history*. A special commit is the most recent one on the branch, and the branch itself often acts as a representative of this commit. The most recent commit, and thus the branch itself, also identifies the latest versions of items.

New branches can be created by *branching off* from any commit on a *parent branch*, thereby starting an alternative commit history. Changes made on a new branch can be *merged* back to its parent branch.

built, computed, and plotted data Generated simulation data that can be recreated from a model as needed. Includes built geometries and meshes, computed solutions, and plotted results.

commit A set of related item changes that have been saved to a database. The changes may involve adding and updating items, assigning tags to items, and deleting items. The changes are saved to the database as a "unit" and, as such, can also be *reverted* as a unit.

Each commit is associated with a date and time when the changes were saved, that is the *commit date*, the user that saved the changes, and an optional save comment. Each commit also identifies the set of item versions that were the *latest versions* at the time of the commit, as well as all tag assignments present at that time. Given a particular commit, such item versions and tag assignments can be browsed and searched in the database.

content filter A search filter on model content — that is, node properties, parameters, features, and other settings in the model tree of a model.

data file A version-controlled file stored in a database that is neither a model or application file (`mph`) nor a physics file (`mphphb`). Typically abbreviated as just *file* in Model Manager. Examples include CAD data, interpolation functions, plots, and reports.

database configuration The configuration values used to connect to a local database or a server database accessed via a Model Manager server.

date field The type for a field whose value is a date and time.

draft model An ancillary model meant for intermediate modeling work, and whose versions are saved as a separate version history *split off* from that of its *origin* model. Once a draft has been completed, it can be saved back as a new version of the main model it originated from.

everyone A special group of users that all users automatically are members of. Can be used when assigning permissions to database objects.

file A shorthand for a data file.

fileset A file version consisting of multiple file resources that are version controlled as a collective whole in a database. This includes, for example, a CAD assembly with multiple external component files or an HTML report with image files.

file resource Binary or text content belonging to a file version. A file version can contain any number of file resources.

item A model, data file, or tag stored in a database.

item filter A search filter on general item settings and metadata. This includes, for example, the time when the item version was saved, the user that saved the item version, and the item's assigned tags.

item save type Type distinguishing a regular model from a draft model.

item version type Type distinguishing a model, application, physics, file, and fileset in a database.

group A collection of users and other groups.

keyword field The type for a field whose value is a string. Typically found for short, name-like search data such as filenames or node names.

local database A database stored on the same computer as the COMSOL Multiphysics process runs on. Meant for single-user use.

(commit) location A branch, commit, or snapshot in the database. Each location is either a commit in its own right or acts as a natural representation of a commit — the most recent commit for a branch and the referenced commit for a snapshot.

Browsing or searching items in a database is always done with respect to a fixed location. The encountered item versions are the ones that were the latest at the time of the commit. For a branch, which is the default location, these item versions are the latest at the present time.

(item version) location An identifier that uniquely locates a model version or data file version stored in a Model Manager database.

merge The operation of applying item changes made in one branch to another branch.

model A COMSOL Multiphysics simulation model, application, or physics stored in a database.

negated match Reverse the matching criterion of a filter.

numeric field The type for a field whose value is a real or complex scalar.

origin model A model that another model "originates" from — a concept arising, for example, when creating a new model from an existing model or when saving a new draft.

owner The user that owns a database object. Such a user can set permission requirements for accessing the object.

permission template A reusable template of permissions granted to users and groups.

phrase match Require that search words match a sequence of words in a text.

regular model The standard type for models saved to a database.

repository A container for a collection of items and their versions in the database.

revert The operation of undoing a set of changes made in a commit.

selection field The type for a field whose value belongs to a predetermined set of values.

server database A database accessed via a Model Manager server. Meant for multiple-user use.

snapshot A reference to a commit in a database. The item versions and tag assignments that were the latest at the time of the commit are *recorded* in the snapshot.

tag Version-controlled metadata that can be assigned to models, data files, and even other tags. Useful for finding and organizing items in a database.

text field The type for a field whose value is a text.

user A user that has connected to a Model Manager database.

version The result when creating, updating, or restoring an item in a database.

wildcard match Use a placeholder in a search word that matches zero or more arbitrary characters.

Index