MATLAB® 7
Desktop Tools and Development Environment

MATLAB®

The MathWorks™
Accelerating the pace of engineering and science
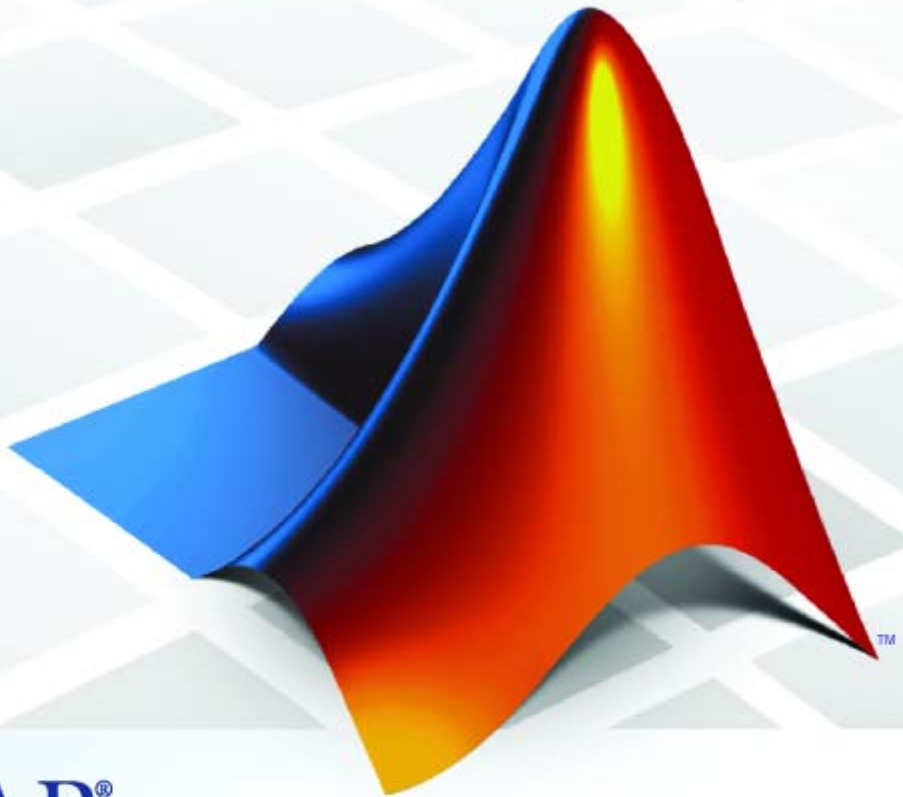
How to Contact The MathWorks

www.mathworks.com	Web
comp.soft-sys.matlab	Newsgroup
www.mathworks.com/contact_TS.html	Technical Support

suggest@mathworks.com	Product enhancement suggestions
bugs@mathworks.com	Bug reports
doc@mathworks.com	Documentation error reports
service@mathworks.com	Order status, license renewals, passcodes
info@mathworks.com	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

MATLAB® Desktop Tools and Development Environment

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

The MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Contents

Startup and Shutdown

1

Desktop

2

Running Functions — Command Window and History

3

Help, Demos, and Related Resources

4

Workspace, Search Path, and File Operations

5

Editing and Debugging M-Files

6

Tuning and Managing M-Files

7

Publishing M-Files

8

Using Notebook to Publish to Microsoft Word

9

Source Control Interface

10

Internationalization

11

Index

1

Startup and Shutdown

This set of topics includes options for customizing startup and shutdown of the MATLAB® program.

- "Overview of Starting the MATLAB Program" on page 1-2
- "Starting the MATLAB Program on Windows Platforms" on page 1-2
- "Starting the MATLAB Program on UNIX Platforms" on page 1-7
- "Starting the MATLAB Program on Macintosh Platforms" on page 1-9
- "Startup Directory for the MATLAB Program" on page 1-11
- "Startup Options" on page 1-17
- "Toolbox Path Caching in the MATLAB Program" on page 1-22
- "Other Startup Topics" on page 1-25
- "Quitting the MATLAB Program" on page 1-27

Overview of Starting the MATLAB Program

The way you start the MATLAB program depends on the platform you use:

- On Microsoft® Windows® platforms, start MATLAB by double-clicking the `MATLAB R2009a` shortcut on your Windows desktop.

- On Apple® Macintosh® platforms, start MATLAB by double-clicking the `MATLAB_R2009a` icon in the `Applications` folder.

- On platforms that run the UNIX®[1] operating system, start MATLAB by typing `matlab` at the operating system prompt.

When you start MATLAB, all the product software from The MathWorks™ that you are licensed to use, is available. You do not have to start each product separately.

There are alternative ways to start MATLAB, and you can customize startup. For example, you can change the directory in which MATLAB starts or automatically execute MATLAB statements upon startup.

Starting the MATLAB Program on Windows Platforms

In this section...
"Starting the MATLAB Program from the Windows Desktop or a DOS Window" on page 1-3
"Starting the MATLAB Program from an M-File or Other File Type on Windows Platforms" on page 1-3
"Utility to Change File Associations for Windows Platforms" on page 1-6
"Changing File Associations for the MATLAB Program from the Windows Environment" on page 1-6

---

1. UNIX is a registered trademark of The Open Group in the United States and other countries.

Starting the MATLAB Program from the Windows Desktop or a DOS Window

To start the MATLAB program on a Microsoft Windows platform, select **Start > Programs > MATLAB > R2009a > MATLAB R2009a**, or double-click the shortcut icon for MATLAB R2009a 🖼 on your Windows desktop. The shortcut was automatically created when you installed MATLAB. If you have trouble starting MATLAB, see troubleshooting information in the *Installation Guide for Windows*.

To start MATLAB from a DOS window, `cd` to the directory in which you want to start MATLAB and type `matlab` at the DOS prompt.

You can specify the current directory upon startup as well as other options—for more information, see "Startup Directory for the MATLAB Program" on page 1-11 and "Startup Options" on page 1-17.

After starting MATLAB, the desktop opens. Desktop components that were open when you last shut down MATLAB will be opened on startup. For more information, see Chapter 2, "Desktop".

Starting the MATLAB Program from an M-File or Other File Type on Windows Platforms

On Windows platforms, you can start MATLAB from the Windows Explorer tool by double-clicking a file with one of these extensions: `.fig`, `.m`, `.mat`, and `.mdl`. MATLAB starts and opens in an appropriate tool. If MATLAB is already running, the file opens in an appropriate tool in the existing session.

This startup feature is based on your file type associations for the Windows operating system. When you installed MATLAB for Windows platforms, you specified the file types to associate with MATLAB. For example, if you accepted the default options, double-clicking an M-file in the Windows Explorer tool opens the file in the MATLAB Editor.

The default option also associates MEX-files and P-files with MATLAB in the Windows Explorer tool, which assigns the file types an icon for MATLAB. However, double-clicking a file with a `.mex` (`.mexw32` or `.mexw64`), or `.p` extension does not run or open the file in MATLAB.

File Type and Resulting Action

File Type	Result
FIG-file	Opens file in figure window
M-file	Opens file in Editor
MAT-file	Opens Import Wizard to load the data into the MATLAB workspace
MDL-file	Opens file in a Simulink® model window
MEX-file	Displays icon for MATLAB in Windows Explorer tool
P-file	Displays icon for MATLAB in Windows Explorer tool

Other applications you use can change file associations for your Windows environment. For example, the Microsoft® Access™ application might associate files having a .mat extension. Then, double-clicking a MAT-file opens the Access™ application rather than MATLAB.

If you double-click a FIG-file, M-file, MAT-file, or MDL-file and it does not open in , and you want it to, try this:

1 In the Windows Explorer tool, right-click a file that has one of the extensions for MATLAB, for example, myfile.mat.

2 From the context menu, select **Open With**. If MATLAB is one of the choices, select it to open myfile.mat in MATLAB. If MATLAB is not one of the choices, you will need to associate the file type with MATLAB using one of these techniques:

- "Utility to Change File Associations for Windows Platforms" on page 1-6
- "Changing File Associations for the MATLAB Program from the Windows Environment" on page 1-6

After associating a file type with MATLAB, you can open other applications that have the same extension via the context menu. For example, if you want to open a MAT-file with the Access application, right-click myfile.mat, and from the context menu, select **Open With**. The Access application should be one of the options.
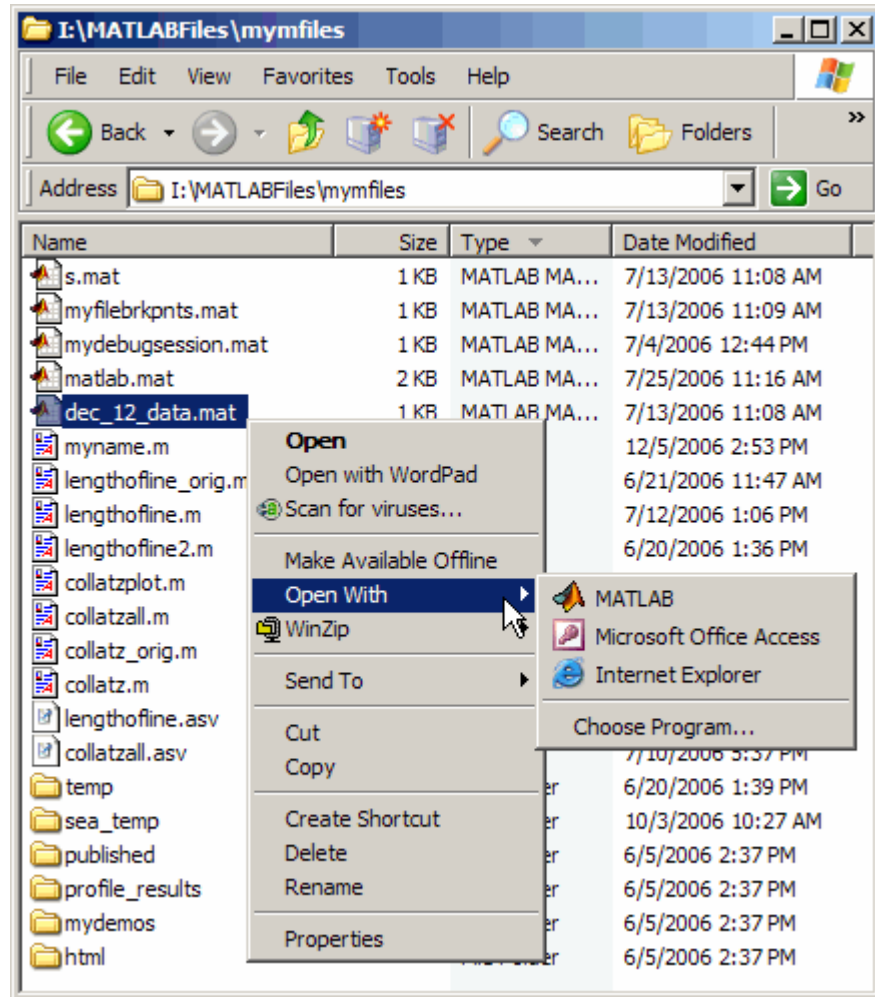
File associations for the Windows Explorer tool do not affect what happens when you open one of these file types from *within* MATLAB. MATLAB acts on the file using the MATLAB tool associated with that file type. For example, even if you associate .mat files with the Access application, when you open a MAT-file from within MATLAB, it opens the Import Wizard to load the data.

Utility to Change File Associations for Windows Platforms

If you are viewing this topic in the MATLAB Help browser, you can run one of the utilities provided here to create associations in the Windows environment for common file types used by MATLAB. This requires you to have permission to write to the HKEY_CLASSES_ROOT registry key, which typically requires power user or administrator privileges.

- Run utility to associate MATLAB with FIG-files

- Run utility to associate MATLAB with M-files

- Run utility to associate MATLAB with MAT-files

- Run utility to associate MATLAB with MDL-files

- Run utility to associate MATLAB with MEX-files

- Run utility to associate MATLAB with P-files

- Run utility to associate MATLAB with all of these file types: FIG, M, MAT, MDL, MEX, and P

The file type icon in the Windows Explorer tool might not reflect the change immediately.

Changing File Associations for the MATLAB Program from the Windows Environment

You can associate file types with MATLAB from the Windows Explorer tool. This is useful if you want associate file types other than those you can change with the "Utility to Change File Associations for Windows Platforms" on page 1-6. For example, you can associate the .xml extension with MATLAB so that when you double-click an XML file, it opens in the MATLAB Editor.

The following examples show one way to change file associations in the Windows Explorer tool. Note that these instructions might not exactly apply to your version of the Windows operating system. If you encounter differences or problems, try to delete the association before using these instructions, or see your Windows documentation.

Assume that when you double-click a `.mat` file in the Windows Explorer tool, it opens in the Microsoft Access application, but you want the file to open in MATLAB.

1 In the Windows Explorer tool, select **Tools > Folder Options**.

2 In the resulting Folder Options dialog box, select the **File Types** tab. From the **Registered file types** list, select the `MAT` extension. (If you do not see `MAT` in the list, click **New** to add it.)

Under **Details for 'MAT" extension**, click **Change**.

3 In the resulting Open With dialog box, select `MATLAB` from the list.

If the list does not include `MATLAB`, click **Browse**. Then look for and select `matlab.exe`, and click **Open**. The file is located in the folder in which you installed MATLAB. An example of the default location is `C:\Program Files\MATLAB\R2009a\bin`.

4 In the Open With dialog box, click **OK**. In the Folder Options dialog box, click **Close**.

Starting the MATLAB Program on UNIX Platforms

To start the MATLAB program on UNIX[2] platforms, type `matlab` at the operating system prompt.

If you did not set up symbolic links in the installation procedure, you must enter the full pathname to start MATLAB, *matlabroot*/bin/matlab, where *matlabroot* is the name of the directory in which you installed MATLAB. (For more information, see the `matlabroot` reference page). If you have trouble starting MATLAB, see troubleshooting information in the *Installation Guide for UNIX*.

---

2. UNIX is a registered trademark of The Open Group in the United States and other countries.

You can specify the current directory upon startup as well as other options—for more information, see "Startup Directory for the MATLAB Program" on page 1-11 and "Startup Options" on page 1-17.

After starting MATLAB, the desktop opens. Desktop components that were open when you last shut down MATLAB will be opened on startup. For more information, see Chapter 2, "Desktop". If the DISPLAY environment variable is not set or is invalid, the desktop will not display.

Starting the MATLAB Program on Macintosh Platforms

In this section...

Starting the MATLAB Program from the Macintosh Desktop

To start the MATLAB program on Apple Macintosh platforms, double-click the `MATLAB_R2009a` icon in the `Applications` folder.

You can specify the current directory upon startup as well as other options—for more information, see "Startup Directory for the MATLAB Program" on page 1-11 and "Startup Options" on page 1-17.

If MATLAB fails to start due to a problem with required system components such as X11 or Sun Microsystems™ Java™ software, diagnostics run automatically and advise you of the problem, along with suggestions to correct it.

After starting MATLAB, the desktop opens. Desktop components that were open when you last shut down MATLAB will be opened on startup. For more information, see Chapter 2, "Desktop". If the `DISPLAY` environment variable is not set or is invalid, the desktop will not display.

Limitation

On Macintosh platforms, if you run MATLAB remotely, for example using `rlogin`, you must run with `nodisplay`, `noawt`, and `nojvm` startup options—for more information, see "Startup Options" on page 1-17.

Starting the MATLAB Program from a Shell on Macintosh Platforms

You can start MATLAB on Macintosh platforms the way you would start it on other UNIX[3] platforms. For example, you would run

```
/Applications/MATLAB_R2009a.app/bin/matlab
```

For more information, see "Starting the MATLAB Program on UNIX Platforms" on page 1-7.

---

3. UNIX is a registered trademark of The Open Group in the United States and other countries.

Startup Directory for the MATLAB Program

In this section...

What Is the Startup Directory?

The *startup directory* is the current directory in the MATLAB application when it starts. It is convenient if you make the current directory upon startup be a directory that you frequently use. On Windows and Apple Macintosh platforms, a directory called `userpath` is added automatically to the search path upon startup, and is the default startup directory. On UNIX[4] platforms, you can set the `userpath` as the startup directory. The default value for `userpath` is, for example, `Documents/MATLAB` on Microsoft Windows Vista™ platforms. You can specify a different default value for `userpath`, or specify a different startup directory.

Accepting the default value for `userpath` and using it as the startup directory offers these benefits:

- You can store the MATLAB files you work with in one, appropriately-named location, such as `Documents/MATLAB`.

- Your MATLAB files are readily available upon startup, because the current directory is always the same, for example, `Documents/MATLAB`.

- You can always run your files because MATLAB automatically adds the `userpath` subdirectory to the top of the search path upon startup.

- The first time you run a new version of MATLAB, MATLAB automatically creates the `userpath` subdirectory if it does not exist.

---

4. UNIX is a registered trademark of The Open Group in the United States and other countries.

- When you upgrade to a newer version of MATLAB, MATLAB automatically continues to use the same `MATLAB` subdirectory and your existing files, with all of its other benefits.

- The default `userpath` also utilizes the benefits provided by the standard location in the Windows and Macintosh environments for storing personal files. Files in the `Documents/MATLAB` subdirectory (or `My Documents/MATLAB` on Windows platforms other than Windows Vista) are available to you when you use other machines. Because each user has their own `Documents/MATLAB` subdirectory, other users, even those using your machine, cannot access files in your `Documents/MATLAB` subdirectory.

To view the `userpath` value, run the `userpath` function. To specify a location other than the default for `userpath`, or if you do not want to take advantage of `userpath`, make changes with the `userpath` function.

There are other ways to change the startup directory as well as the directories on your search path. For more information, see "Changing the Startup Directory" on page 1-14 and "Overview of Viewing and Changing the Search Path" on page 5-40.

Startup Directory (Folder) on Windows Platforms

The startup directory (folder) on Windows platforms depends on any startup options you specified and the way you started MATLAB:

- "Startup Directory When Starting the MATLAB Program from a Windows Shortcut" on page 1-12

- "Startup Directory When Starting the MATLAB Program from an Associated File" on page 1-13

- "Startup Directory When Starting the MATLAB Program from a DOS Window" on page 1-13

Startup Directory When Starting the MATLAB Program from a Windows Shortcut

When you start the MATLAB program from a Windows shortcut, by default MATLAB sets the startup directory to the `userpath` value, whose default value is `My Documents\MATLAB`, or `Documents\MATLAB` on Windows Vista platforms. The `userpath` directory is automatically added to the search path.

If there is a value specified in the **Start in** field of the Properties dialog box for the MATLAB program, that value is the startup directory, although the `userpath` is added to the search path. If MATLAB does not find a valid `userpath` value upon startup, and the **Start in** field is empty, the startup directory is the Windows desktop.

Startup Directory When Starting the MATLAB Program from an Associated File

When you start MATLAB by double-clicking a file type associated with MATLAB, that file's directory is the startup directory. The `userpath` directory is automatically added to the search path.

Startup Directory When Starting the MATLAB Program from a DOS Window

When you start MATLAB from a DOS window, the startup directory is the directory in which you ran the `matlab` function. The `userpath` directory is automatically added to the search path.

Startup Directory on UNIX Platforms

On UNIX platforms, the default startup directory is the directory from which you started MATLAB. You can specify that the `userpath` be the startup directory by setting the value of the environment variable `MATLAB_USE_USERPATH` to 1 prior to startup. By default, `userpath` is *userhome*`/Documents/MATLAB`, and MATLAB automatically adds the `userpath` directory to the top of the search path upon startup. To specify a different directory for `userpath`, and for other options, use the `userpath` function.

Startup Directory on Macintosh Platforms

When you start MATLAB on Apple Macintosh platforms by double-clicking the MATLAB application, the startup directory is the value returned when you enter `userpath`, which by default is *userhome*`/Documents/MATLAB`. MATLAB automatically adds the `userpath` directory to the top of its search path upon startup. To specify a different directory for `userpath`, and for other options, use the `userpath` function.

When you start MATLAB in a shell, the startup directory is the same as for other UNIX platforms—see "Startup Directory on UNIX Platforms" on page 1-13.

Changing the Startup Directory

You can start MATLAB in a directory other than the default in one of these ways:

- "Changing the Startup Directory Via the userpath Function" on page 1-14
- "Changing the Startup Directory Using the Shortcut — Windows Platforms Only" on page 1-14
- "Changing the Startup Directory Using the startup.m File" on page 1-16

Changing the Startup Directory Via the userpath Function

Use the userpath function to change the startup directory as well as to add the startup directory to the search path upon startup. For more information, see the userpath reference page and "Startup Directory for the MATLAB Program" on page 1-11.

Changing the Startup Directory Using the Shortcut — Windows Platforms Only

To change the startup directory on Windows platforms using the shortcut,

1 Right-click the shortcut icon for MATLAB and select **Properties** from the context menu.

The Properties dialog box for MATLAB opens to the **Shortcut** pane.

2 The **Target** field contains the full path to start MATLAB.

By default, the startup directory is `My Documents\MATLAB` or `Documents\MATLAB` on Windows Vista platforms; for more information, see "Startup Directory (Folder) on Windows Platforms" on page 1-12.

In the **Start in** field, specify the full path to the directory in which you want MATLAB to start, and click **OK**.

In the **Start in** field, enter the full path to the directory in which you want MATLAB to the start. For example, `I:\my_matlab_files`.
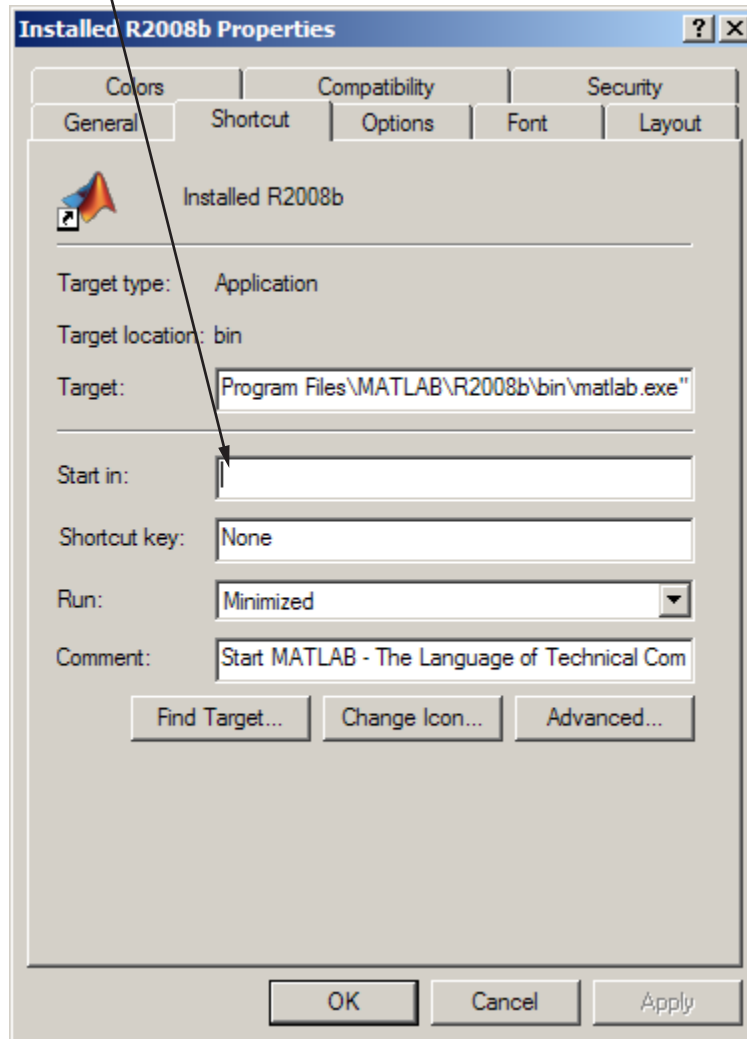
The next time you start MATLAB using that shortcut icon, the current directory will be the one you specified in step 2.

You can make multiple shortcuts to start MATLAB, each with its own startup directory, and with each startup directory having different startup options.

Changing the Startup Directory Using the startup.m File

Use the `startup.m` file to specify the startup directory as well as other startup options—for details, see "Specifying Startup Options Using the Startup File for the MATLAB Program, startup.m" on page 1-19.

Startup Options

In this section...
"About Startup Options" on page 1-17
"Specifying Startup Options for Windows Platforms" on page 1-17
"Specifying Startup Options for UNIX Platforms" on page 1-19
"Specifying Startup Options for Macintosh Platforms" on page 1-19
"Specifying Startup Options Using the Startup File for the MATLAB Program, startup.m" on page 1-19
"Commonly Used Startup Options" on page 1-20

About Startup Options

You can define startup options that instruct the MATLAB program to perform certain operations when you start it. On MicrosoftWindows platforms, you can use a GUI to specify the options. On all platforms, you can specify these options using a startup file (`startup.m`), or in conjunction with the `matlab` startup function.

Specifying Startup Options for Windows Platforms

You can add selected startup options (also called command flags or switches for the command line) to the target path for your shortcut in the Windows environment for MATLAB. Or you can add them to the command line when you start MATLAB in a DOS window. For more information about the options, see "Commonly Used Startup Options" on page 1-20.

On Windows platforms, a startup option is preceded by either a hyphen (-) or a slash (/). For example, `-nosplash` and `/nosplash` are equivalent ways of specifying the `nosplash` option for users on Windows platforms.

Startup Options for a Shortcut in Windows Environment

To use startup options for the MATLAB shortcut icon in a Windows environment, follow these steps:

1 Right-click the shortcut icon for MATLAB 📙 and select **Properties** from the context menu. The Properties dialog box for MATLAB opens to the **Shortcut** pane.

2 In the **Target** field, after the target path for `matlab.exe`, add the startup option, and click **OK**. For example, adding `-r "filename"` runs the M-file `filename` after startup.

This example instructs MATLAB to automatically run the file `results` after startup, where `results.m` is in the startup directory or on the search path for MATLAB. The statement in the **Target** field might appear as

```
C:\Program Files\MATLAB\R2009a\bin\matlab.exe -r "results"
```

Include the statement in double quotation marks (`"statement"`). Use only the filename, not the file extension or pathname. For example, MATLAB produces an error when you run

```
... matlab.exe -r "D:\results.m"
```

Use semicolons or commas to separate multiple statements. This example changes the format to `short`, and then runs the M-file `results`:

```
... matlab.exe -r "format('short');results"
```

Separate multiple options with spaces. This example starts MATLAB without displaying the splash screen, and then runs the M-file `results`:

```
... matlab.exe -nosplash -r "results"
```

Startup Options in a DOS Window

When you start MATLAB in a DOS window, include startup options after the `matlab` command.

This example uses the `nosplash` startup option to start MATLAB without the splash screen, and adds the `-r` option to run the `results` function located in the startup directory, after starting MATLAB in a DOS window:

```
matlab -nosplash -r "results"
```

Specifying Startup Options for UNIX Platforms

Include startup options (also called command flags or command line switches) after the matlab command when you start MATLAB on UNIX[5] platforms. For more information about the options, see "Commonly Used Startup Options" on page 1-20. On UNIX platforms, a startup option is preceded by a hyphen (-). For example, to start MATLAB without the splash screen, type

```
matlab -nosplash
```

See also the userpath function.

Specifying Startup Options for Macintosh Platforms

On Apple Macintosh platforms, specify startup options for Macintosh platforms with the matlab command, as described at "Specifying Startup Options for UNIX Platforms" on page 1-19.

Specifying Startup Options Using the Startup File for the MATLAB Program, startup.m

At startup, MATLAB automatically executes the master M-file matlabrc.m and, if it exists, startup.m. The file matlabrc.m, which is in the *matlabroot*/toolbox/local directory, is reserved for use by The MathWorks and by the system manager on multiuser systems.

The file startup.m is for you to specify startup options. For example, you can modify the default search path, predefine variables in your workspace, or define defaults for Handle Graphics® objects. Use the following statements in a startup.m file to add the specified directory, /home/*username*/mytools, to the search path, and to change the current directory to mytools upon startup.

```
addpath /home/username/mytools
cd /home/username/mytools
```

---

5. UNIX is a registered trademark of The Open Group in the United States and other countries.

Location of startup.m

Place the startup.m file in the default or current startup directory, which is where MATLAB first looks for it. For more information, see "Startup Directory for the MATLAB Program" on page 1-11.

Commonly Used Startup Options

The following table provides a list of some commonly used startup options for both Windows and UNIX platforms. For more information, including a complete list of startup options, see the matlab (Windows) reference page or the matlab (UNIX) reference page.

Platform	Option	Description
All	-c licensefile	Set LM_LICENSE_FILE to licensefile. It can have the form port@host.
All	-h or -help	Display startup options (without starting MATLAB).
All	-logfile "logfilename"	Automatically write output from MATLAB to the specified log file.
Windows platforms	-minimize	Start MATLAB with the desktop minimized. Any desktop tools or documents that were undocked when MATLAB was last closed will not be minimized upon startup.
UNIX platforms	-nojvm	Start MATLAB without loading the Sun Microsystems JVM™ software. This minimizes memory usage and improves initial startup speed, but restricts functionality. With nojvm, you cannot use the desktop, figures, or any tools that require Java software. For example, you cannot set preferences if you start MATLAB with the -nojvm option. However, you can start MATLAB once *without* the -nojvm option, set the preference, and quit MATLAB. MATLAB remembers that preference when you start it again, even if you use the -nojvm option.
All	-nosplash	Start MATLAB without displaying its splash screen.

Platform	Option	Description
All	`-r "statement"`	Automatically run the specified statement immediately after MATLAB starts. This is sometimes referred to as calling MATLAB in batch mode. Files you run must be in the startup directory for MATLAB or on the search path. Do not include pathnames or file extensions. Enclose the statement in double quotation marks (`"statement"`). Use semicolons or commas to separate multiple statements
All	`-singleCompThread`	Limit MATLAB to a single computational thread. By default, MATLAB makes use of the multithreading capabilities of the computer. For more information about multithreading, see "MATLAB Multiprocessing".

Toolbox Path Caching in the MATLAB Program

In this section...
"About Toolbox Path Caching in the MATLAB Program" on page 1-22
"Using the Cache File Upon Startup" on page 1-22
"Updating the Cache and Cache File" on page 1-22
"Additional Diagnostics with Toolbox Path Caching" on page 1-24

About Toolbox Path Caching in the MATLAB Program

For performance reasons, the MATLAB program caches toolbox directory information across sessions. The caching features are mostly transparent to you. However, if MATLAB does not see the latest versions of your M-files or if you receive warnings about the toolbox path cache, you might need to update the cache.

Using the Cache File Upon Startup

Upon startup, MATLAB gets information from a cache file to build the toolbox directory cache. Because of the cache file, startup is faster, especially if you run MATLAB from a network server or if you have many toolbox directories. When you end a session, MATLAB updates the cache file.

MATLAB does not use the cache file at startup if you clear the **Enable toolbox path cache** check box in **File > Preferences > General**. Instead, it creates the cache by reading from the operating system directories, which is slower than using the cache file.

Updating the Cache and Cache File

How the Toolbox Path Cache Works

MATLAB caches (essentially, stores in a known files list) the names and locations of files in *matlabroot*/toolbox directories. These directories are for files provided with MathWorks products that should not change except for product installations and updates. Caching those directories provides better

performance during a session because MATLAB does not actively monitor those directories.

We strongly recommend that you save any M-files you create and any files provided by The MathWorks that you edit in a directory that is *not* in the *matlabroot*/toolbox directory tree. If you keep your files in *matlabroot*/toolbox directories, they may be overwritten when you install a new version of MATLAB.

When to Update the Cache

When you add files to *matlabroot*/toolbox directories, the cache and the cache file need to be updated. MATLAB updates the cache and cache file automatically when you install toolboxes or toolbox updates using the installer for MATLAB. MATLAB also updates the cache and cache file automatically when you use MATLAB tools, such as when you save files from the MATLAB Editor to *matlabroot*/toolbox directories.

When you add or remove files in *matlabroot*/toolbox directories by some other means, MATLAB might not recognize those changes. For example, when you

- Save new files in *matlabroot*/toolbox directories using an external editor
- Use operating system features and commands to add or remove files in *matlabroot*/toolbox directories

MATLAB displays this message:

```
Undefined function or variable
```

You need to update the cache so MATLAB will recognize the changes you made in *matlabroot*/toolbox directories.

Steps to Update the Cache

To update the cache and the cache file,

1 Select **File > Preferences > General**.

   The **General Preferences** pane is displayed.

2 Click **Update Toolbox Path Cache** and click **OK**.
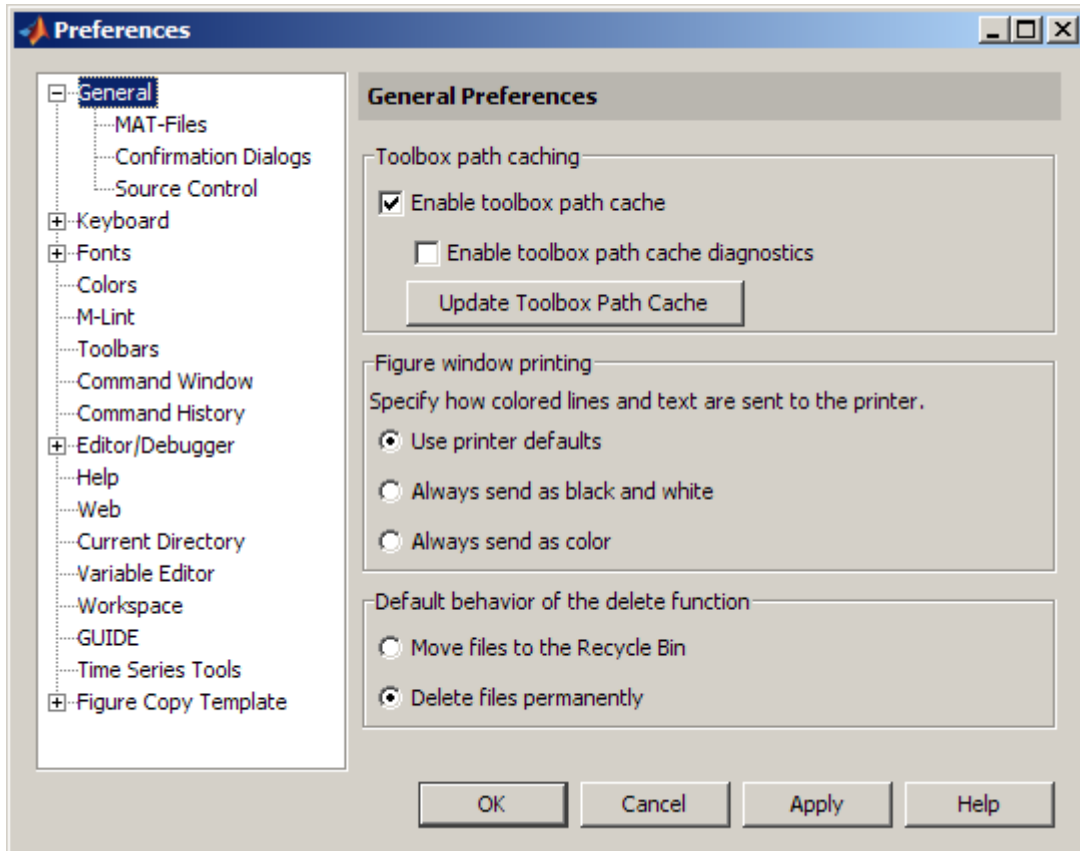
Function Alternative

To update the cache, use `rehash toolbox`. To also update the cache file, use `rehash toolboxcache`. For more information, see `rehash`.

Additional Diagnostics with Toolbox Path Caching

To display information about startup time when you start MATLAB, select the **Enable toolbox path cache diagnostics** check box in **General Preferences**.

Other Startup Topics

In this section...
"Error Log Reporter" on page 1-25
"Passing Perl Variables on Startup" on page 1-25
"Startup and Calling Java Software from the MATLAB Program" on page 1-26

Error Log Reporter

Upon startup, if the MATLAB program detects an error log generated by a serious problem encountered during the *previous* session, an Error Log Reporter prompts you to e-mail the log to The MathWorks for analysis. Click **Send Report** to e-mail the log, or click **Help** for more information. After sending the log, a confirmation message appears in the Command Window. For more information, see "Abnormal Termination" on page 1-28.

Passing Perl Variables on Startup

You can pass Perl variables to MATLAB on startup by using the `-r` option of the `matlab` function. For example, assume a MATLAB function `test` that takes one input variable:

```
function test(x)
```

To start MATLAB with the function `test`, use the command

```
matlab -r "test(10)"
```

On some platforms, you might need to use double quotation marks:

```
matlab -r "test(10)"
```

This command starts MATLAB and runs `test` with the input argument `10`.

To pass a Perl variable instead of a constant as the input parameter, follow these steps.

1 Create a Perl script such as

```
#!/usr/local/bin/perl
$val = 10;
system('matlab -r "test(' . ${val} . ')"');
```

2 Invoke the Perl script at the prompt using a Perl interpreter.

For more information, see the `matlab (Windows)` or `matlab (UNIX)` reference page.

Startup and Calling Java Software from the MATLAB Program

When the MATLAB program starts, it constructs the class path for Sun Microsystems Java software using `librarypath.txt` as well as `classpath.txt`. If you call Java software from MATLAB, see more about this in "The Java Class Path" and "Locating Native Method Libraries" in the MATLAB External Interfaces documentation.

Quitting the MATLAB Program

In this section...
"Ways to Quit the MATLAB Program" on page 1-27
"Confirm Quitting the MATLAB Program" on page 1-27
"Running a Script When Quitting the MATLAB Program" on page 1-28
"Abnormal Termination" on page 1-28

Ways to Quit the MATLAB Program

To quit the MATLAB program at any time, do one of the following:

- Click the Close box ⊠ in the MATLAB desktop.

- Select **Exit MATLAB** from the desktop **File** menu.

- Type quit at the Command Window prompt.

MATLAB closes after

- Prompting you to confirm quitting, if that preference is specified (see "Confirm Quitting the MATLAB Program" on page 1-27)

- Prompting you to save any unsaved files

- Running the finish.m script, if it exists in the current directory or on the search path (see "Running a Script When Quitting the MATLAB Program" on page 1-28)

Confirm Quitting the MATLAB Program

To set a preference that displays a confirmation dialog box when you quit MATLAB, select **File > Preferences > General > Confirmation Dialogs**, select the **Confirm before quitting** check box, and click **OK**. MATLAB then displays the following dialog box when you quit.
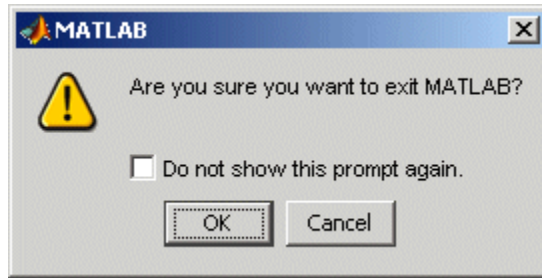
For more information, see "Confirmation Dialogs Preferences" on page 2-112.

You can also display your own quitting confirmation dialog box using a finish.m script, as described in the following section.

Running a Script When Quitting the MATLAB Program

When MATLAB quits, it runs the script finish.m, if finish.m exists in the current directory or anywhere on the search path. You create the file finish.m. It contains statements to run when MATLAB terminates, such as saving the workspace or displaying a confirmation dialog box. There are two sample files in *matlabroot*/toolbox/local that you can use as the basis for your own finish.m file:

- finishsav.m — Includes a save function so the workspace is saved to a MAT-file when MATLAB quits.
- finishdlg.m — Displays a confirmation dialog box that allows you to cancel quitting.

For more information, see the finish reference page.

Abnormal Termination

- "When the MATLAB Program Terminates Unexpectedly" on page 1-29
- "Error Log Reporting" on page 1-30
- "Recovering Data After an Abnormal Termination" on page 1-30

When the MATLAB Program Terminates Unexpectedly

In the event MATLAB experiences a segmentation violation (segv) or other serious problem, the MATLAB System Error dialog box opens to notify you about the problem. When this occurs, the internal state of MATLAB is unreliable and not suitable for further use. You should exit as soon as possible and then restart. However, you might want to first try to save your work in progress.

To exit and restart without trying to save your work, follow these steps:

1 If you want to view the stack trace for the problem, click **Details**.

2 Click **Close** to terminate MATLAB.

3 Restart MATLAB. If the Error Log Reporter dialog box opens, send a report to The MathWorks.

To try to save your work in progress before exiting and restarting MATLAB, follow these steps:

1 If you want to view the stack trace for the problem, click **Details**.

2 Click **Attempt to Continue**. MATLAB tries to return to the Command Window or tool you were using.

The Command Window displays the message `Please exit and restart MATLAB` to the left of the prompt, which reminds you to discontinue use.

3 From the Command Window or tool, try to save the workspace and unsaved files.

Caution   Because the internal state of MATLAB might be corrupted, do not save existing files to the same filename. Instead, specify a new filename. The information in the new file might be corrupted or incomplete.

4 Exit MATLAB immediately after saving because any further usage would be unreliable.

5 Restart MATLAB. If the Error Log Reporter dialog box opens, send a report to The MathWorks.

Error Log Reporting

Upon startup, if MATLAB detects an error log generated by a serious problem during the *previous* session, an Error Log Reporter prompts you to e-mail the log to The MathWorks for analysis. The error log contains the stack trace and information about the MATLAB software configuration. If the problem occurs repeatedly, make note of what seems to cause it, look for information about it in the MathWorks Bug Reports database, and if the problem is reproducible, please submit a Service Request via http://www.mathworks.com/support/contact_us/ts/help_request_1.html.

E-Mailing Error Log Reports. There are some situations where the Error Log Reporter will not open, for example, when you start MATLAB with a -r option or run in deployed mode. It also will not open if you selected the option to never send error reports the last time the Error Log Reporter opened. If you experience abnormal termination but do not see the Error Log Reporter on subsequent startups, you can instead e-mail the reports.

Send e-mail to segv@mathworks.com with this file attached: C:\Temp\matlab_crash_dump.####. After you send the log file, delete it or move it to another location. If you do not delete the log file, the Error Log Reporter can detect it on the next startup and prompt you to send it, even though you already e-mailed it.

Recovering Data After an Abnormal Termination

If MATLAB terminates unexpectedly, you might lose information. After you start MATLAB again, you can try these suggestions to recover some of the information.

- Use the Command History or the file on which it is based, history.m, to run statements from the previous session. You might be able to approximately recreate data as it was prior to the termination. For more information, see "Overview of the Command History Window" on page 3-65.

- If you used the diary function or -logfile startup option for the session in which MATLAB terminated unexpectedly, you might be able to recover output.

- If you saved the workspace to a MAT-file during the session, you can recover it by loading the MAT-file. For more information, see "Viewing and Loading a Saved Workspace and Importing Data" on page 5-7, and "Saving the Current Workspace" on page 5-5.

- If you were editing a file in the Editor when MATLAB terminated unexpectedly, and you had the autosave preference enabled, you should be able to recover changes you made to files you had not saved.

- If you were in a Simulink session when a segmentation violation occurred, and you have the Simulink **Autosave Options** preference selected, note that the last autosave file for the model reflects the state of the autosave data prior to the segmentation violation. Because Simulink models might be corrupted by a segmentation violation, a model is not autosaved after a segmentation violation occurs.

Some of the above suggestions refer to actions you might have needed to take during the session when MATLAB terminated. If you did not take those actions, consider regularly performing them to help you recover from any future abnormal terminations you might experience.

2

Desktop

If you have an active Internet connection, you can watch the Working in the Development Environment video demo for an overview of the major desktop functionality. The easiest way to learn to use the desktop is just by working with it. If you have problems or questions, refer to the following sections:

Desktop Overview

In this section...
"About the Desktop" on page 2-2
"Summary of Desktop Tools" on page 2-4

About the Desktop

In general, when you start the MATLAB program, it displays the MATLAB desktop, a set of tools (graphical user interfaces or GUIs) for managing files, variables, and applications associated with MATLAB.

The first time you start MATLAB, the desktop appears with the default layout, as shown in the following illustration.

Menus change, depending on the tool you are using.

Select the title bar for a tool to use that tool.

Get help.

View or change the current directory.

Move, maximize, minimize or close a window.

Click the **Start** button for quick access to tools and more.

Drag the separator bar to resize windows.

Enter MATLAB statements at the prompt.

View or execute previously run statements.
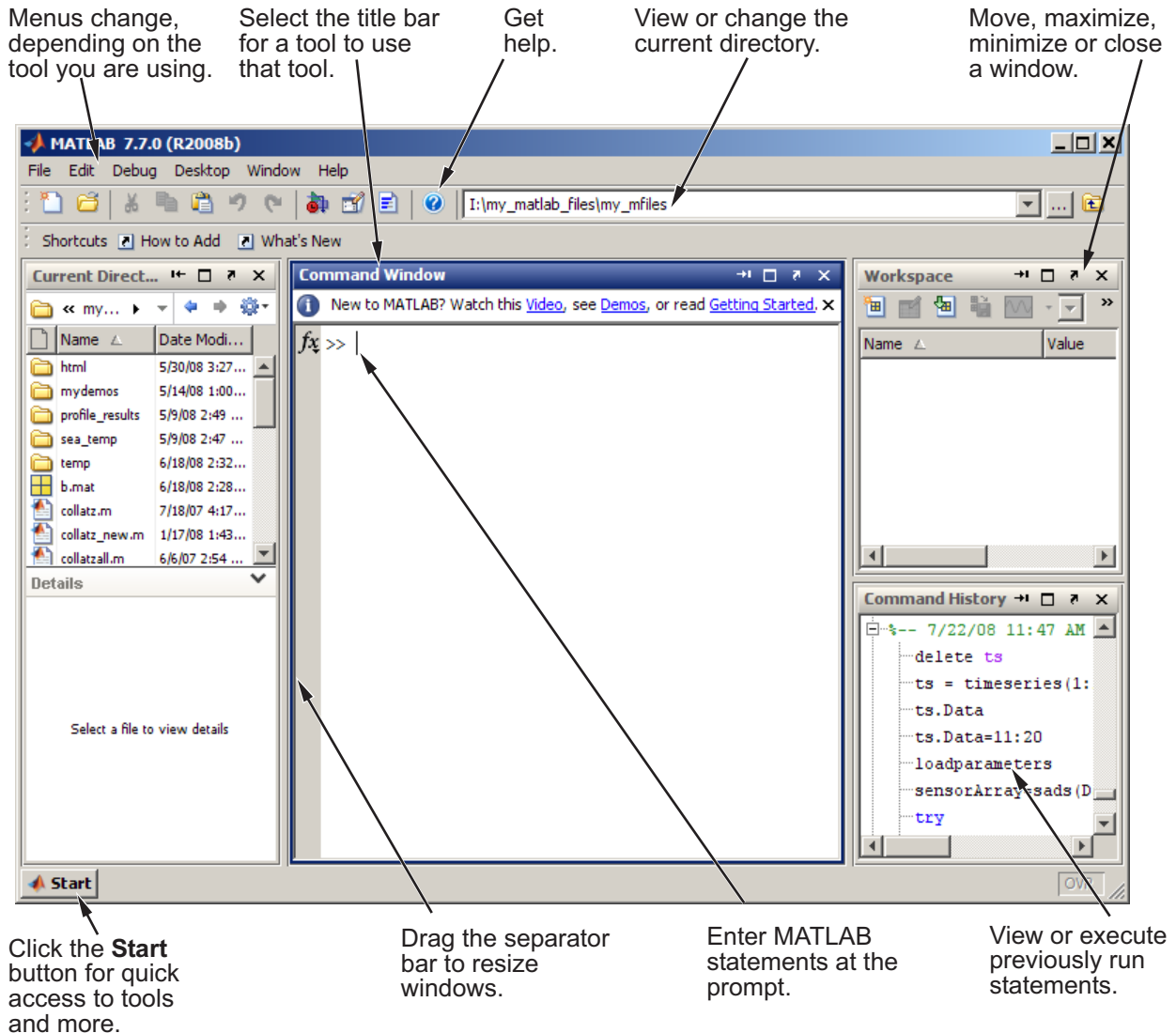
You can change the desktop arrangement to meet your needs, including resizing, moving, and closing tools. The desktop manages tools differently from documents. The Command History and Editor are examples of tools, and an M-file is an example of a document, which appears in the Editor tool. For

details, see "Opening and Arranging Desktop Tools" on page 2-5 and "Opening and Arranging Desktop Documents" on page 2-21.

Summary of Desktop Tools

The following tools are managed by the MATLAB desktop, although not all of them appear by default when you first start. If you prefer a command-line interface, you can often use equivalent functions to accomplish the same results. To perform the equivalent of the GUI tasks in M-files, you must use the equivalent functions. Instructions for using equivalent functions to perform the task are provided with the documentation for each tool and are typically labeled as Function Alternatives.

Desktop Tool	Description
Command History	View a log of or search for the statements you entered in the Command Window, copy them, execute them, and more.
Command Window	Run MATLAB language statements.
Current Directory Browser	View files, perform file operations such as open, find files and file content, and manage and tune your files.
Editor	Create, edit, debug, and analyze M-files (files containing MATLAB language statements).
Figures	Create, modify, view, and print figures generated with MATLAB.
File and Directory Comparisons	View line-by-line differences between two files.
Help Browser	View and search the documentation and demos for all your MathWorks products.
Profiler	Improve the performance of your M-files.
Start Button	Run tools and access documentation for all your MathWorks products, and create and use shortcuts for MATLAB.
Variable Editor	View array contents in a table format and edit the values.
Web Browser	View HTML and related files produced by MATLAB.
Workspace Browser	View and make changes to the contents of the workspace.

Opening and Arranging Desktop Tools

See also "Examples of Desktop Arrangements" on page 2-41.

Opening Desktop Tools

To open a tool, select it from the **Desktop** menu. A check mark in front of the tool name on the menu indicates that the tool is open. The tool opens in the location it occupied the last time you used it. The dimensions of other open tools adjust to accommodate the newly opened tool. If you close and then reopen multiple tools sequentially, the location and size of the tools when you reopen them might be different from when you last closed them.

Tools and the documents associated with them can be part of the desktop. You can open a document and its associated tool at the same time, as follows:

- Variable Editor — Open it by double-clicking a variable in the Workspace browser.

- Editor — Open it by creating a text file, such as an M-file, or opening an existing file. For instructions, see "Starting, Creating Files, and Closing the Editor" on page 6-7.

- Figures — Create figures using `plot` and other graphics functions.

You also can open most desktop tools by:

- Clicking the desktop **Start** button, selecting **Desktop Tools**, and then clicking the tool you want to open.

- Using a function. For example, type `helpbrowser` to open the Help browser. For information on how to open a given tool using a function, see the documentation for that tool.

The following example shows how the MATLAB desktop can look with the Command Window, Command History window, Help browser, and the files `collatz.m` and `lengthofline.m` open in the Editor. Because the Command Window is the active window, its title bar is dark blue.
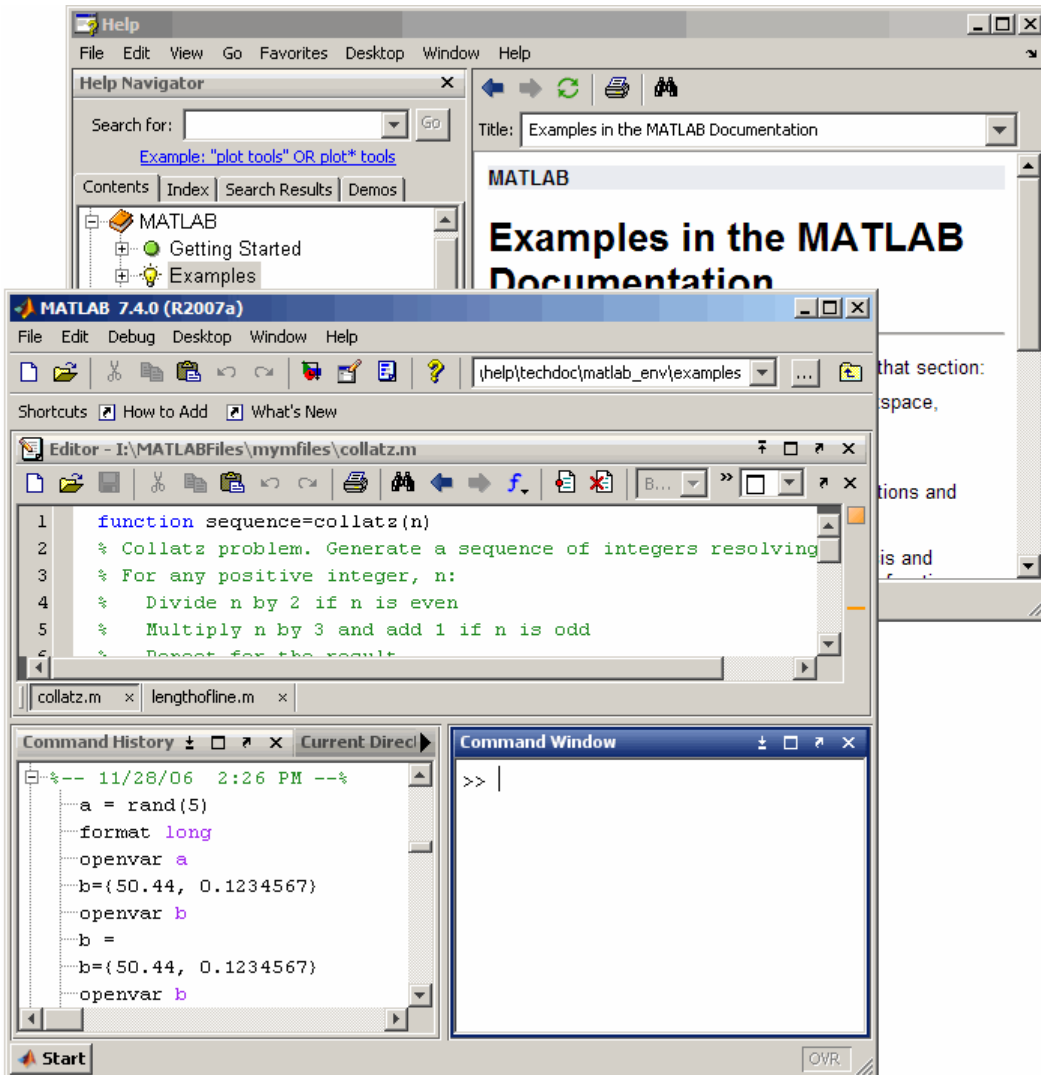
Navigating Among Desktop Tools and Documents

- "Navigating Among Desktop Tools and Documents" on page 2-8
- "Making a Tool or Document the Active Window" on page 2-8

• "Using Window Menu Shortcuts and Mnemonics" on page 2-8

Navigating Among Desktop Tools and Documents

You can navigate among desktop tools and documents by:

• Choosing an entry in the **Window** menu

• Using a function that opens the tool or document

• Clicking the entry for an undocked tool or document on the Microsoft Windows task bar (or the equivalent for your platform)

Making a Tool or Document the Active Window

To make a tool or document the active window, do one of the following:

• Select the tool from the **Window** menu.

  The **Window** menu displays all open desktop tools and documents, as well as opened tools for other MathWorks products.

• Use the shortcut or mnemonic indicated on the **Window** menu for that tool or document.

  See "Using Window Menu Shortcuts and Mnemonics" on page 2-8

• Run the function that opens the tool.

  - If the tool is already open, the command selects the tool.

  - If the tool is not already open, the command opens and selects the tool. For example, type `helpbrowser` to open or select the Help browser. The documentation for each tool includes information on the function for opening and selecting that tool.

Using Window Menu Shortcuts and Mnemonics

The following is an example of the **Window** menu indicating the following are open:

• Command Window

• Command History window

• Help browser

- The Editor along with the files `collatz.m` and `lengthofline.m`

Window Help		
Split Screen		▶
⊞ Tile		▶
⊞ Tile…		
☐ Left/Right Tile		
✓ ⊟ Top/Bottom Tile		
⊡ Float		
☐ Maximize		
Cascade Editor Documents		
Minimize Editor Documents		
Close Editor Documents		
Close All Documents		
Next Tool	Ctrl+Tab	
Previous Tool	Ctrl+Shift+Tab	
Next Tab	Ctrl+Page Down	
Previous Tab	Ctrl+Page Up	
0 Command Window	Ctrl+0	
1 Command History	Ctrl+1	
2 Current Directory	Ctrl+2	
3 Workspace	Ctrl+3	
4 Help	Ctrl+4	
Editor	Ctrl+Shift+0	
A lengthofline.m		
B collatz.m		

To go to the next tool, click Next Tool or press Ctrl+Tab.

To go to the next document, click Next Tab or press Ctrl+Page Down

To go to the Command Window, click it or press Ctrl+0.

To go to the open document, collatz.m, click it or press Alt+W+B.

In this example, as indicated on the menu, you can:

- Make the Command Window active by pressing **Ctrl+0**.

2-9

See also "Performing Desktop Tasks Using Keyboard Shortcuts" on page 2-67.

- Go to the most recently used Editor document by pressing **Ctrl+Shift+0**.

- Go to the file `collatz.m` by pressing **Alt+W**, and then **B**.

  The mnemonic identifier for documents is temporary. For example, if you close `collatz.m` and open it later, the mnemonic could no longer be A.

- Go to the next or previous tool on the desktop menu by clicking **Next Tool** or **Previous Tool**, or by pressing **Ctrl+Tab** or **Ctrl+Shift+Tab**, respectively.

- Go to the next or previous document on the desktop by clicking **Next Tab** or **Previous Tab**, or by pressing **Ctrl+Page Down** or **Ctrl+Page Up**, respectively.

Closing Desktop Tools

To close a desktop tool, do one of the following:

- Select the item on the **Desktop** menu.

  The check mark preceding the tool name on the menu clears and the tool closes.

- Click the Close box ⊠ on the title bar for the tool.

- Select **File > Close *Toolname***.

- Right-click the Microsoft Windows task bar entry for an undocked tool and select **Close**.

- On Windows platforms, use the standard shortcut, **Alt+F4**.

When you close a tool, other tools in the desktop adjust their sizes accordingly.

For tools that contain documents, all documents in that tool close, as well. For the Editor, a dialog box appears asking you to save any documents that have unsaved changes. If you do not want to see that dialog box or save any unsaved changes, hold **Ctrl** and click the Close box.

Resizing Desktop Tools

To resize tools on the MATLAB desktop, you can use the mouse or the keyboard, as described in the following sections:

- "Resizing Desktop Tools Using the Mouse" on page 2-11
- "Resizing Desktop Tools Using the Keyboard" on page 2-11

Resizing Desktop Tools Using the Mouse

To expand or reduce the size of adjacent tool windows, use the pointer to drag the bar that appears between them. This bar is the separator bar. When you move the pointer onto the separator bar, the pointer assumes a different shape, as follows:

- On Windows platforms, when the pointer is between two tools or documents, it is a double-headed arrow ↔.
- On UNIX platforms, when the pointer is between two tools or documents, it is an arrow with a bar. →|
- When the pointer is between three or four documents, it is a four-headed arrow ⊕.

Drag the separator bar to resize tools in the desktop.
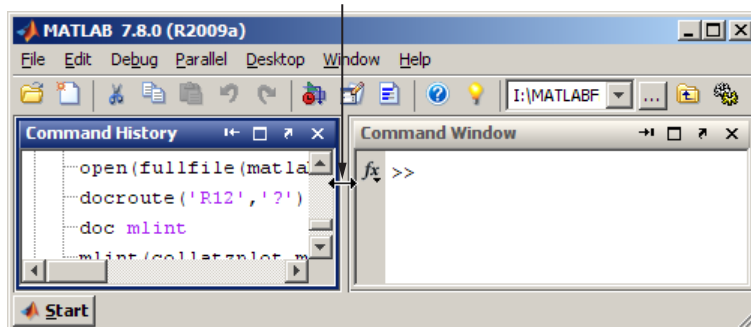
Resizing Desktop Tools Using the Keyboard

You can use menu item mnemonics to resize desktop tools using the keyboard.

For example, suppose the Command Window is open on the desktop along with other tools. To make the Command Window the active tool:

1 Press **Ctrl+0** or click in the Command Window.

2 Press **Alt+D, Z**. This action is the keyboard equivalent of selecting **Desktop > Resize Command Window**.

The pointer shape becomes an arrow.

3 Use the keyboard arrow keys to change the size of the Command Window.

4 Press **Enter** to accept the new size, or press **Esc** to return the Command Window to its original size.

Moving Tools Within the Desktop

To move the location of tools on the MATLAB desktop, use the mouse or the keyboard, as described in the following sections:

- "Moving Tools Using the Mouse" on page 2-12
- "Moving Tools Using the Keyboard" on page 2-14

Moving Tools Using the Mouse

To move a tool to another location on the MATLAB desktop using the mouse, follow these steps:

1 Drag the title bar of the tool to where you want the tool to be.

As you drag the tool, an outline of it appears. The status bar states where the tool moves if you release the mouse. For instance, it might display one of the following:

- Release the mouse to doc Editor on the top.
- Release the mouse to tab-doc Current Directory.
- Release the mouse to leave Editor in the current location.

2 When the outlined position is where you want the tool to be, release the mouse button.

The tool stays at the new location.

In the following example, the Command History tool is to the left of the Command Window. The illustration shows how it looks as you drag it above the Command Window. When the top of the Command History window touches the bottom of the toolbar, the outline appears. If you release the mouse button, you change the arrangement from side-by-side to top-bottom.

To move the Command History window, drag its title bar.

When the outline of the Command History window appears where you want it, release the mouse.

The status bar displays instructions about moving the tool.
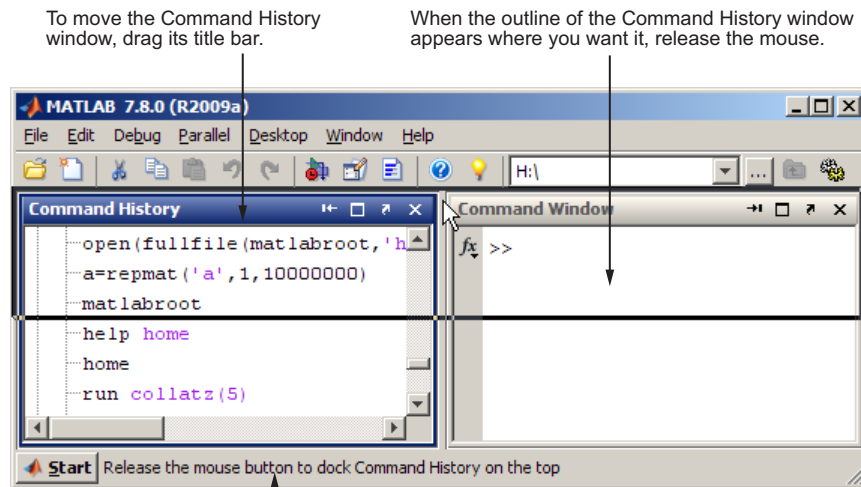
Other tools on the desktop automatically resize to accommodate the new configuration. The following example shows how the desktop looks after the Command History window moves above the Command Window.
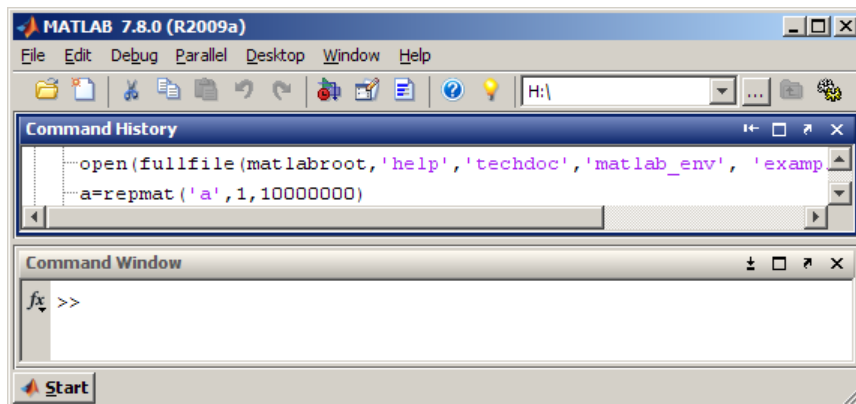
Moving Tools Using the Keyboard

To move desktop tools using the keyboard, follow the menu item mnemonics. For example, suppose the Command Window and other tools are currently open on the desktop. To move the Command Window to a new location, follow these steps:

1 Make the Command Window the active tool by pressing **Ctrl+0**.

2 Press **Alt+D, V**, which is the mnemonic equivalent for selecting **Desktop > Move Command Window**.

   The pointer shape becomes an arrow.

3 Use the arrow keys to move the outline of the Command Window to a new location.

4 Press **Enter** to keep the tool at the new location, or press **Esc** to return the Command Window to its original position.

Undocking Tools to Move Them Outside the Desktop

You can move a tool outside the MATLAB desktop (called undocking) to make it larger or easier to use. For example, when referring to the online documentation, you might want to move the Help browser off the desktop. .

To move a tool outside the desktop:

1 Select the tool to make it active.

2 Perform one of the following:

- Click the Undock button in the title bar of the tool you want to move outside the desktop.
- Select **Undock** for that tool from the **Desktop** menu; the tool must be the currently active one.
- Drag the title bar of the tool outside the desktop. As you drag, an outline of the tool appears. Release the mouse.

The tool displays outside the MATLAB desktop and an entry for it appears in the Windows task bar or the equivalent for your platform. Tools within the desktop resize accordingly.

Moving Undocked Tools Back onto the Desktop

To move a tool that is outside the MATLAB desktop back onto the desktop, do one of the following:

- Click the Dock button in the menu bar for that tool.
- Select **Dock** from the **Desktop** menu for that tool.

The tool moves onto the desktop and other tools within the desktop automatically resize to accommodate the new tool.

Grouping Desktop Tools Together

You can group tools so that they occupy the same location on the MATLAB desktop. Basically, you are stacking one tool on top of another. Then, you can access the individual tools using the tool name on the title bar:

To group tools:

1 Drag the title bar of one tool on the desktop on top of another tool on the desktop.

An outline of the tool you are dragging overlies the target tool.

Outline of the Command History window being dragged on top of the Command Window
to group both tools together
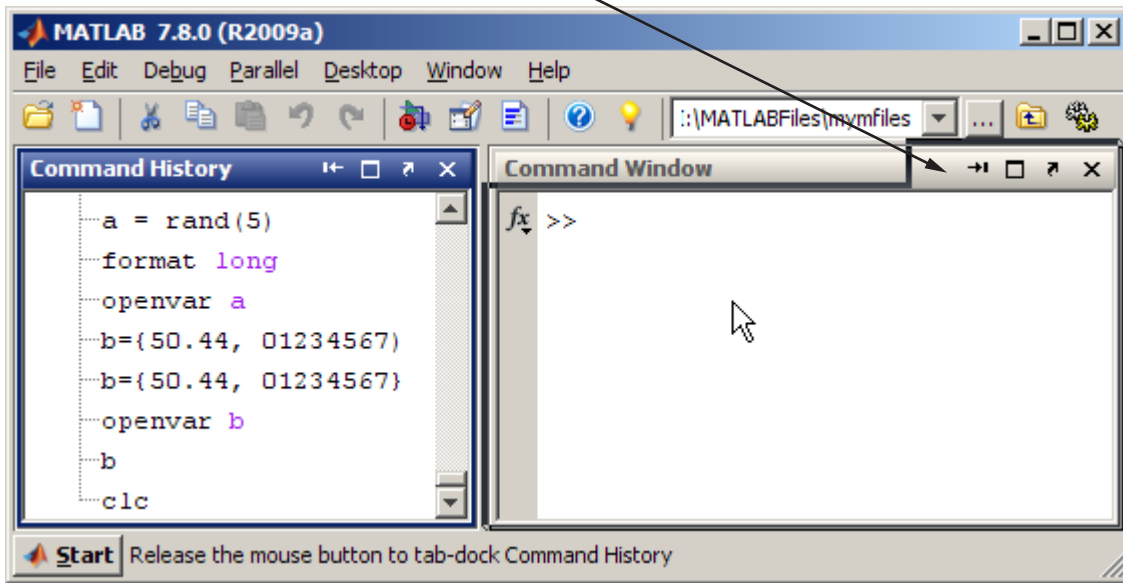
2 Release the mouse.

Both tools occupy the same space. Labeled tabs appear at the top of that
space.
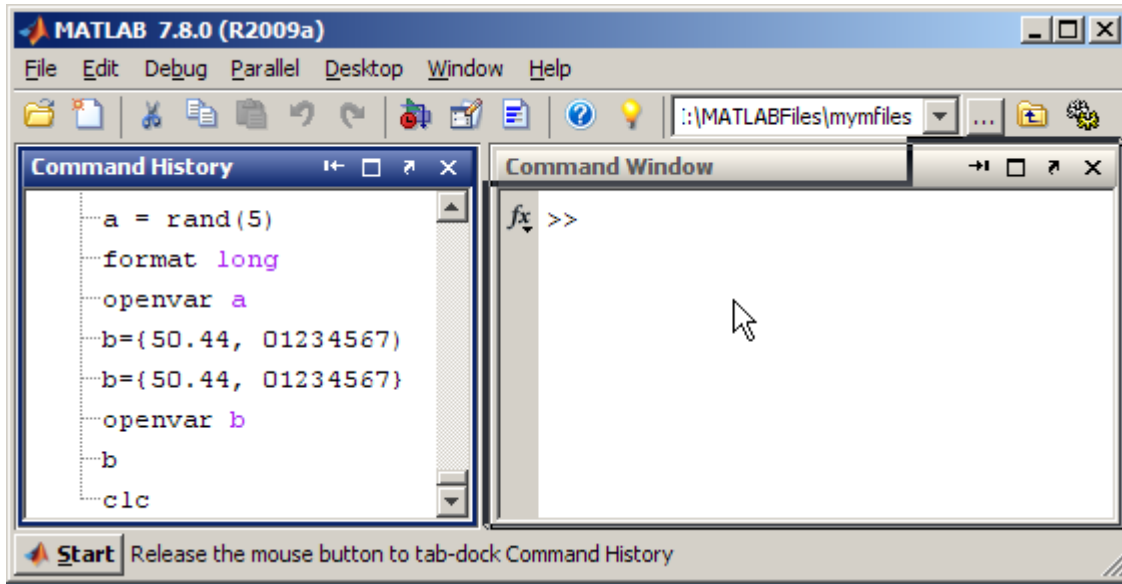
To view a grouped tool, click the tool's title bar. The selected tool moves to the foreground and becomes the currently active window.

When you click the Close box ⊠ for a tool grouped with other tools, that tool closes. You cannot close all the grouped tools at once. Instead, close each tool individually.

Right-click the title bar for a tool and use the context menu to close, undock, maximize, or minimize the tool.

Maximizing Available Space on the Desktop

To hide the title bars for desktop tools so they use less space, select **Desktop > Titles**. This action clears the check mark next to the **Titles** menu item. Identify a desktop tool with a hidden title by hovering over the area where the title bar used to be. A ToolTip displays the name of the tool.

Maximizing Tools Within the Desktop

MATLAB software provides multiple ways to maximize tools on the desktop. It also has multiple ways of restoring the desktop to the layout in place before you resized it. For example:

- To resize the active tool so it occupies the entire MATLAB desktop do one of the following:
  - Double-click the title bar in that tool.
  - Select **Desktop > Maximize Toolname**.
  - Click the Maximize button ■ on the tool title bar.

- To return to the layout as it appeared before you maximized it, do one of the following:
  - Double-click the maximized title bar for that tool.
  - Select **Desktop > Restore Toolname**.
  - Click the Restore button ▣ on the title bar in that tool.

Minimizing Tools Within the Desktop

You can minimize any tool on the desktop, which creates a button representing the tool along an edge of the desktop. For example, suppose you minimize the Command Window. The desktop looks like the following, although the layout of the desktop and the location of the button can be different for your desktop.

Command Window button
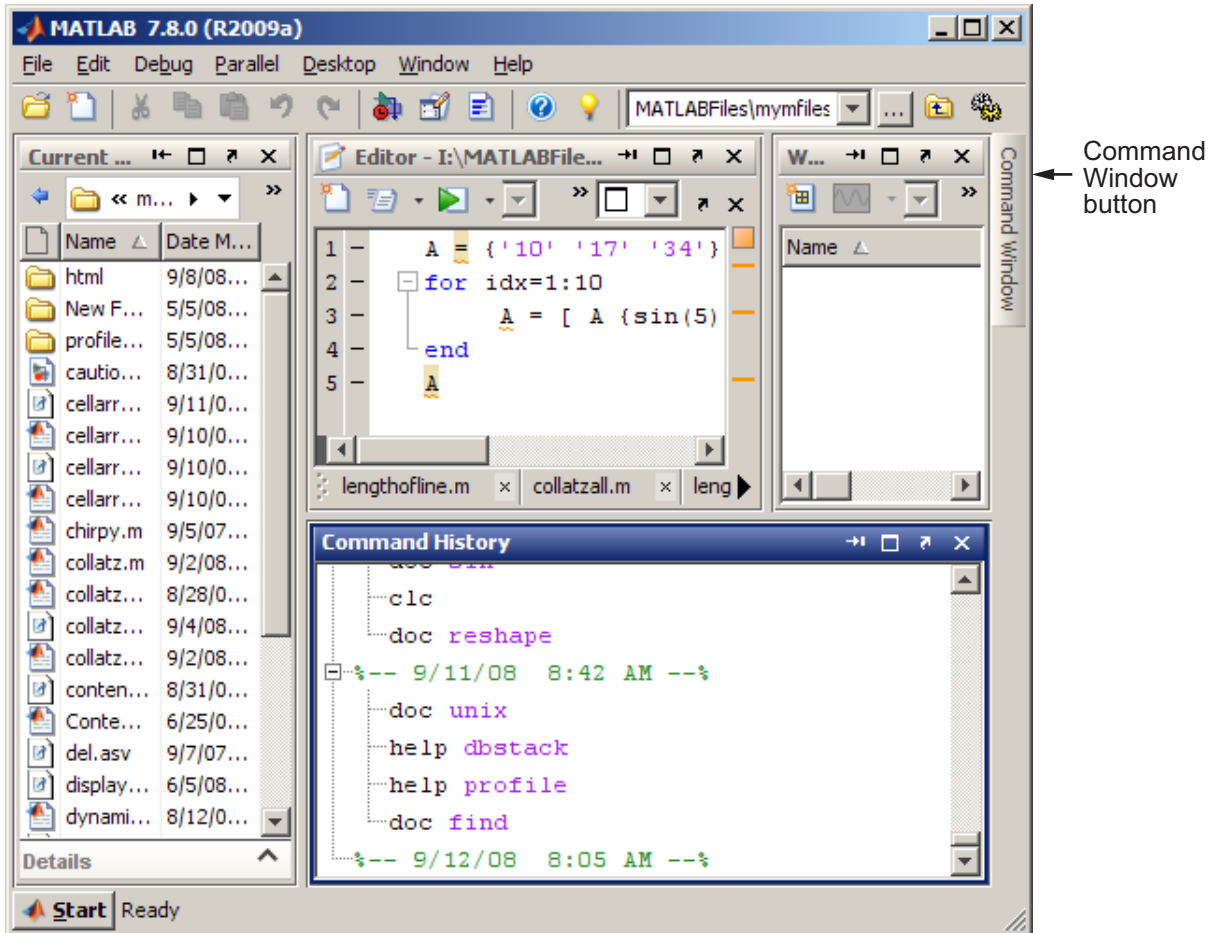
MATLAB software provides multiple ways to minimize tools in the desktop, restore the previous desktop layout, and manipulate the location of the tool button. For example:

- To minimize a tool, do one of the following:
  - Select **Desktop > Minimize Toolname**.
  - Use the Minimize button ⊟ on the title bar for the tool.

The button for the tool appears along the edge indicated by the minimize arrow in the **Desktop** menu item or by the arrow on the button.

- To view or use a minimized tool, hover over or click the button for the tool. This action temporarily opens the tool on the desktop. When you finish using the tool, click the Minimize button or another tool. The tool appears again as a button along the edge of the desktop.

- To return the tool to the position it occupied before you minimized it, do one of the following:

  - Double-click the button for the tool.

  - Right-click the button for the tool, and then select **Restore > Toolname**.

  - Hover over or click the button for the tool, and then click the Restore button on the title bar of the tool.

- To move the button for the tool, drag it to a different edge.

  If you drag the button to a nonedge location on the desktop or outside of the desktop, it moves the tool and opens restores it.

Opening and Arranging Desktop Documents

In this section...
"Opening Documents" on page 2-21
"Navigating Among Open Documents Using the Document Bar" on page 2-24
"Adjusting the Document Bar" on page 2-25
"Positioning Documents" on page 2-26
"Moving and Resizing Documents" on page 2-33
"Closing Documents" on page 2-34
"Moving Documents Outside of the Desktop (Undocking)" on page 2-37
"Docking Documents and Tools" on page 2-37
"Grouping Documents in a Tool Outside the Desktop" on page 2-37

Opening Documents

Use the document bar to go to a document that is open, but not in view. The names of all open documents appear on the document bar. Click the document name to open the document. If the document bar is not open, select **Desktop > Document Bar > Bar Position** and select the position for it, for example, **Right**. For more information, see "Navigating Among Open Documents Using the Document Bar" on page 2-24.

Entries for undocked documents appear on the Windows task bar, or the equivalent for your platform. Click the task bar entry for a document to make that document active.

When you open MATLAB documents, they open in the associated tool and appear in the position they occupied when last used. Figures open undocked, regardless of the last position occupied. If the tool is not already open, it opens when you open the document.

How you open a document depends on the document type, as described in the following table.

Document Type and Tool	How to Open Document	Where Document Appears by Default	Other Techniques to Open Document
M-file (or other text file) in the Editor	Click the **Open file** button 📂 on the desktop toolbar and select the file.	In the last location of the Editor. The default location for the Editor is outside the desktop.	"Opening Existing Files in the Editor" on page 6-10
Variable in the Variable Editor	Double-click a variable in the Workspace browser.	In the last location of the Variable Editor. The default location of the Variable Editor is docked on the desktop.	"Opening the Variable Editor" on page 5-14
HTML or similar page in the Web browser	Double-click the file name in the Current Directory browser.	In the last location of the Web browser, replacing the existing Web browser document.	"Displaying Documents and Web Sites in Web Browsers" on page 2-93
Figure	Use the `plot` function.	In a figure window, outside the desktop.	Any other function or tool that creates a figure window.

Example of Working with Documents on the Desktop

Some common actions for working with documents on the desktop are:

- Select a document from the document bar, making it the active open document.

- Use the **Window** menu or equivalent toolbar buttons to position documents.

- Close or undock a tool, including all documents in the tool.

- Undock a document from its tool.

- Use the document Close box with the **Ctrl** key to close the document without saving it and without displaying the unsaved document dialog box.

See also "Examples of Desktop Arrangements" on page 2-41.

Close, undock, or resize tool (Editor),
including its open documents.

Position documents within the
tool using these options.

Undock document
from tool, or close
document.

To go to an
opendocument,
click its
name in the
document bar.

To go to an
opentool, click
its name in
the title bar.
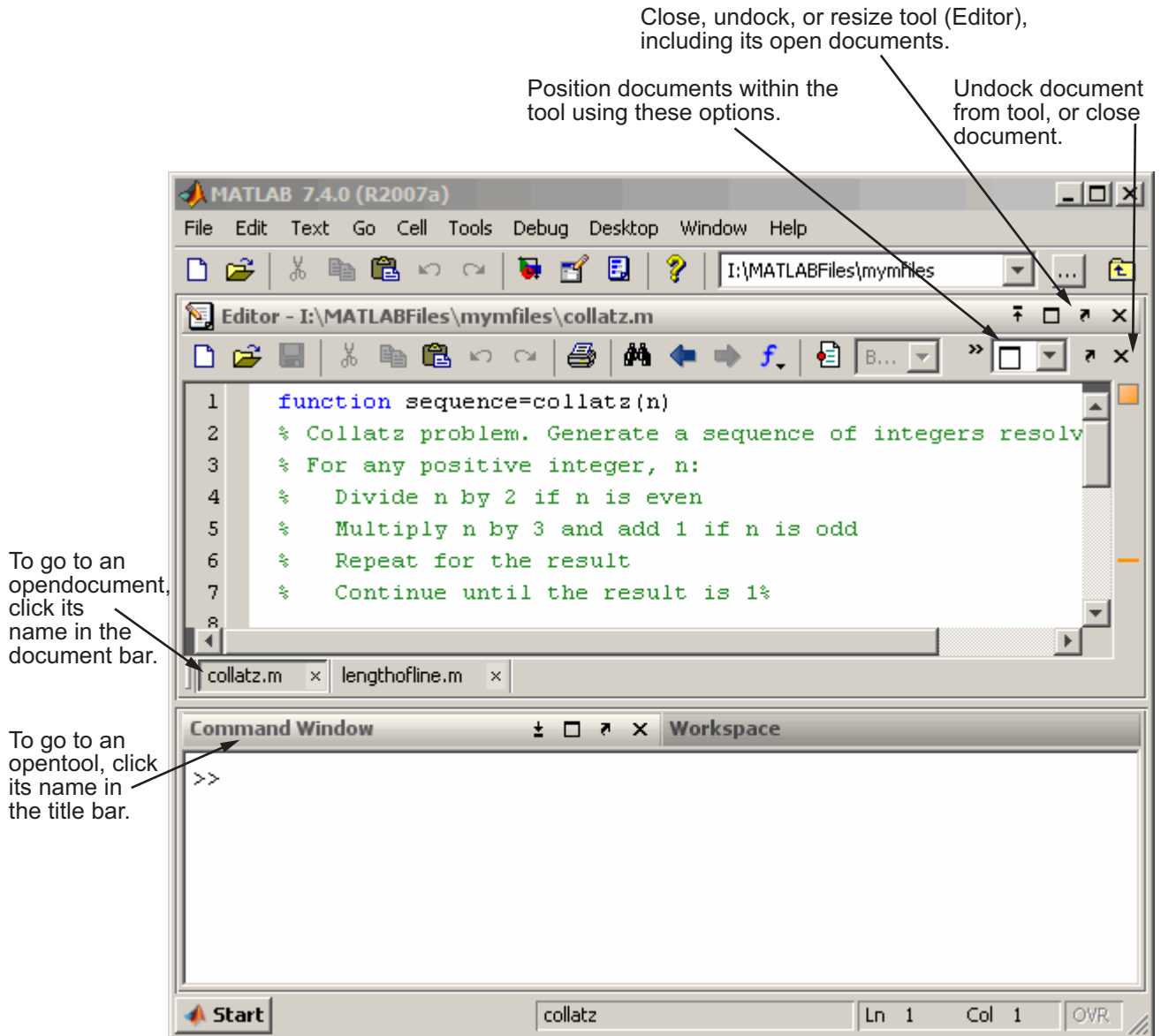
Navigating Among Open Documents Using the Document Bar

When you have more than one document open in a tool, each document appears either maximized (the default), tiled, or floating (cascading). Tiled and floating arrangements make multiple documents visible simultaneously. The document bar shows the names for all open documents docked together in a tool.

Making a Document Active

To make a document that is open and in view active, click it.

To make an open document that is not in view active, do one of the following:

- Select the document from the document bar.

  If all the document names do not fit on the document bar, use the **More Documents** button ▶ on the document bar. This button enables you to see the names of additional open documents. Hover over the arrow to automatically scroll through all the names, or click the arrow to move quickly through the names.

- From the **Window** menu, select the document name.

- From the **Window** menu, select **Next Tab** to make the next document on the document bar active (relative to the currently active document).

- From the **Window** menu, select **Previous Tab** to make the previous document on the document bar active (relative to the currently active document).

See also "Performing Desktop Tasks Using Keyboard Shortcuts" on page 2-67.

Adjusting the Document Bar

You can show, hide, move, alphabetize, and adjust the size of the tabs in the document bar as described in the following table.

To Accomplish This:	Do This:
Show the document bar.	Select **Desktop > Document Bar > Bar Position**, and then select a location, for example, **Right**.
Hide the document bar.	Select **Document Bar > Bar Position > Hide** from the **Window** menu or the document bar context menu.
Move the document bar.	Do one of the following: • Drag it to another location • Select a new location from the **Desktop > Document Bar > Bar Position** submenu.
Alphabetize the names of the documents on the document bar. Alphabetizing is useful if you have many documents open at once.	Do one of the following: • Right-click on the document bar and select **Alphabetize**. • Select **Desktop > Document Bar > Alphabetize**.
Reorder document names on the document bar.	Do one of the following: • Drag a document name to a different position in the document bar. • Select **Move documentname On Bar** and select a direction. For example, select **to Beginning** from either the **Desktop > Document Bar** menu or from the document bar context menu.
Widen or narrow document names on the document bar. If document names are long, or if you have many documents open, the entire document name does not display. Instead, you see the first	Do one of the following: • When the document bar is on the top or bottom drag the separator bar between two names on the bar. (Do this, for example, to see an entire document name.)

To Accomplish This:	Do This:
few characters followed by ellipsis (. . .).	• When the document bar is on the left or right, change the width of the bar by dragging its left or right edge.

Positioning Documents

You can position open documents so that one document or multiple documents are in view from within a tool. Select the arrangement from the **Window** menu or use the Arrange Documents drop-down menu □ ▾, as described in the sections that follow. When you *tile* documents, they are all visible within the tool, arranged in a grid pattern.

- "Viewing One Document (Default)" on page 2-26
- "Viewing All Open Documents, Layered on Top of One Another" on page 2-26
- "Viewing Documents, Side-By-Side" on page 2-27
- "Viewing Open Documents, One Above the Other" on page 2-27
- "Viewing Open Documents, Tiled Within the Tool" on page 2-27
- "Viewing a Subset of Open Documents, Tiled Within the Tool" on page 2-31

Viewing One Document (Default)

To have one document in view that occupies the entire tool (the default), do one of the following:

- Select **Window > Maximize**.
- From the **Arrange Documents** drop-down menu, choose the Maximize option □.

The illustration in "Example of Working with Documents on the Desktop" on page 2-22 shows this arrangement.

Viewing All Open Documents, Layered on Top of One Another

You can use the **Float** or **Cascade** options to layer open documents one on top of another. To do this, use one of the following methods:

- Select **Window > Float**.

- In the **Arrange Documents** drop-down menu, choose the **Float** option ⊡.

  Optionally, select **Window > Cascade** to make the document arrangement neater.

Viewing Documents, Side-By-Side

You can use the **Tile** option to view two documents side-by-side. To do this, use one of the following methods:

- Select **Window > Left/Right Tile**.

- In the Arrange Documents drop-down menu, choose the Left/Right Tile option ⊓.

See also the Editor "Split Screen Display" on page 6-61 that enables you to view two different parts of the same file simultaneously.

Viewing Open Documents, One Above the Other

You can use the **Top/Bottom Tile** option to view two documents stacked one above the other. To do this, use one of the following methods:

- Select **Window > Top/Bottom Tile**.

- In the Arrange Documents drop-down menu, choose the Top/Bottom Tile option ⊟.

See also the Editor "Split Screen Display" on page 6-61 that enables you to view two different parts of the same file simultaneously.

Viewing Open Documents, Tiled Within the Tool

To have all open documents in view, tiled within the tool, follow these steps:

1 Select the tiling option using one of these methods:

- Select **Window > Tile**.

  On the Apple Macintosh platform, this option might be unavailable, so use the drop-down menu instead.

- In the **Arrange Documents** drop-down menu, choose the Tile option ⊞.

  A four-by-four grid displays.

2 Move the pointer across the grid to define the number and position of the tiles, as shown in the following illustration.

  You can select more or fewer tiles than there are open documents. In the example, there are three open documents, but you must select four tiles to make a square grid shape. The tiles that will contain documents appear blue, whereas the tiles that will be empty appear gray.

This example shows how to select an arrangement so that all three documents will be in view. The resulting arrangement has two documents above, one below, and one empty tile.

```
Window
    Split Screen                        ▶
    ▦  Tile                             ▶
    ▦  Tile...
    ▯▯ Left/Right Tile
    ▭  Top/Bottom Tile
    ▱  Float
    ▢  Maximize

    Cascade Editor Documents
    Minimize Editor Documents
    Close Editor Documents
    Close All Documents

    Next Tool            Ctrl+Tab
    Previous Tool        Ctrl+Shift+Tab
    Next Tab             Ctrl+Page Down
    Previous Tab         Ctrl+Page Up

    0 Command Window     Ctrl+0
    3 Workspace          Ctrl+3
    4 Help               Ctrl+4

    Editor               Ctrl+Shift+0

    A collatz.m
    B collatzplot.m
    C lengthofline.m
```

Cancel

The following arrangement shows three documents tiled in the Editor. The Editor is undocked from the desktop.
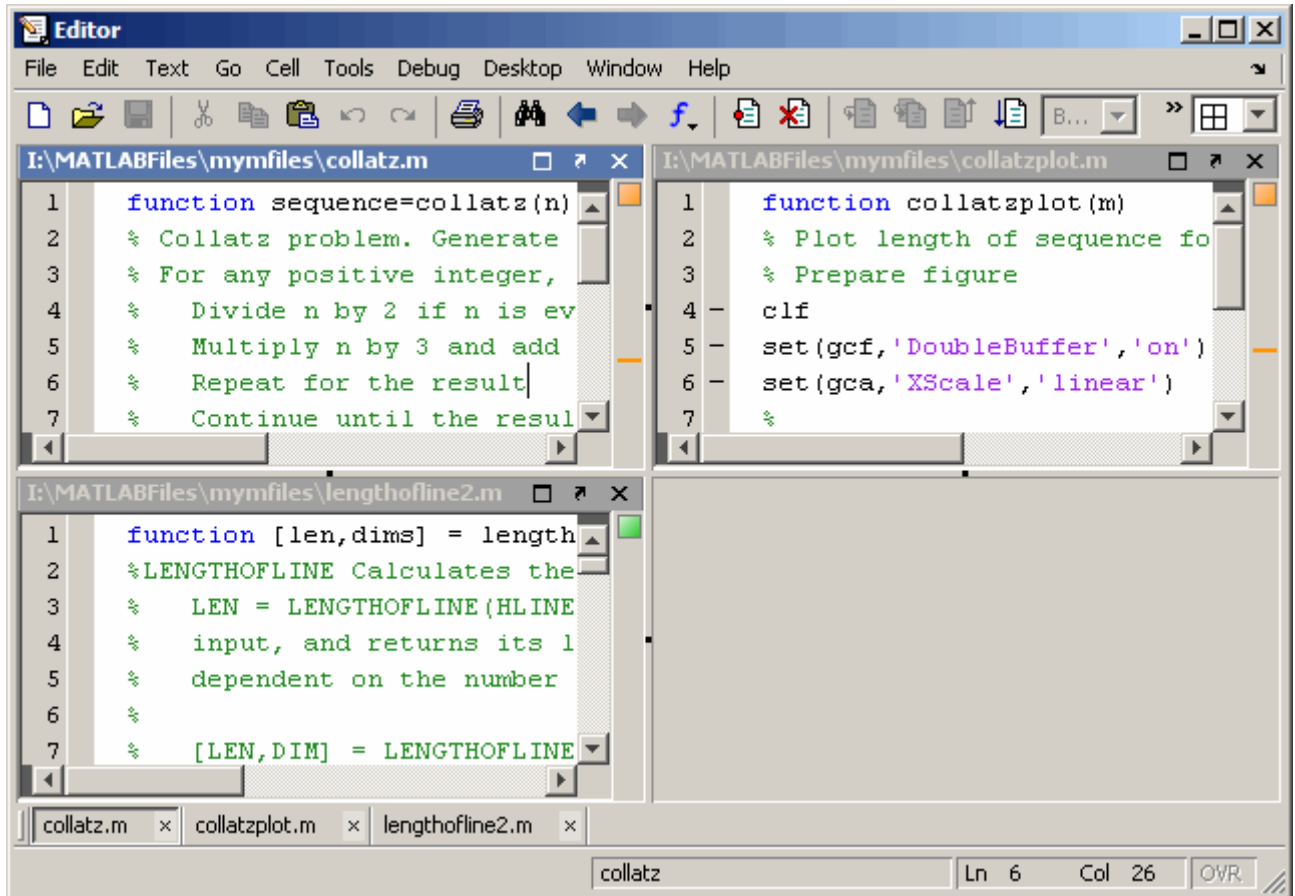
To close an empty tile, move the pointer over the handle ▪ on the separator bar, and then click the Close box that appears. If you click the handle between two open documents, both documents stay open, but one moves on top of the other.

Handle on separator bar.

To close an empty tile, move the pointer onto the handle on the separator bar and click the Close box that appears.
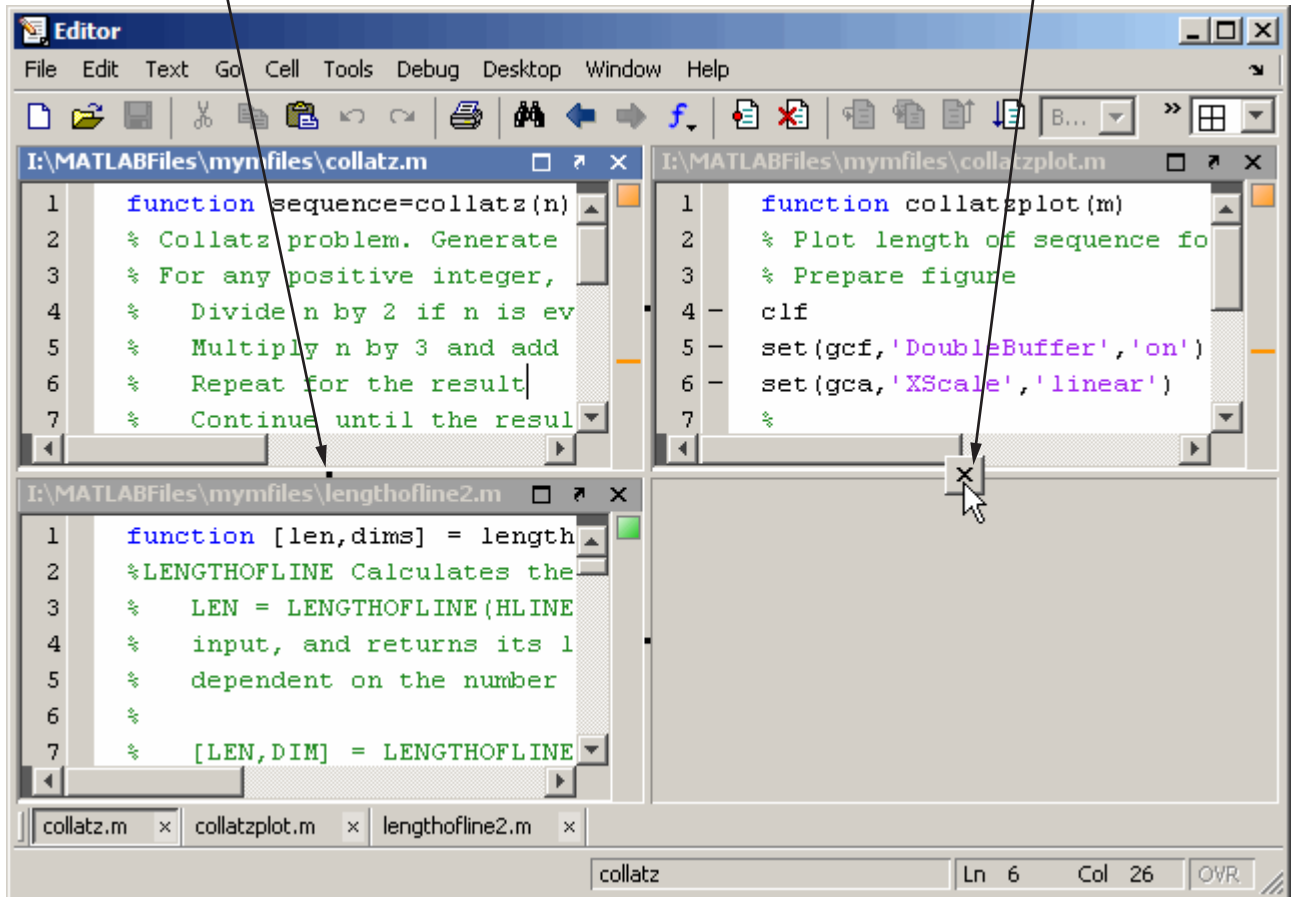
Viewing a Subset of Open Documents, Tiled Within the Tool

To view only a subset of all your open documents displayed in tile format, follow these steps:

1 Select **Window > Tile ...**.

The Tile dialog box opens.

2 Indicate the documents you want to view and the grid pattern to use for the arrangement of their display.

The following illustrations show how to specify the arrangement for three variables in three rows in the Variable Editor, and the resulting configuration.
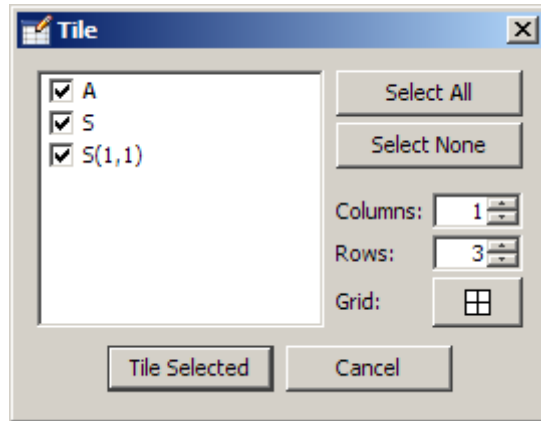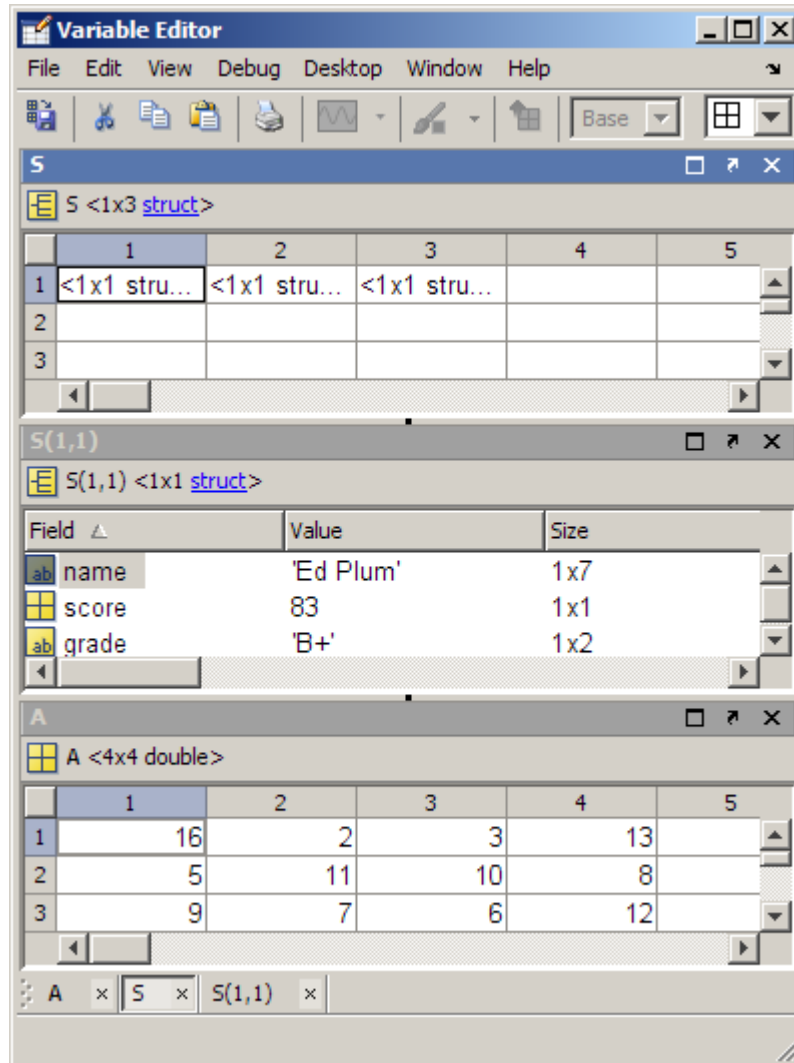
Moving and Resizing Documents

You can move and resize documents to organize them as you want, as
described in the following table.

To Accomplish This:	Do This:
Minimize all open documents in a tool	Make that tool active, and then select **Window > Minimize *Toolname* Documents**
Float documents	Select **Windows > Float**.
Minimize (hide) a floating document	Click the minimize button ▬ in the document title bar.
Access a minimized document	Select its name from the document bar or the **Window** menu.
Move or resize a maximized document	Move or resize the tool that contains it.
Make a document larger when it is next to an empty tile	Hover over the handle on the separator bar, and then click the Close box that appears.
Resize tiled documents	Drag the separator bar that is between the documents.
Move tiled documents	Drag the title bar of the document to another tile. If you drag it to a tile that already contains a document, the document you are dragging covers up the other document.

Closing Documents

There are many ways to close a document. Use any of the following methods:

- Click the Close box ☒ in the title bar for the tool.

- Right-click the title bar for the document, and then select the **Close** option.

- Right-click the name of the document in the document bar, and then select one of the **Close** options.

- Click the Close box ☒ next to the name of the document in the document bar.

- Place the mouse pointer over the name of the document in the document bar, and then click the middle mouse button. (Works on Microsoft Windows and Linux® only.)

- Select **Window > Close *Toolname* Documents**.

- Select **File > Close *Current_Document_Name***.

- Press **Ctrl+F4** on Windows platforms.

- For an undocked document or tool, right-click the Windows task bar entry (or equivalent for your platform) and select **Close**.

- When there are open documents, undocked from within their tools, close all open documents and the tool by selecting **Window > Close All Documents** from the desktop.

For example, in the undocked Editor, select **Window > Close Documents** to close all documents in the Editor. The Editor remains open with no documents in it, and any undocked Editor documents remain open.

To close a document, click the close box in the document's toolbar.

To suppress the save confirmation dialog box, which closes documents without saving changes, hold **Ctrl** while clicking the close box.

To close a tool and all the documents within it, click the close box in the tool's title bar.

Any undocked documents remain open.
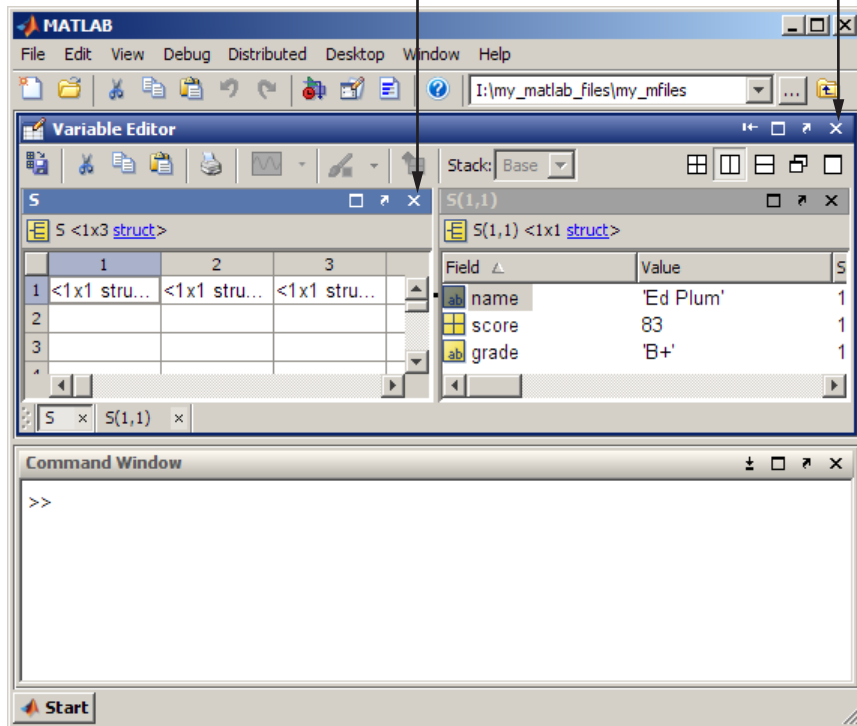
In the Editor, when you close a file that has unsaved changes, a prompt appears asking if you want to save the document. To close a file without saving changes and without seeing the save prompt, use **Ctrl** when you click the Close box for the document.

Moving Documents Outside of the Desktop (Undocking)

You can move a tool outside of the MATLAB desktop (called undocking) to make it larger or easier to work with. For example, you might want to move the Help browser outside of the desktop when referring to the online documentation.

To move a tool outside the desktop, do one of the following:

• Click the Undock arrow ▣ on the title bar of the tool you want to move outside the desktop.

• Make the tool you want to move outside the desktop active. Then for that tool, select **Desktop > Undock**.

• Drag the title bar of the tool outside the desktop. As you drag the title bar, an outline of the tool appears outside of the desktop. When the outline appears where you want the tool to be, release the mouse.

The tool displays outside the MATLAB desktop and an entry for it appears in the Windows task bar. Tools within the desktop automatically resize accordingly.

Docking Documents and Tools

To dock documents and their associated tool, click the Dock button ▣ on the menu bar for the tool.

If you dock a tool that includes documents, the tool and the documents within the tool move onto the desktop. For example, when you dock the Editor and it has open files, both the Editor and the documents move onto the desktop.

When you dock a document, it moves to the position in the tool that it occupied before you undocked the document.

Grouping Documents in a Tool Outside the Desktop

To group all the documents for a tool outside of the desktop, undock the tool from the desktop, not just the individual documents.

If you have already undocked all the documents and closed the empty tool that had contained them, follow these steps:

1 Select **Desktop > Dock All in Editor**, for example.

This selection moves all the documents into the tool in the desktop.

2 Undock the tool.

Saving Desktop Arrangements

When you end a session, MATLAB saves the basic desktop arrangement. The next time you start MATLAB, the desktop appears similar to the way you left it. However, some tools, such as the Help browser, Web browser, and Array Editor do not reopen automatically, even if they were open when you ended the last session. You can use startup options to specify tools that you want to open on startup. For example, to have the Help browser open each time you start MATLAB, add `helpbrowser` to a `startup.m` file. For more information, see "Startup Options" on page 1-17.

You can also use predefined layouts, and you can save your own layouts for later reuse.

MATLAB stores the layouts you save as XML files in the preferences directory for MATLAB. The layout last used in a session is `MATLABDesktop.xml`. Type `prefdir` in the Command Window to display the location of the XML files. The `MATLABDesktop.xml` file loads when MATLAB starts and is overwritten when you close MATLAB.

In this section...
"Predefined Layouts" on page 2-39
"Saving, Reusing, and Renaming Desktop Layouts" on page 2-40

Predefined Layouts

Select a predefined MATLAB desktop layout from the **Desktop > Desktop Layout** menu. Document tools already open in the desktop remain open.

- **Default** — Contains the Command Window and the Command History window, with the Current Directory browser and the Workspace browser tabbed together.

- **Command Window Only** — Contains only the Command Window. This menu item makes MATLAB appear as it did in older versions.

- **History and Command Window** — Contains the Command History window and Command Window, side by side.

- **All Tabbed** — All tools opened, maximized, and tabbed together in the desktop.

- **All but Command Window Minimized** — All tools opened and minimized in the desktop, except for the Command Window, which remains maximized. The Editor also remains maximized if it is open and contains a document at the time you choose the **All but Command Window Minimized** menu item.

After selecting a predefined layout, you can move, resize, and open and close tools.

Saving, Reusing, and Renaming Desktop Layouts

You can save the current desktop layout and use it again at a later time, as follows:

1 Select **Desktop > Save > Layout**.

2 Assign a name to the layout in the resulting dialog box, and then click **OK**.

To reuse the layout, select **Desktop > Desktop Layout**, and then select the layout name.

To rename or delete a saved layout:

1 Select **Desktop > Organize Layouts**.

2 In the resulting dialog box, select a layout, click the **Rename** or **Delete** button, and then click **Close**.

When renaming a layout, type the new name over the existing name. You can rename or delete only desktop layouts that you created.

Examples of Desktop Arrangements

About These Examples

Scan the illustrations in the following examples for a desktop arrangement similar to what you want, and then follow the brief instructions to achieve the arrangement. There are many different ways to accomplish the result; these instructions present just one way. The instructions might not apply exactly, depending on how your desktop looks before you start. For details, see "Opening and Arranging Desktop Tools" on page 2-5 and "Opening and Arranging Desktop Documents" on page 2-21.

Tool Outside of Desktop and Other Tools Grouped Inside Desktop Example

This example shows two ways you can increase the size of a tool:

- Move a tool outside of the desktop to increase its size.

  In the illustration that follows, the Help browser is outside of the desktop and made larger. To move a tool outside of the desktop, click the Undock button ◪ in the tool's title bar when the tool is in the desktop.

- Group tools together inside the desktop, and then access a particular tool by clicking that tool's name in the title bar.

  In the illustration that follows, the Command Window, Command History, Workspace browser, and Current Directory browser appear as together as a group. To achieve this arrangement, drag the title bar of one tool on top of the title bar of the tool (or tools) you want to group it with.

Help browser is undocked from desktop to provide a large area for viewing documentation when needed.

Four tools in the desktop are grouped together, providing a large area for working with a given tool.
Click a tool's name in the title bar to make that tool active.
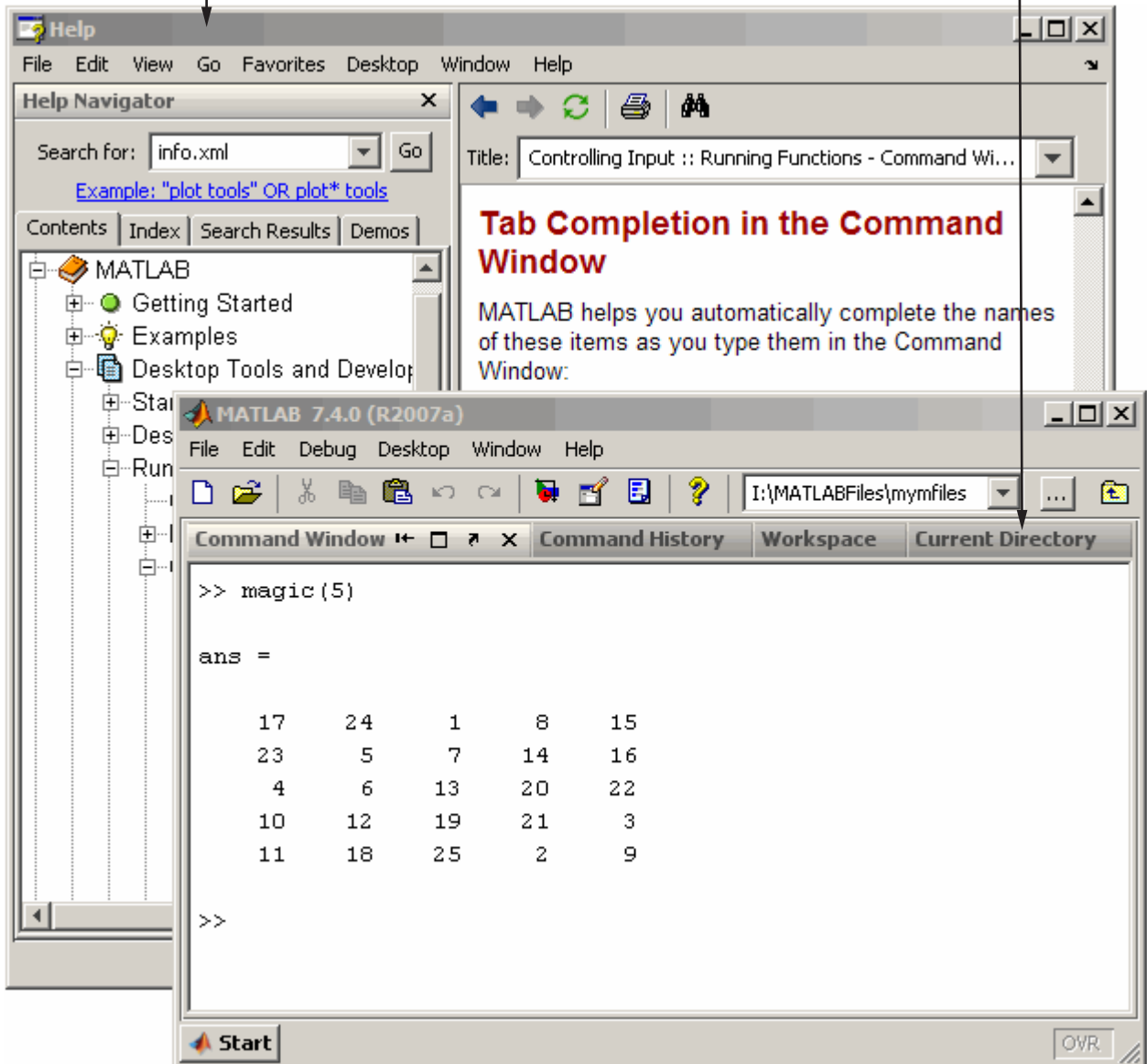
Maximized Tool in Desktop Example

This example shows a way you can temporarily increase the size of a tool so that it occupies the entire area of the desktop. In this example, the Command Window is being temporarily maximized by clicking the Maximize button ■ in the tool's title bar.

Maximize a tool, for example, the Command Window
so it occupies the full MATLAB desktop area.
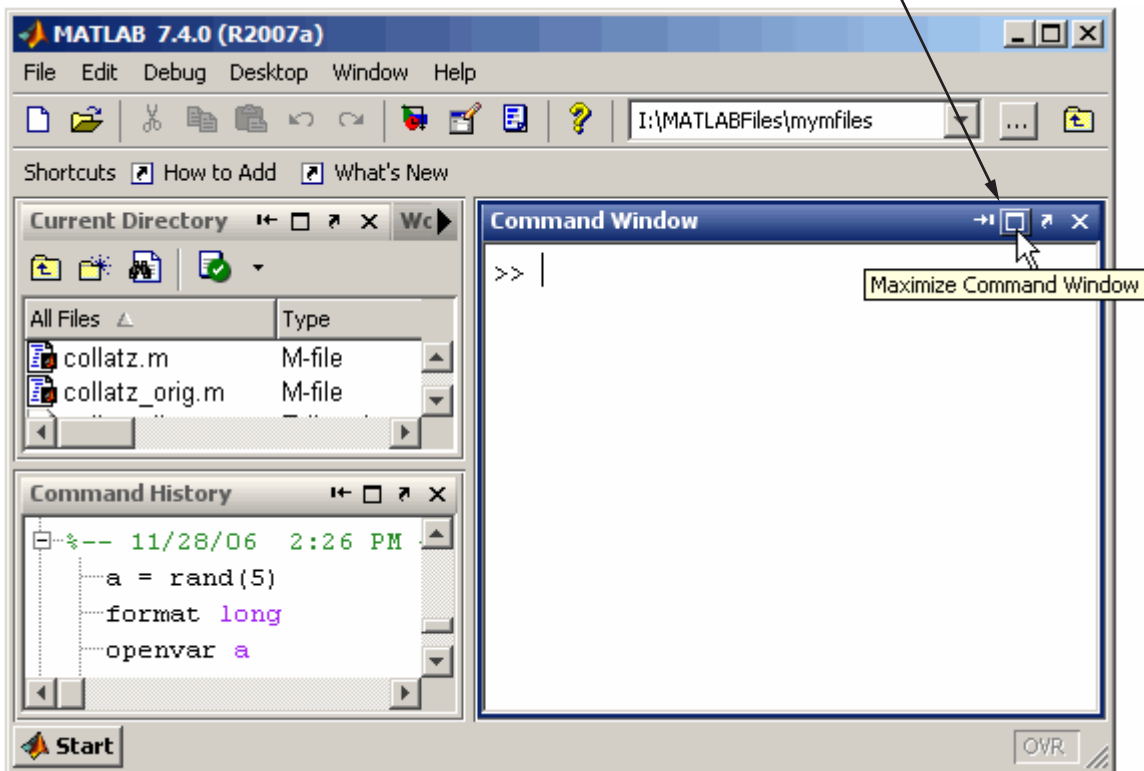
In this example, you return the maximized Command Window to its size and position in the desktop by clicking the Restore button ▣ in the title bar.

Maximized, the Command Window now occupies the full desktop area. Restoring the Command Window returns it to its original size and location in the desktop.
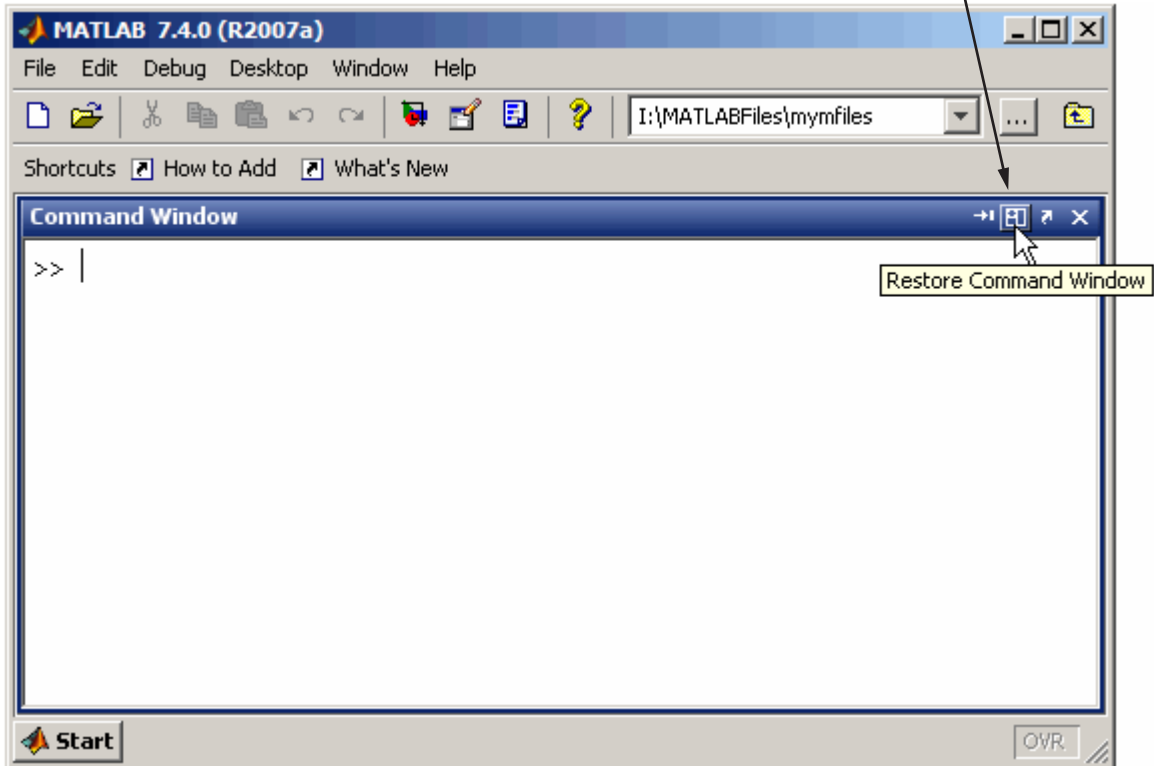
Minimized Tools in Desktop Example

Minimize a tool in the desktop to give the remaining desktop tools more space in the desktop. Minimizing is available on Microsoft Windows and UNIX[6] platforms. This illustration shows the button for minimizing the Command History window to the left edge of the desktop.

Minimize a tool, for example, the Command History window.
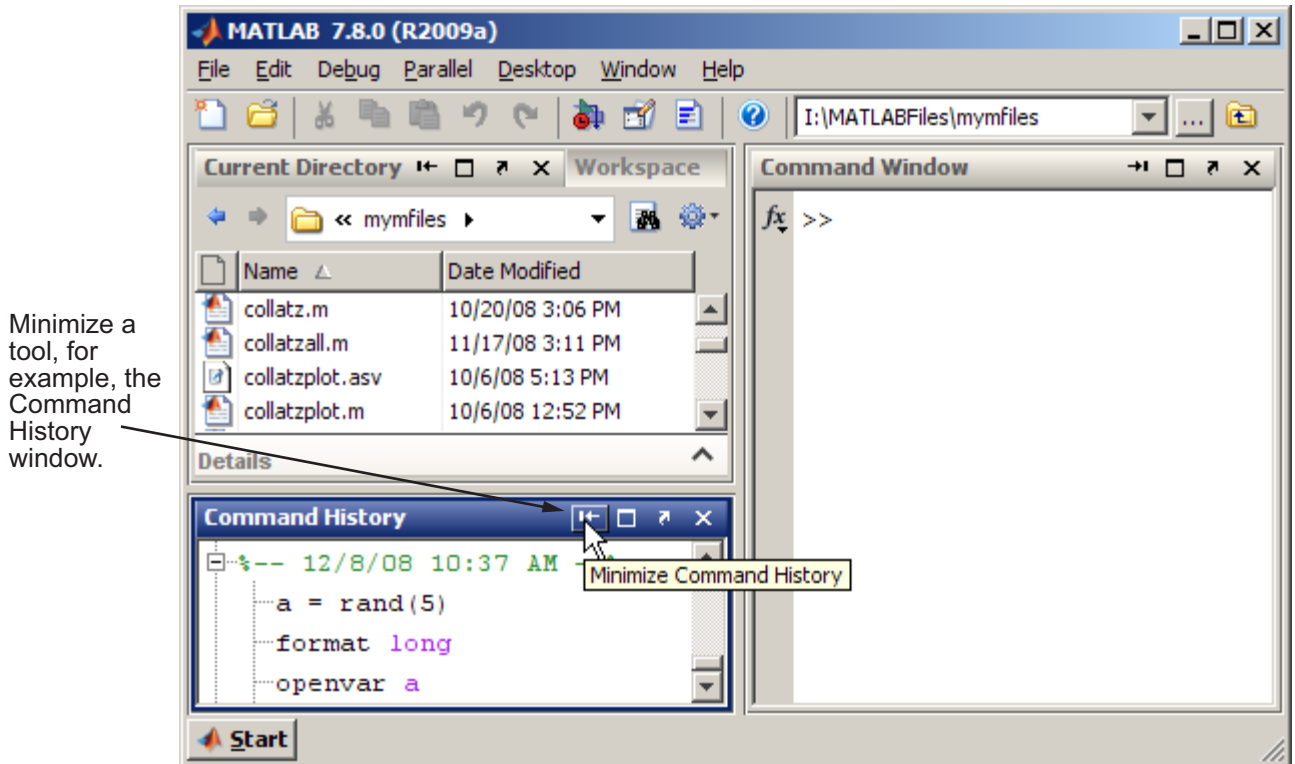
In this illustration, the Command History is minimized and appears as a button along the left edge.

---

6. UNIX is a registered trademark of The Open Group in the United States and other countries.

When minimized, a tool such as the Command History window in this example, is represented by a button on the desktop border.
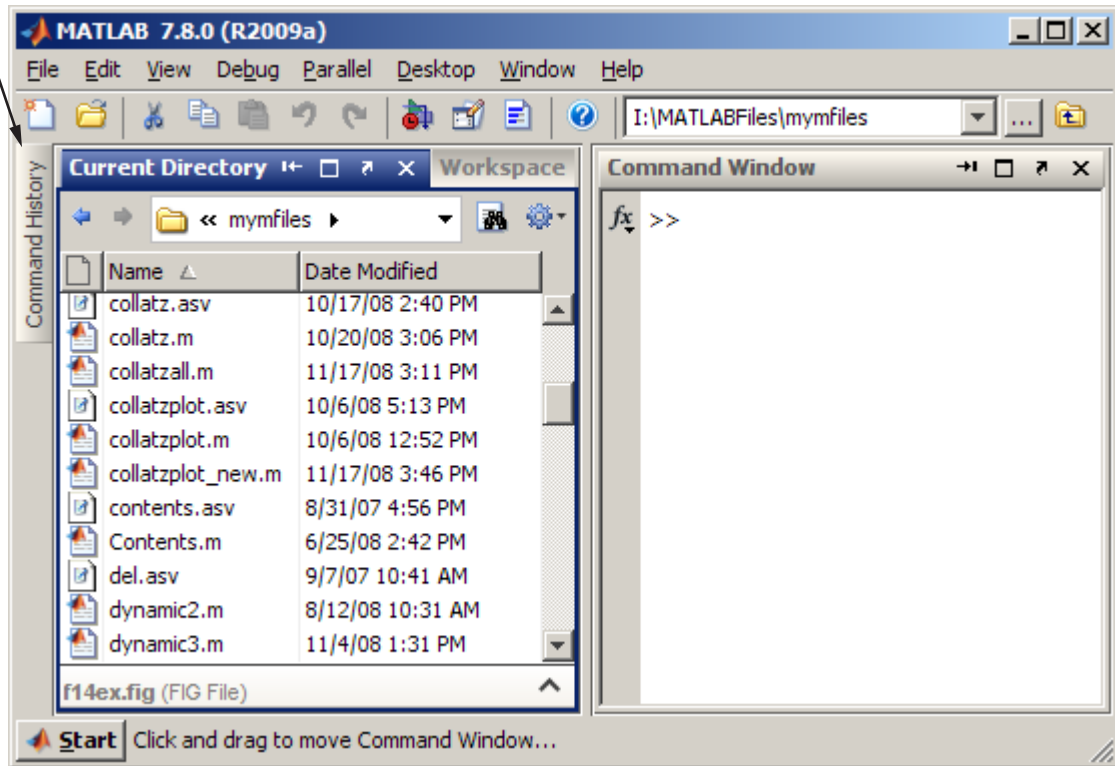
This illustration shows the minimized Command History temporarily open, as a result of clicking or hovering over the button.

Hover over or click the button for a minimized tool to temporarily view or use the tool.
The tool is temporarily displayed until you select another tool.
Then the tool becomes minimized again.
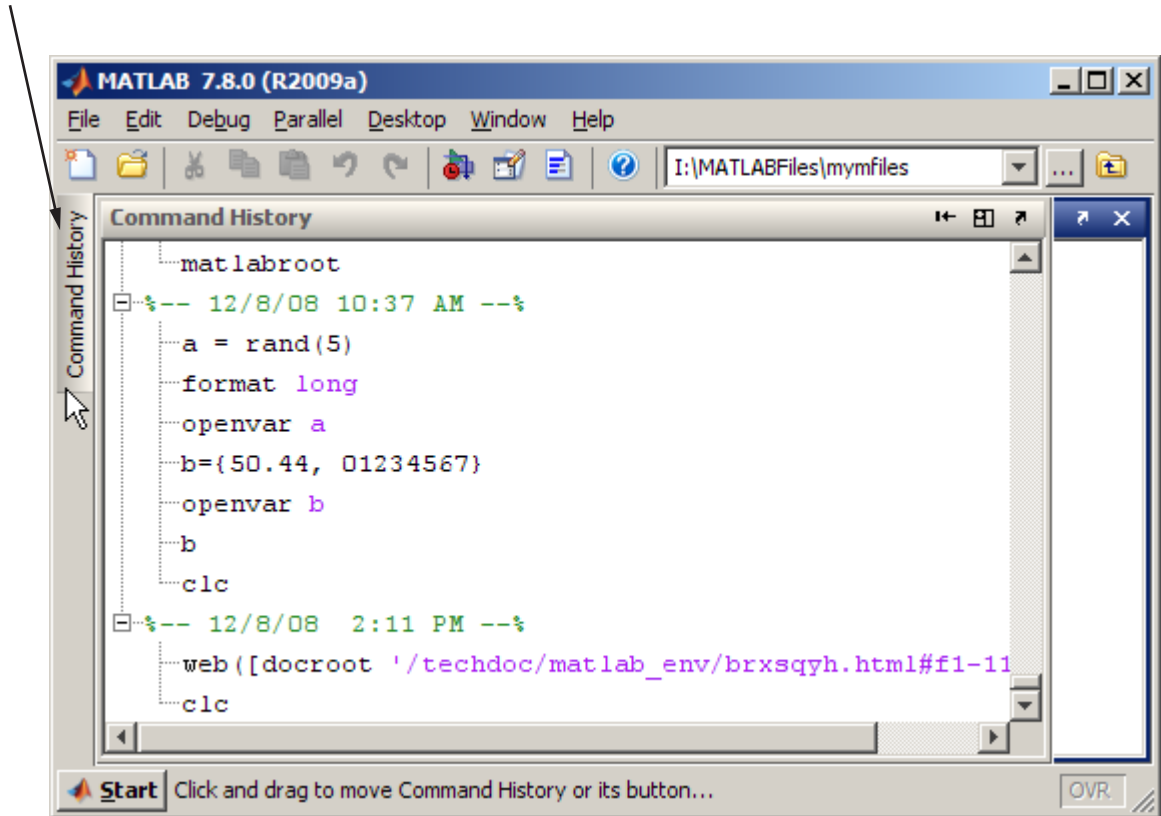
After using the Command History and clicking the button, or moving on to another tool, the Command History again becomes minimized as a button along the left edge.

This illustration shows a way to return the Command History to the position and size it occupied in the desktop before it was minimized. You can also achieve this effect by clicking the Restore button ⊞.

On the button for a minimized tool, right-click, and from the context menu, select **Restore**. The tool resumes the size and position it had in the desktop before it was minimized.
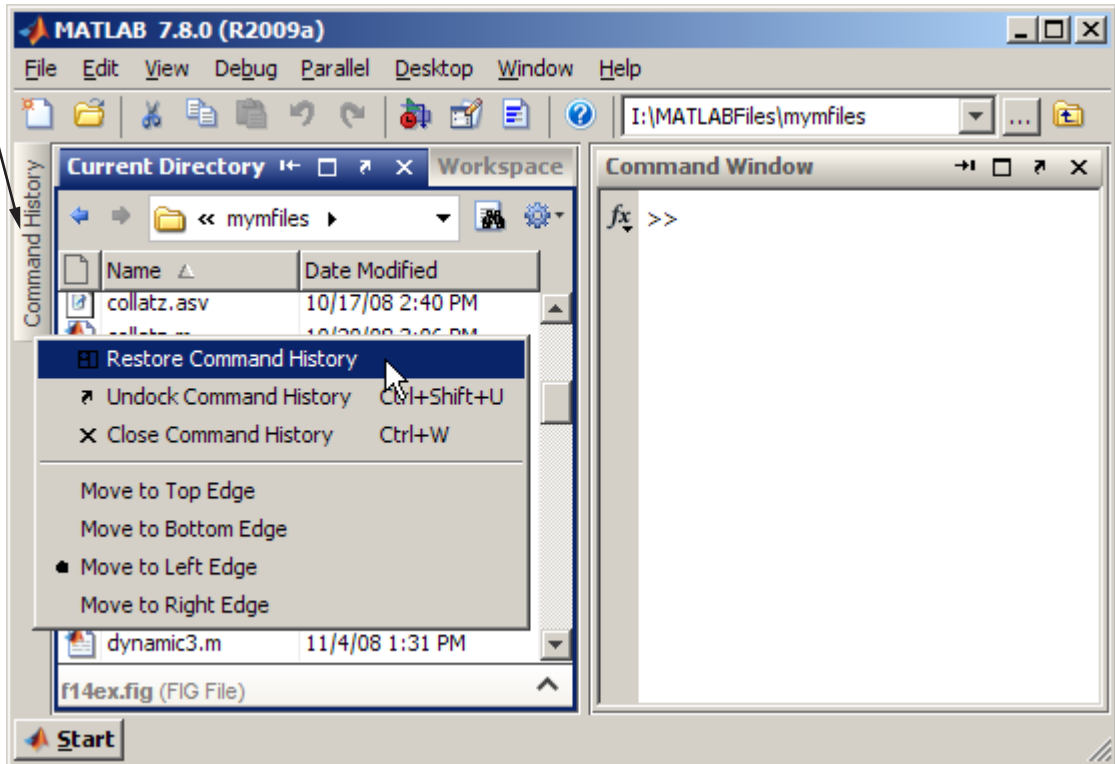
Tiled Documents in Desktop Example

When you open a document (for example, an M-file), it also opens the tool (for example, the Editor) if the tool is not already open. Subsequent documents of the same type open in the tool and you can then arrange the documents within the tool, as follows:

- Accept the default to have documents appear one on top of another, such that the one on top hides the one or ones beneath it.

  After changing the arrangement from the default, you can restore it by selecting **Window > Maximize** (or the □ toolbar button).

- To arrange documents so that two documents display simultaneously, side-by-side, select **Window > Left/Right Tile** (or the □ toolbar button).

When tools and documents are docked, you can save space by hiding toolbars and document bars:

- To hide (or show) a toolbar, select **Desktop > Toolbar name**.

- To see or move the document bar, select **Desktop > Document Bar > Bar Position**, and choose its location, for example, **Top**.

The following example shows two M-files, side-by-side, with the desktop shortcuts toolbar hidden.

The shortcuts toolbar is hidden. The document bar is at the top edge of the Editor.

Select a button from the list to arrange the documents, such as Left/Right Tile.
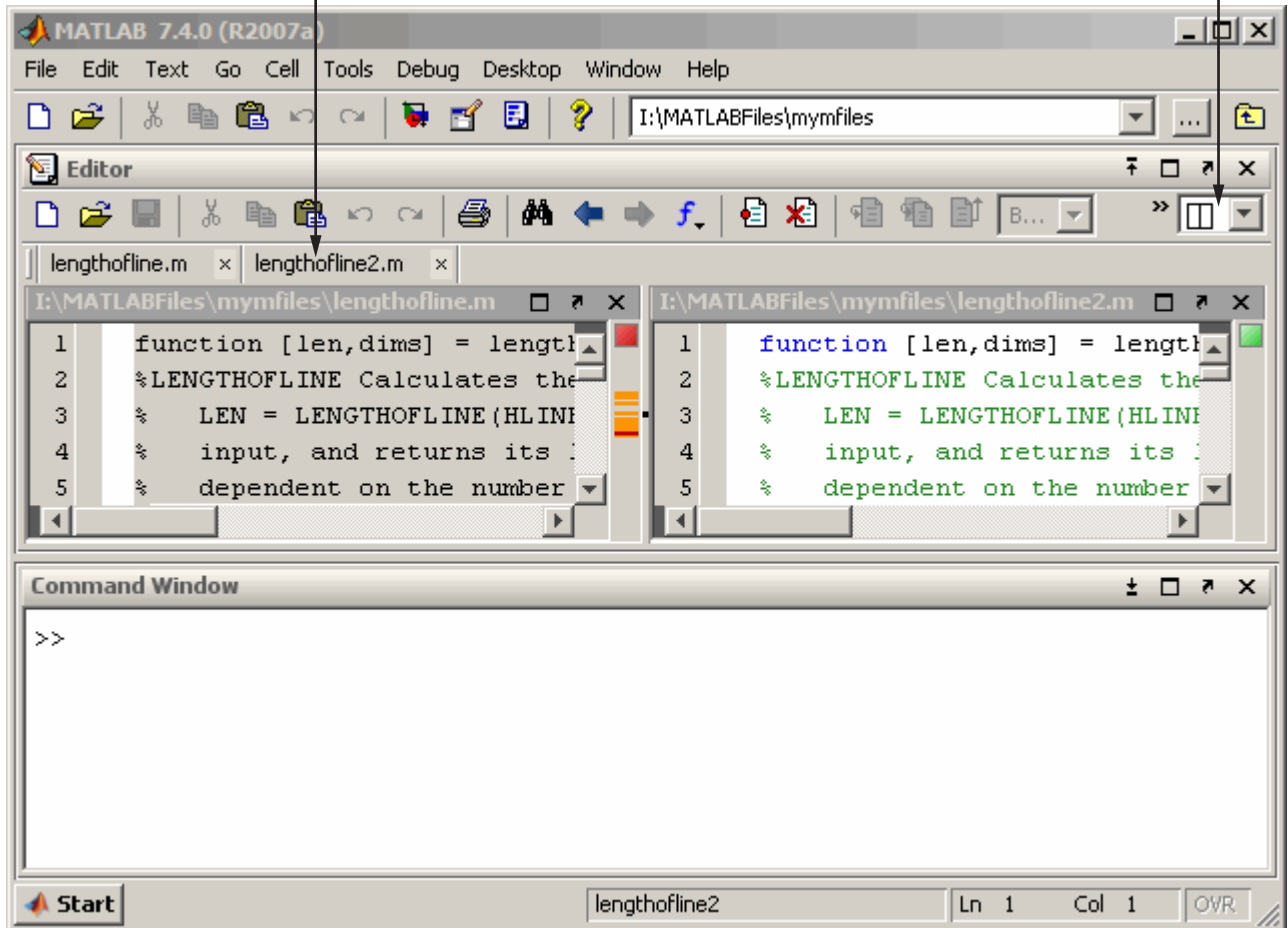
No Empty Document Tiles Example

The following example illustrates many of the options described in this list for creating and manipulating tiled documents.

- To see more than two documents at once, select the Tile button, and then move the pointer across the gird that appears to select the number of tiles you want.

  The following "Before" illustration has four tiles, but only three documents are open. (The empty tile is gray.)

- To move a document to any empty tile, drag its title bar to the new location.

- To close an empty tile:

  1 Position the pointer over the handle ▪ on the separator bar.

  The handle becomes a Close box, as shown in the example.

  2 Click the Close box.

  The empty tile closes and the neighboring document expands as shown in the following "After" illustration.

- To hide one document behind another, click the Close box between two tiles containing documents.

  One document becomes hidden.

Before

Close the empty tile
using the handle
on the separator bar.

Tile more than
two documents
with the Tile button.
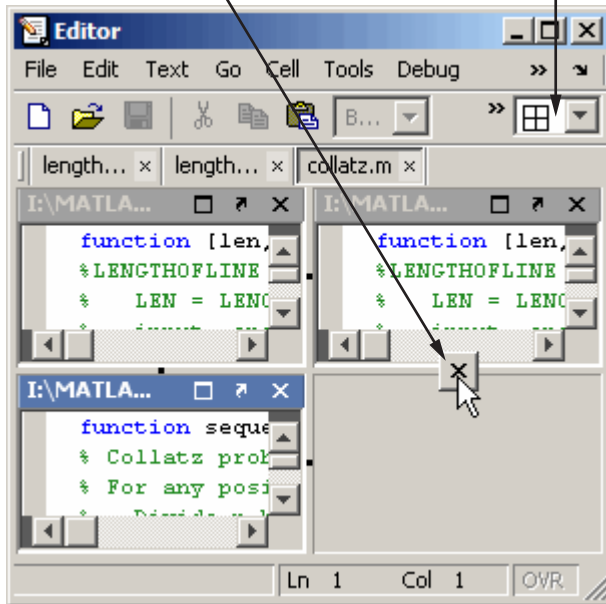
After

There are only
three tiles.
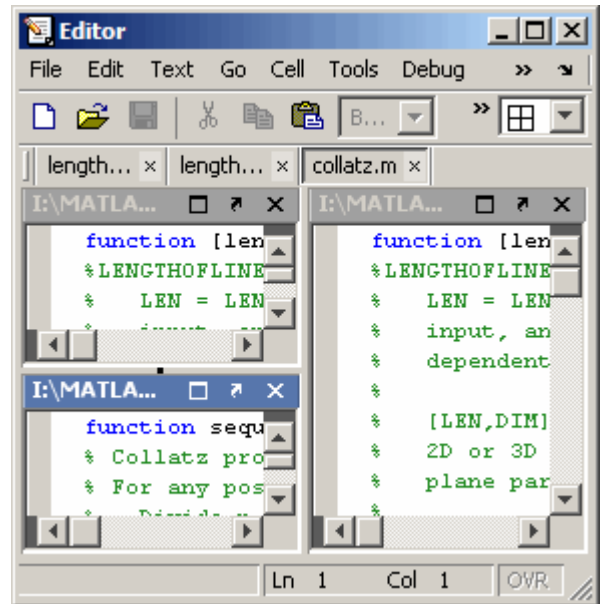
Maximized Documents Outside of the Desktop Example

This example illustrates a way to provide a large area for multiple documents, in this case, M-files maximized in the undocked Editor.
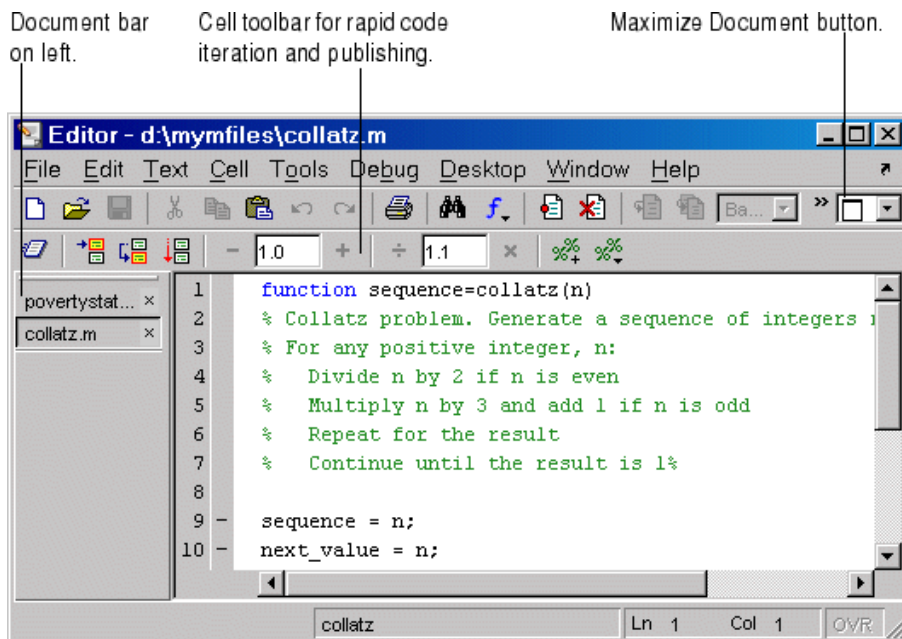
Some common actions for working with documents outside of the desktop are

- Group all Editor documents together — select **Desktop > Dock All in Editor** from any Editor document.

- Move all Editor documents outside of the desktop — select **Desktop > Undock Editor** when the Editor is the active window.

- Make a document occupy the full area in the Editor — click the Maximize button in the Editor toolbar, or select **Window > Maximize**.

- Display the cell toolbar — select **Desktop > Cell Toolbar**. This menu item is available only when the current document is an M-file.

- Access any document in the Editor using the document bar. To show the document bar on the left side of the Editor, select **Desktop > Bar Position > Document Bar > Left** from the Editor.

Floating (Cascaded) Figures in Desktop Example

This example illustrates multiple figures in the desktop. By default, figures open outside the desktop. You can arrange and adjust the figures, as follows:

- To move the figures into the desktop, click the Dock button in each figure's menu bar.

- To float (also called cascade) the figures, select **Window > Float**, or click the Float button ⊡.

- To get more screen area for the figures, hide the document bar as shown in this example — select **Desktop > Document Bar > Bar Position > Hide**.

Dock figures in the desktop and use the float option to arrange them within a Figures group. The document bar is hidden.
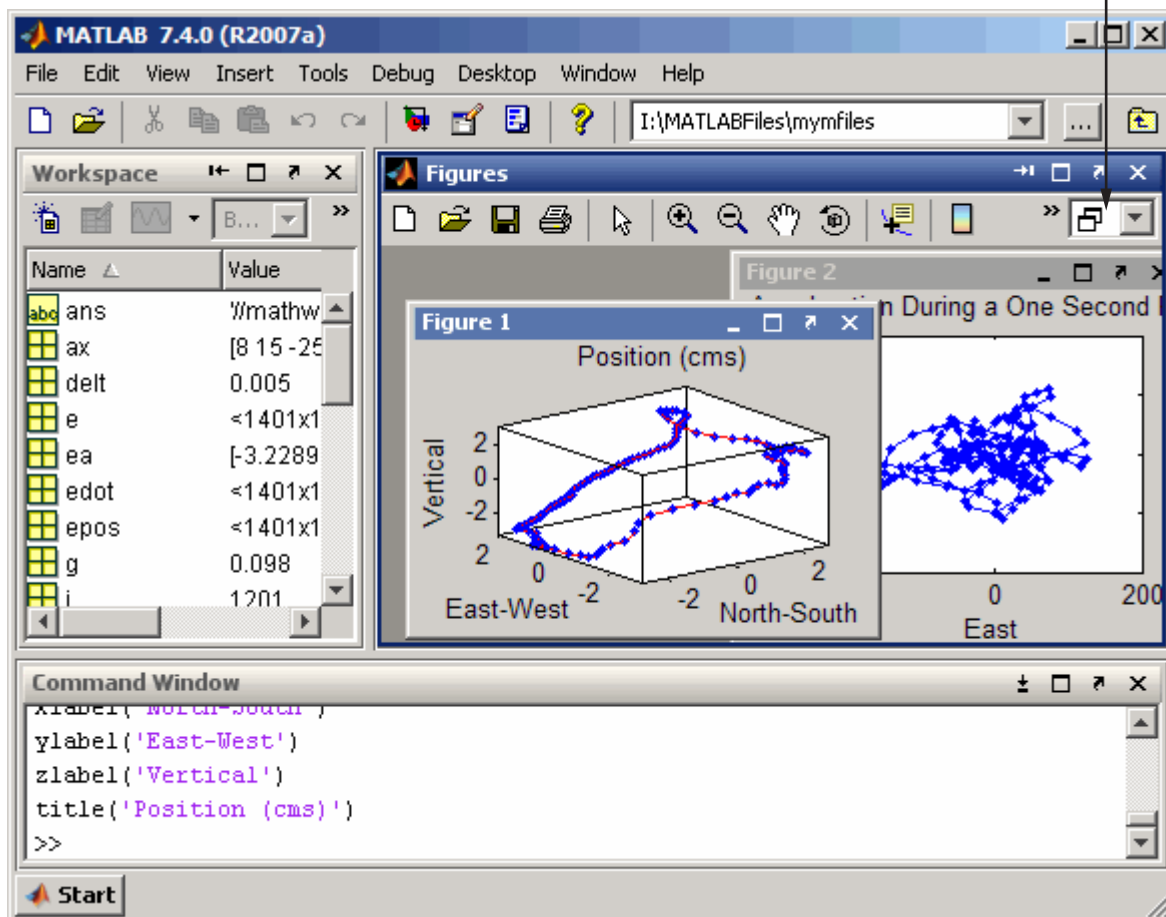
Undocked Tools and Documents Example

You can use tools and documents outside of the desktop as illustrated in the example that follows.

To undock a tool and it's documents:

1 Select **Desktop > Undock Toolname**.

2 Select **Desktop > Undock Documentname** from the tool.

If you undock all documents from a tool, an empty tool window remains.

To close all undocked documents and their tools at once, select
Window > Close All Documents from an undocked document window.

Notice the following in this example:

- One of the Editor documents, `collatz.m`, includes the name of the tool with it.

- The other Editor document, `lengthofline.m`, does not include the name of the tool with it.

    If you close the Editor, the `lengthofline.m` document remains open, but `collatz.m` closes.

- Neither of he Variable Editor documents includes the name of the tool.

    This is because the Variable Editor is undocked from the desktop, the variables are undocked from the Variable Editor, and the "empty" Variable Editor window is closed. The tool's undocked documents remain open.
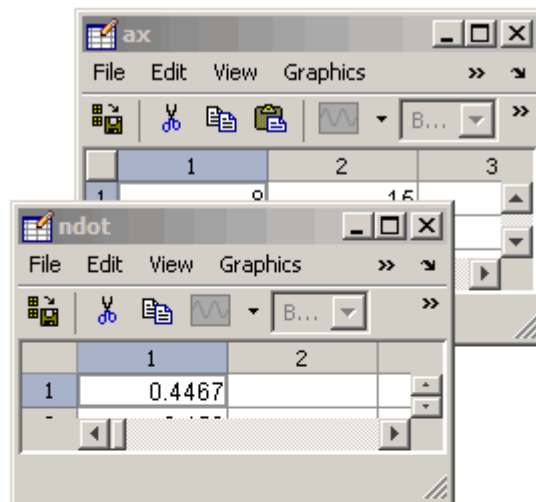
Using MATLAB Shortcuts to Easily Run a Group of Statements

In this section...
"What Is a Shortcut?" on page 2-59
"Examples of Useful Shortcuts" on page 2-59
"Creating Shortcuts" on page 2-60
"Running Shortcuts" on page 2-62
"Shortcuts Toolbar" on page 2-62
"Organizing and Editing Shortcuts" on page 2-65

What Is a Shortcut?

A MATLAB shortcut is an easy way to run a group of MATLAB language statements that you use regularly. First you create a shortcut that contains all the statements. Then you select and run the shortcut to execute all the statements it contains. Create, run, and organize shortcuts from the **Start > Shortcuts** menu or the desktop **Shortcuts** toolbar.

Differences Between Shortcuts and M-Files

A shortcut is like an M-file script. However, unlike an M-file, a shortcut does not have to be on the search path or in the current directory when you run it. In addition, you can run the shortcut by selecting it from the **Start** button or desktop **Shortcuts** toolbar, which are readily accessible.

Although shortcuts run MATLAB language statements, they are not M-files and MATLAB does not store them as M-files.

Examples of Useful Shortcuts

These are some examples of useful types of shortcuts:

- If you frequently run the same group of functions, consider creating a shortcut for them. For example, you can set up your environment when you start working if you do not use a startup file, or if there are statements

you do not want to include in the startup file. You can create a shortcut for even a single function that you use frequently, such as clc to clear the Command Window.

- Create a shortcut to set the same properties for figures you create, such as adding a legend and setting the background color.

- Create a shortcut for a long statement, such as changing the current directory (cd) when the pathnames are long.

- Create a shortcut for a statement you use, but do not easily remember.

Creating Shortcuts

This is an example of a shortcut you can create for a project called the Sea Temperature project. Suppose when you work on this project, you want to set up your environment in a certain way by running a series of MATLAB language statements. You create a shortcut called sea_temp_env, which contains the statements. Then when you work on the project, you run the shortcut to execute all the statements with a single click. The statements are

```
more on
format long e
cd d:/mymfiles/sea_temp_project
clear
workspace
filebrowser
clc
```

To create a shortcut, perform the following steps:

1 From the **Start** button, select **Shortcuts > New Shortcut**.

The Shortcut Editor dialog box appears.

2 Create the shortcut by completing the dialog box.

   a Provide a shortcut name in the **Label** field, for example, sea_temp_environment.

   b Put the statements in the **Callback** field as shown in the following illustration. Either type them in, or copy and paste or drag them from a desktop tool. Edit the statements as needed. The field uses the Editor

preferences for key bindings, colors, and fonts. Be aware that if you copy the statements from the Command Window, the prompt appears in the shortcut, but MATLAB removes the prompt when you save the shortcut.

c Assign a category, which is like a directory for organizing shortcuts. Specify sea_temp_project. To add the shortcut to the shortcuts toolbar, select the Toolbar Shortcuts category.

d Use the default shortcuts icon , or select your own.

e Click **Save**. MATLAB automatically removes any Command Window prompts (>>) in the **Callback** field upon saving the shortcuts.
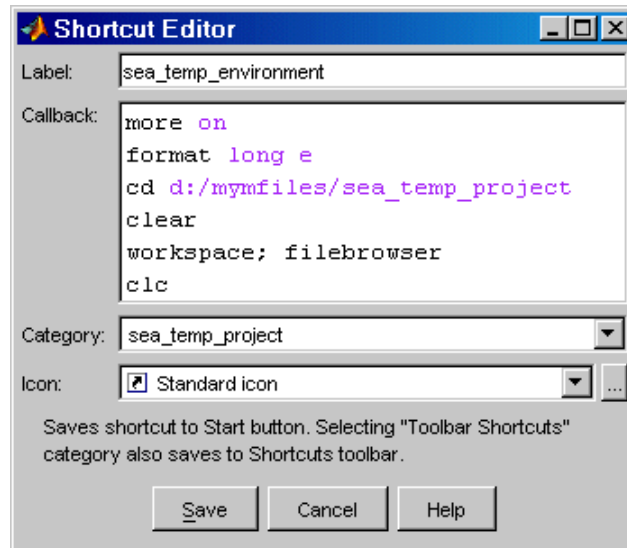
3 MATLAB adds the shortcut to the **Shortcuts** entry in the **Start** button, and to the **Shortcuts** toolbar, if you selected that **Category**.

After creating a shortcut, run it by selecting it from its category in the **Start** button. You also can run it from the **Shortcuts** toolbar if you selected the Toolbar Shortcuts category.

MATLAB maintains shortcut information in the file shortcuts.xml. Type prefdir, and MATLAB displays the location of the file. Most likely, you will not need to access this file, as MATLAB updates the file automatically.

For more information on the options in the Shortcut Editor dialog box, click the **Help** button.

Additional Ways to Create Shortcuts

You also can use these methods to create shortcuts:

- Add shortcuts to and run them from the desktop **Shortcuts** toolbar. See "Shortcuts Toolbar" on page 2-62.

- From the Command History window, create a shortcut by selecting MATLAB language statements, right-clicking, and selecting **Create Shortcut** from the context menu. By default, shortcuts created from the Command History window are assigned to the `Toolbar Shortcuts` category, meaning they will appear on the **Shortcuts** toolbar.

- From the Help browser, select **Favorites > Add to Favorites**, complete the Favorites Editor dialog box, and the shortcut appears in the shortcuts `Help Browser Favorites` category. You also can access `Help Browser Favorites` shortcuts from the Help browser **Favorites** menu.

- Drag statements from a desktop tool, such as the Command History, onto the **Start** button.

Running Shortcuts

To run a shortcut, select the shortcut name, for example, **sea_temp_environment**, from the **Start > Shortcuts** menu or from one of its category submenus. All the statements in the shortcut **Callback** field execute. It is as if you ran those statements from the Command Window, although they do not appear in the Command History window.

If you added a shortcut to the **Shortcuts** toolbar, you can run it by clicking its icon on the shortcuts toolbar.

Shortcuts Toolbar

The **Shortcuts** toolbar is an alternative to creating and running shortcuts via the **Start** button. To show or hide the shortcuts toolbar, use **Desktop > Shortcuts Toolbar**. To create and run shortcuts via the desktop **Shortcuts** toolbar, perform these steps:

1 Select statements from the Command History window, the Command Window, or an M-file.

2 Drag the selection to the desktop **Shortcuts** toolbar. The following illustration shows two statements being dragged from the Command Window.
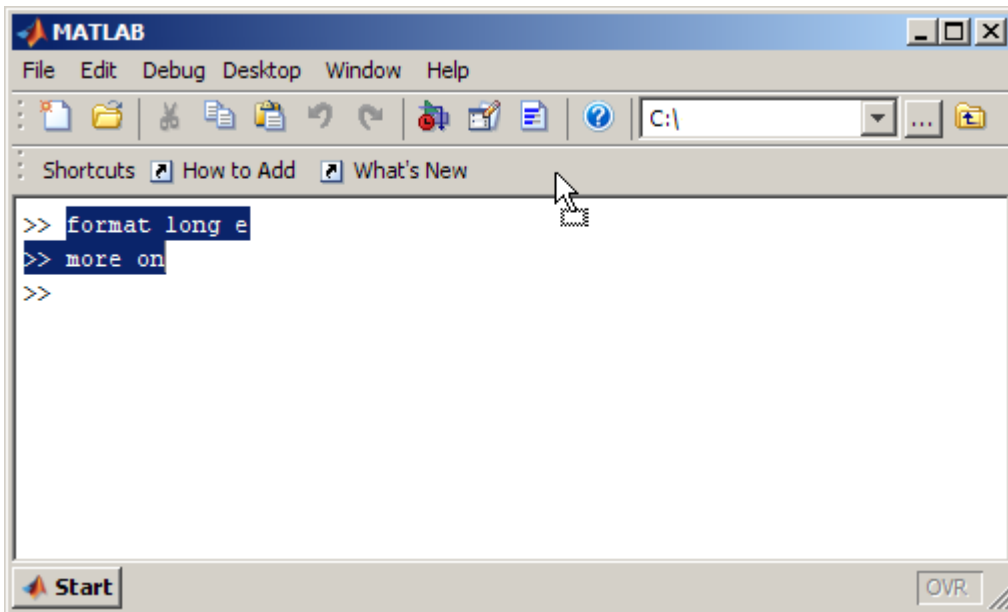
3 The Shortcut Editor dialog box appears. The **Callback** field contains the selected statements, which you can edit as needed. If prompts (>>) from the Command Window appear, note that MATLAB automatically removes them when you save the shortcut. The **Category** field is Toolbar Shortcuts, which you must retain for the shortcut to appear on the toolbar.

Provide the **Label**, select an **Icon**, and click **Save**.

The shortcut icon and label appear on the toolbar. If you have more shortcuts on the toolbar than the desktop can display concurrently, use the drop-down list to access them all. For more information, click the **Help** button in the Shortcut Editor dialog box.

4 Click the icon on the **Shortcuts** toolbar to run the shortcut. You also can run the shortcut from the **Start** button by selecting it in the `Toolbar Shortcuts` category.

Click a shortcut to run it.
Right-click a shortcut, then delete or edit it.

You also can add a shortcut to the desktop **Shortcuts** toolbar by right-clicking the toolbar and selecting **New Shortcut**. Complete the resulting Shortcut Editor dialog box. Assuming that you maintain the `Toolbar Shortcuts` category, the shortcut appears on the toolbar. To change the order of the shortcuts on the toolbar, select **Start > Shortcuts > Organize Shortcuts** and move the shortcuts within the `Toolbar Shortcuts` category.

How to Add and What's New Shortcuts

The **Shortcuts** toolbar includes two shortcuts provided with MATLAB. The **How to Add** shortcut provides help about shortcuts and adding them to the **Shortcuts** toolbar. **What's New** displays the Release Notes documentation.

To remove the **How to Add** or **What's New** shortcut from the **Shortcuts** toolbar, choose a different category. For instructions, see "Organizing and Editing Shortcuts" on page 2-65.

If you do not want to keep these shortcuts, remove each one by right-clicking its toolbar shortcut button and selecting **Delete** from the context menu. Click **OK** in the confirmation dialog box to remove the shortcut.

Shortcut Labels on Toolbar

You can hide the shortcut labels on the toolbar. Right-click in the **Shortcuts** toolbar. From the context menu, select **Show Labels**, which clears the check mark next to the item. The shortcut icons appear on the toolbar without labels. When you move the mouse over a shortcut icon, its label appears as a

ToolTip. To make labels display in the toolbar, right-click the toolbar, and then select **Show Labels**. This adds a check mark next to the item and displays the labels.

Organizing and Editing Shortcuts

To create categories for shortcuts, and to move, edit, and delete shortcuts, perform these steps:

1 From the **Start** button, select **Shortcuts > Organize Shortcuts**. Alternatively, access it via the shortcuts toolbar context menu.

The Shortcuts Organizer dialog box appears. When you select a shortcut category in the dialog box, the **Edit Shortcut** button replaces the **Rename Category** button.
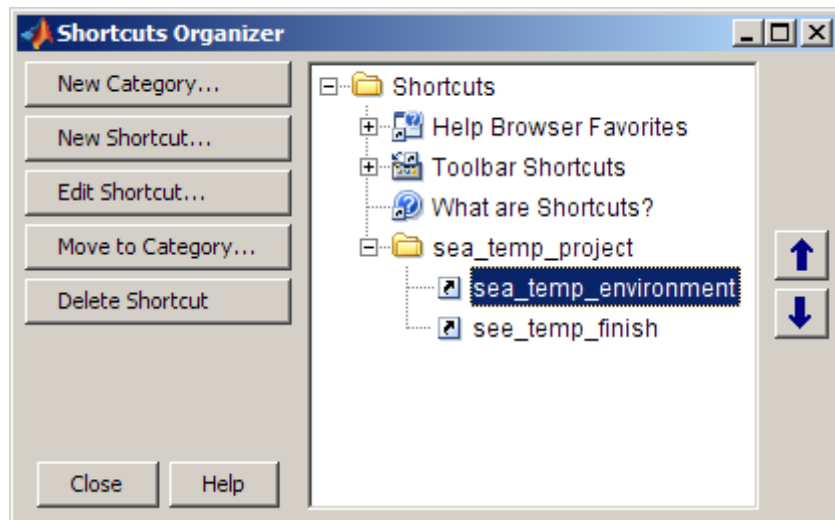
2 Use the buttons in the dialog box to edit and organize shortcuts and categories. You also can right-click an item and select an action from the context menu.

Changes take effect immediately.

3 Click **Close**.

For more information about using the Shortcuts Organizer dialog box, click the **Help** button.

Performing Desktop Tasks Using Keyboard Shortcuts

In this section...
"Keyboard Shortcuts (Accelerators or Hot Keys) and Mnemonics" on page 2-67
"Default Button and Active Button (Button with Focus)" on page 2-69

Keyboard Shortcuts (Accelerators or Hot Keys) and Mnemonics

You can access many of the desktop menu items using shortcut keys (sometimes called accelerators or hot keys) for your platform. For example, use the **Ctrl+X** shortcut to perform a cut on Microsoft Windows platforms. Many of the menu items show the shortcuts. Additional standard shortcuts for your platform usually work but only one is listed with each menu item.

Instructions in the documentation specify shortcuts using the key convention for Windows platforms, **Ctrl+**. With key bindings for Apple Macintosh platforms selected, you use the Command key instead of the **Ctrl** key. On the Macintosh platform, to make full use of all keyboard shortcuts, you might need to enable full keyboard access. To access this option, choose **Apple menu > System Preferences**, and click **Keyboard & Mouse**. Click **Keyboard Shortcuts** and select **Turn full keyboard access on or off**.

You also can use mnemonics to access menu items and buttons, such as **Alt+F** to open the **File** menu. This is not supported on the Macintosh platform. Mnemonics are listed with the menu item or button. For example, on the **File** menu, the **F** in **File** is underlined, which indicates that **Alt+F** opens the menu. In the Profiler, the **R** in the **Run this code** toolbar field is underlined, indicating that **Alt+R** moves the cursor to this field.

Note that some versions of the Windows operating system do not automatically show the mnemonics on the menu. For example, you might need to hold down the **Alt** key while the tool is selected to see the mnemonics on the menus and buttons. Use the Windows Control Panel to set preferences for underlining keyboard shortcuts. See the Windows documentation for details.

Following are some general shortcuts that are not listed on menu items.

Key	Result
Enter	The equivalent of double-clicking, **Enter** performs the default action for a selection. For example, press **Enter** while a statement in the Command History window is selected to run that statement in the Command Window.
	For buttons in tools and dialog boxes, **Enter** executes the default button (the button with a border around it). If there is no default button, press the **space bar** to execute the active button (the button with a dotted outline inside it). See "Default Button and Active Button (Button with Focus)" on page 2-69 for an illustration.
Esc (escape)	Cancels the current action. For example, if you select the **Edit** menu, the menu items display. Pressing **Esc** retracts the menu items. Pressing **Esc** in a dialog box is the same as selecting the **Cancel** button.
Tab	Advances to the next button or field in a tool or dialog box.
	In the Command Window, completes a statement if the tab completion preference is selected.
Space bar	For buttons in tools and dialog boxes, activates the active button. See "Default Button and Active Button (Button with Focus)" on page 2-69 for an illustration of selecting default and active buttons using keys.
+ or **-** or ***** on numeric keypad	Use these keys on the numeric keypad to expand and collapse items in tree views. The Help browser **Help Navigator** pane and the Command History window use tree views. Use **+** to expand the selected item, use **-** to collapse the selected item, and use ***** to recursively expand it, meaning open all items contained in the selected item.
Alt+S	Displays the **Start** button menu (except on Macintosh platforms).
Alt+Y	Provides access to the current directory field in the toolbar (except on Macintosh platforms).
Ctrl+Tab	Moves to the next open tool in the desktop, or to the next open group of tools tabbed together.
Ctrl+Shift+Tab	Moves to the previous open tool or group of tabbed tools in the desktop.
Ctrl+Page Down	Moves to the next tool within a group of tools tabbed together. In a group of documents, moves to next document.

Key	Result
Ctrl+Page Up	Moves to the previous tool within a group of tools tabbed together. In a group of documents, moves to previous document.
Ctrl+F6	Moves to the next tool or document (only for Windows and Sun Microsystems Solaris™ platforms).
Ctrl+Shift+F6	Moves to the previous tool or document (only for Windows and Solaris platforms).
Ctrl+Shift+G	On Macintosh platforms, in a file browser GUI, opens the Go To Folder dialog box. This is required to access the contents of *matlabroot* from a file browser GUI. For more information, see "Using File Browser GUIs on Macintosh Platforms to Navigate Within the MATLAB Root Directory" on page 2-88.
Alt+F4	Closes the desktop and consequently, shuts down the MATLAB program. Or outside the desktop, closes the active window (except on Macintosh platforms).

For additional shortcuts available in the various desktop tools, see the documentation for each tool. For example, see "Keyboard Shortcuts in the Command Window" on page 3-19 and "Keyboard Shortcuts in the Editor" on page 6-93.

Default Button and Active Button (Button with Focus)

These illustrations demonstrate the default versus active button in a dialog box.

The default button has a border around it. Here, **Save** is the default button. Press the **Enter** key to execute the default button.

The active button (the button with focus) has a dotted outline inside it. Here, **Cancel** is the active button. Press the space bar to execute the active button.
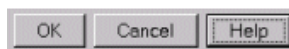
Here, the **Help** button is both the default button and the active button. In some cases, the default always changes to match the active button. You can press either **Enter** or the space bar to execute the **Help** button

Accessing Tools with the Start Button

The MATLAB **Start** button provides easy access to tools, demos, and documentation for all your MathWorks products. From it, you also can create and run MATLAB shortcuts, which are groups of MATLAB language statements.

Viewing Products and Tools with the Start Button

1 Click the **Start** button to view a menu of product categories and desktop tools installed on your system. As an alternative, press **Alt+S** (except on Apple Macintosh platforms). In the following illustration, the **Start** button shows MATLAB selected.
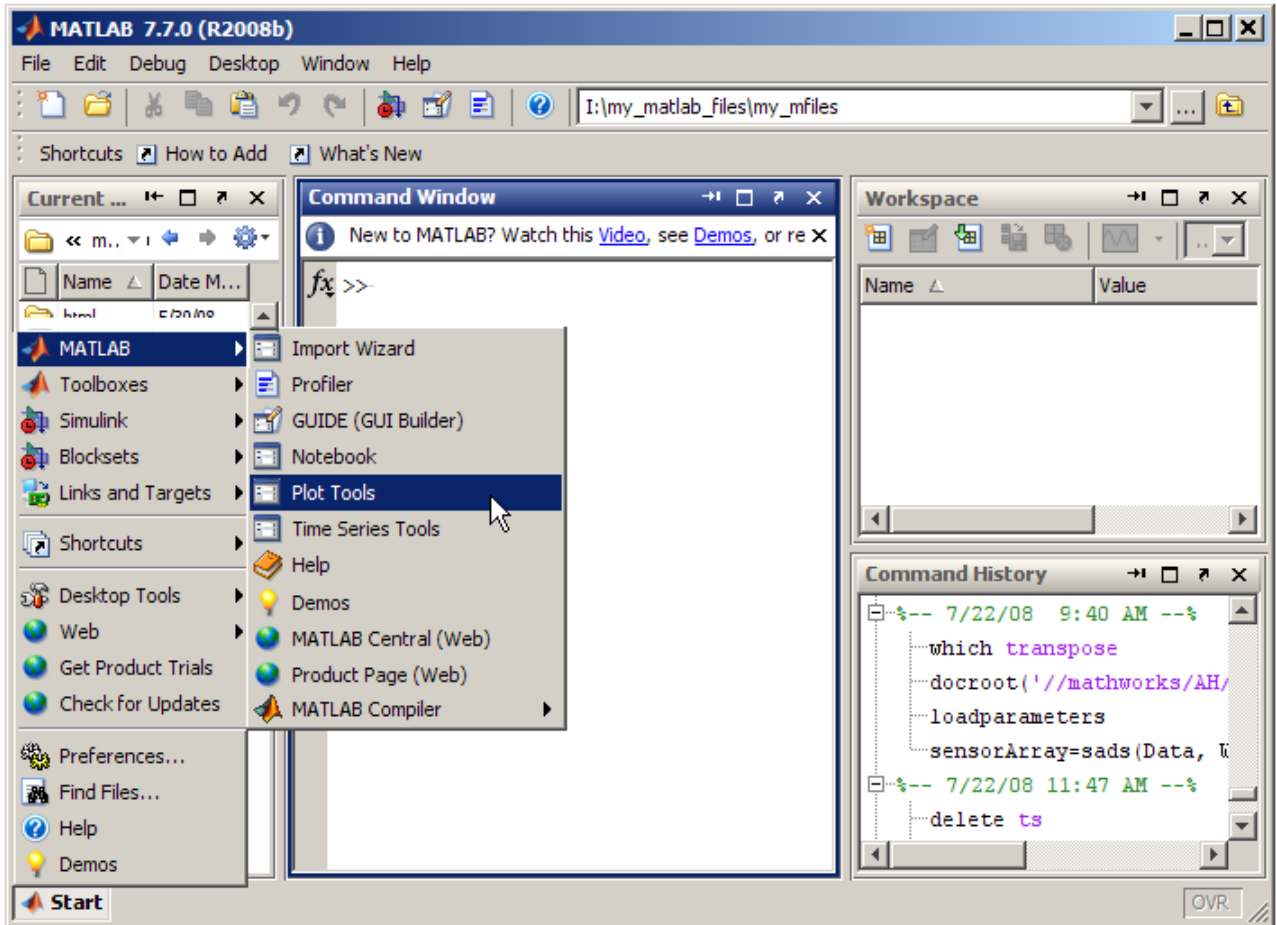
2 From the menu and submenu items, select an item to open it. The icons help you quickly locate a type of product or tool—see the icon descriptions in the following table.

For example, to open plot tools, select **Start > MATLAB > Plot Tools**.

Identifying Icons in the Start Button

Icons in the Start button menus help you quickly locate a particular type of product or tool. This table describes the action performed when you select an entry with one of these icons in the **Start** button.

Icon	Description of Action When Opened
	Documentation for that product opens in the Help browser.
	Demos for the product are listed in the Help browser **Demos** pane.
	Selected tool opens.
	Block library opens.
	Document opens in your system Web browser.

Adding Your Own Toolboxes to the Start Button

- "About Adding Your Own Toolbox to the Start Button" on page 2-72

- "Procedure for Adding Your Own Toolbox to the Start Button" on page 2-73

- "Description of info.xml File for Adding Your Own Toolbox to the Start Button" on page 2-75

- "See Also" on page 2-80

About Adding Your Own Toolbox to the Start Button

If you offer a collection of files for MATLAB and related products (referred to here as a toolbox) to other users, you can provide access to it from the **Start** button. (For background information about the **Start** button, see "Accessing Tools with the Start Button" on page 2-70.)

MATLAB determines what information to display on the **Start** button using info.xml files; the info.xml files must be in directories on the search path or in the current directory.

You create an info.xml file for your toolbox, which contains tags for all of the subentries you want on the **Start** button. Provide the info.xml to

users, along with your toolbox files. With the `info.xml` in a directory on their search path or in their current directory, users can access your toolbox from the **Start** button.

Procedure for Adding Your Own Toolbox to the Start Button

The example file, `startinfo.xml`, accompanies the following instructions. It adds the SpecSim (Example) Toolbox to the **Start** button. Use the example as a basis for creating an `info.xml` file for your own toolbox.

1 Copy the example file *matlabroot*/help/techdoc/matlab_env/examples/startinfo.xml to a directory for which you have write access.

2 In that directory, rename `startinfo.xml` to `info.xml`.

3 Add the directory containing the `info.xml` file to the search path for MATLAB.

4 To see what the example `info.xml` file does:

a Select **Start > Desktop Tools > View Start Button Configuration Files**.

The Start Button Configuration Files dialog box opens. Click **Refresh Start Button**. Perform this step whenever you want to see changes to `info.xml` files reflected in the **Start** button.
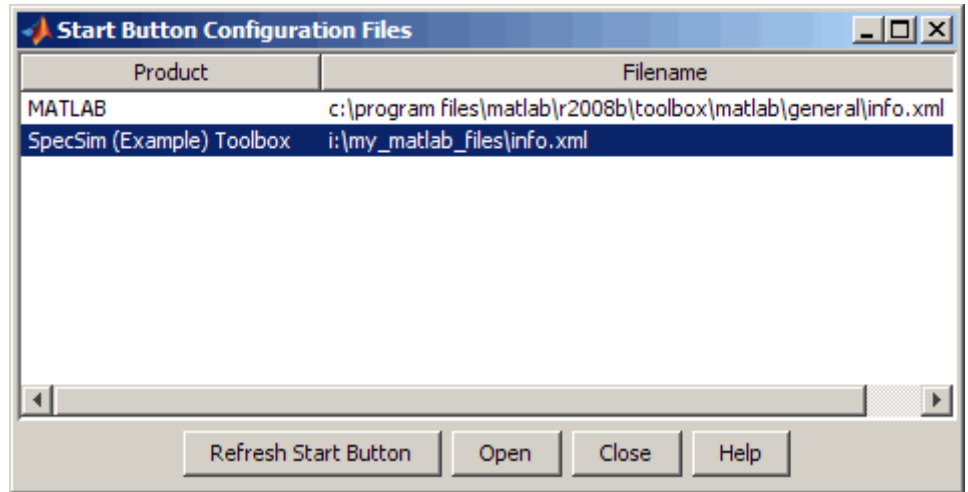
b In the desktop, view the item in the **Start** button. For the example, select **Start > Toolboxes > SpecSim (Example)**.
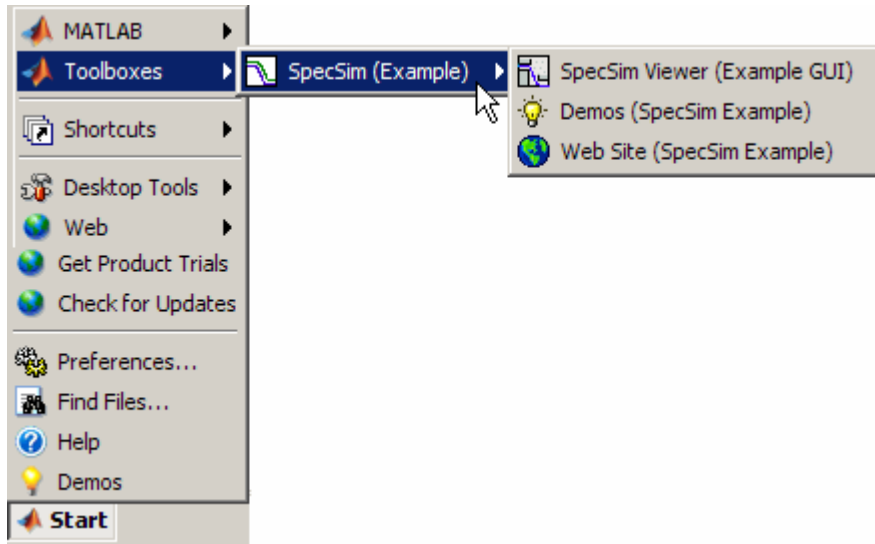
5 In the Start Button Configuration Files dialog box, select the `info.xml` file that you want to edit. In this example, select `SpecSim (Example) Toolbox`. Click **Open**.

The `info.xml` file for the selected product opens in the Editor.

6 In the Editor, change the entries in the `info.xml` file so that they pertain to your toolbox. For details, see the sample code and table shown after the last step.

For more information about the `info.xml` file structure, consult its schema, which is at *matlabroot*`/sys/namespace/info/v1/info.xsd`.
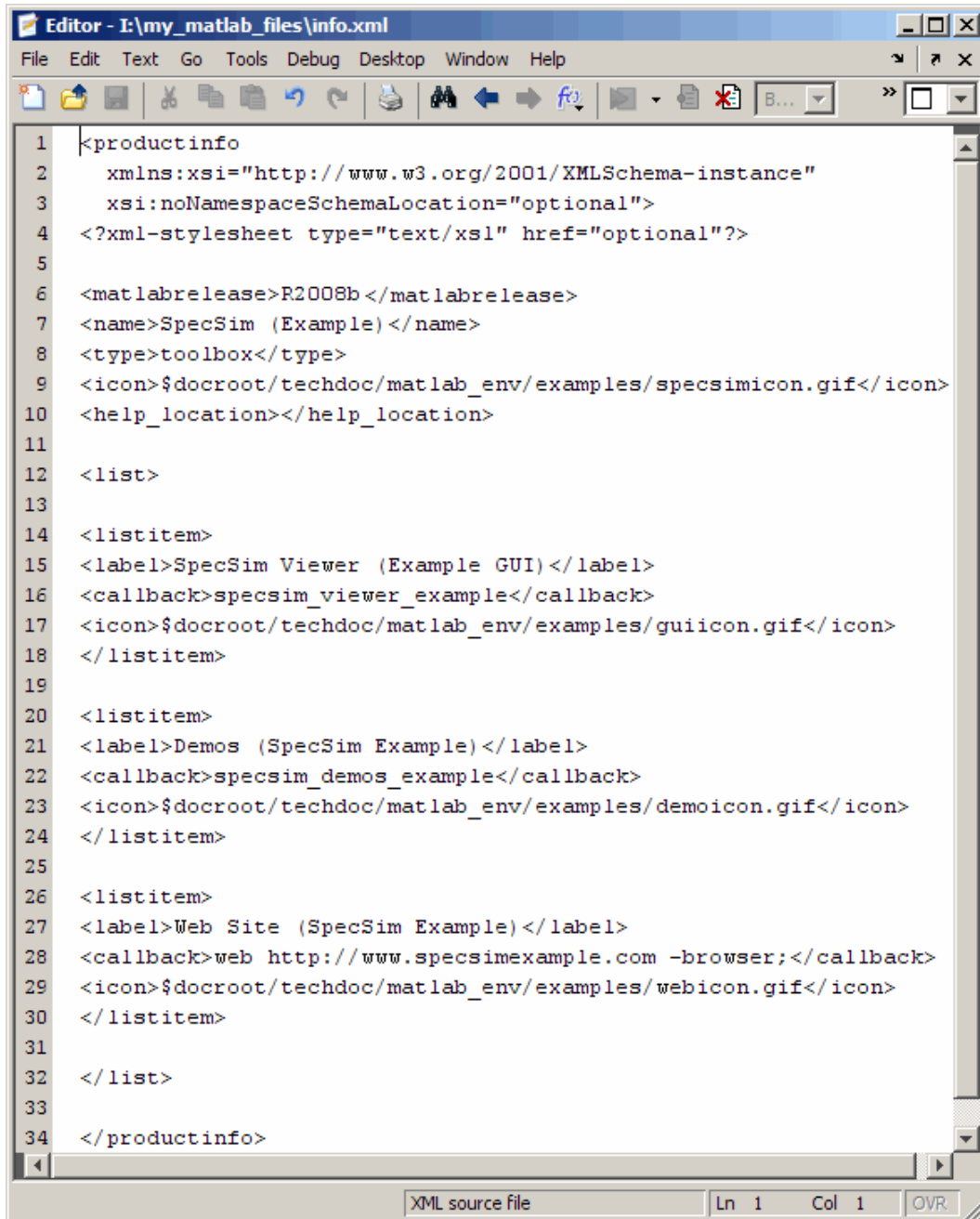
7 In the Editor, select **File > Save** to save the changes.

8 To see the results of the changes to the `info.xml` file, in the Start Button Configuration Files dialog box, click **Refresh Start Button**, and then view the **Start** button.

MATLAB automatically validates your `info.xml` file against the schema. If there is an invalid construct, MATLAB displays an error in the Command Window.

Use the message in the error to isolate the problem or use any validator. One validator you can use is from the W3C® at `http://www.w3.org/2001/03/webdata/xsv`.

Description of info.xml File for Adding Your Own Toolbox to the Start Button

This is the example file provided with MATLAB.

```
Editor - I:\my_matlab_files\info.xml                              _ □ X
File  Edit  Text  Go  Tools  Debug  Desktop  Window  Help

 1    <productinfo
 2      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 3      xsi:noNamespaceSchemaLocation="optional">
 4    <?xml-stylesheet type="text/xsl" href="optional"?>
 5
 6    <matlabrelease>R2008b</matlabrelease>
 7    <name>SpecSim (Example)</name>
 8    <type>toolbox</type>
 9    <icon>$docroot/techdoc/matlab_env/examples/specsimicon.gif</icon>
10    <help_location></help_location>
11
12    <list>
13
14    <listitem>
15    <label>SpecSim Viewer (Example GUI)</label>
16    <callback>specsim_viewer_example</callback>
17    <icon>$docroot/techdoc/matlab_env/examples/guiicon.gif</icon>
18    </listitem>
19
20    <listitem>
21    <label>Demos (SpecSim Example)</label>
22    <callback>specsim_demos_example</callback>
23    <icon>$docroot/techdoc/matlab_env/examples/demoicon.gif</icon>
24    </listitem>
25
26    <listitem>
27    <label>Web Site (SpecSim Example)</label>
28    <callback>web http://www.specsimexample.com -browser;</callback>
29    <icon>$docroot/techdoc/matlab_env/examples/webicon.gif</icon>
30    </listitem>
31
32    </list>
33
34    </productinfo>

XML source file                        Ln  1        Col  1      OVR
```

This table provides details about the example `info.xml` file.

Line	XML Tag	Value for Example	Notes
6	`<matlabrelease>`	R2008b	Release of MATLAB. Not currently used.
7	`<name>`	SpecSim (Example)	Name of the toolbox.
8	`<type>`	toolbox	The product type. Allowable values are `matlab`, `simulink`, `toolbox`, `blockset`, `links_targets`, `other`. This determines where the entry appears in the **Start** button. **SpecSim (Example)** appears under **Start > Toolboxes**. Within a type, entries are shown in alphabetical order.
9	`<icon>`	`$docroot/techdoc/matlab_env/ examples/specsimicon.gif`	Shows an icon for your toolbox. You can specify a relative path, that is, relative to the location of the `info.xml` file. In this example, `specsimicon.gif` is located in the `examples` directory provided with the documentation.

Line	XML Tag	Value for Example	Notes
10	`<help_location>`	None	Location of HTML help files you provide for the toolbox. For details, see "Adding HTML Help Files for Your Own Toolbox" on page 4-72. Specify a relative path, that is, relative to the location of the info.xml file. In this example, there are no help files, so the entry has no value.
12 to 15	`<list>` `<listitem>` `<label>`	SpecSim Viewer (Example GUI)	Name of a tool in your toolbox. You must have at least one list-listitem-label set of tags for the toolbox to appear in the **Start** button.
16	`<callback>`	specsim_viewer_example	The function that runs when you select the item from the **Start** button. You must have at least one callback for the toolbox to appear in the **Start** button. In this example, specsim_viewer_example is a function that starts the SpecSim Viewer GUI when you select **Start > Toolboxes > SpecSim (Example) > SpecSim Viewer (Example GUI)**. (This example file is not provided.)
17	`<icon>`	$docroot/techdoc/matlab_env/ examples/guiicon.gif	Shows an icon for your callback. You can specify a relative path, that is, relative to the location of the info.xml file. In this example, guiicon.gif is located in the examples directory provided with the documentation.

Line	XML Tag	Value for Example	Notes
21	`<label>`	`Demos (SpecSim Example)`	For examples you provide with the toolbox.
22	`<callback>`	`specsim_demos_example`	The function that runs when you select the item from the **Start** button. In this example, `specsim_demos_example` is an example that runs when you select **Start > Toolboxes > SpecSim (Example) > Demos (SpecSim Example)**. (This example file is not provided.)
23	`<icon>`	`$docroot/techdoc/matlab_env/ examples/demoicon.gif`	Shows an icon for your demos. You can specify a relative path, that is, relative to the location of the `info.xml` file. In this example, `demoimicon.gif` is located in the `examples` directory provided with the documentation.
27	`<label>`	`Web Site (SpecSim Example)`	For a link to your toolbox Web site.
28	`<callback>`	`web http://www.specsimexample.com -browser;`	The statement that runs when you select the item from the **Start** button. In this example, it displays the Web site for the SpecSim (Example) Toolbox in the system browser when you select **Start > Toolboxes > SpecSim (Example) > Web Site**

Line	XML Tag	Value for Example	Notes
			(SpecSim Example). (This URL is intentionally invalid.)

30 to 34	`</listitem>`
	 `</list>`
	 `</productinfo>`
		None	Designates end of `list` and `productinfo` entries in the **Start** button for the toolbox.

See Also

- "Validation Errors for info.xml Files" on page 4-88

- "Adding Your Own Help Files and Demos in the Help Browser" on page 4-72

Other Desktop Features

In this section...
"Using Menus and Context Menus" on page 2-81
"Using Toolbar Features" on page 2-83
"Viewing Status in the Status Bar" on page 2-84
"Sizing, Arranging, and Sorting Columns in Desktop Tools" on page 2-85
"Selecting Multiple Items" on page 2-86
"Cut, Copy, Paste, and Move" on page 2-87
"Macintosh Platform — Differences" on page 2-88
"Printing and Page Setup Options for Desktop Tools" on page 2-89
"Displaying Documents and Web Sites in Web Browsers" on page 2-93
"Accessing The MathWorks on the Web" on page 2-100
"Managing Your Licenses" on page 2-101
"Check for Updates" on page 2-103

Using Menus and Context Menus

Understanding Merged Menus

When you use a tool in the desktop, its menu appears at the top of the desktop. When you work in a different tool in the desktop, you still use the menu at the top of the desktop, but the menu content changes to support that tool. When you undock a tool from the desktop, access its menu at the top of the undocked tool.

Context Menus

Many of the features in MATLAB desktop tools are available from context menus, also known as pop-up or right-click menus. To access a context menu, right-click a selection or an area, or press **Ctrl+Shift+F10**. The context menu for the selection or tool appears, presenting the available actions.

2-81

For example, following is the context menu for a selection in the Command History window.

Access context (pop-up) menus by right-clicking a selection or any area in a tool.
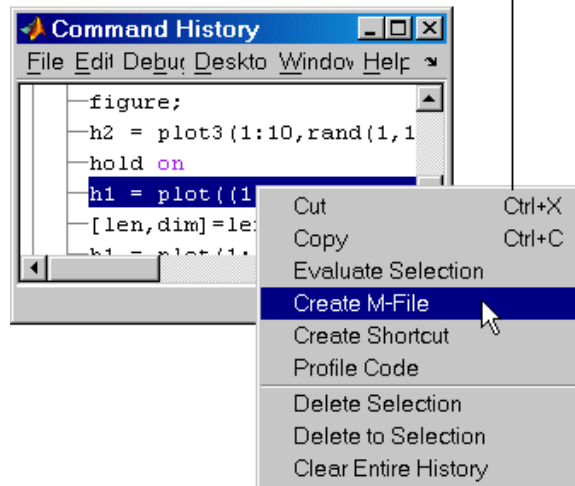
If a context menu does not appear, try right-clicking in a different part of the tool. When a context menu item is gray, the item does not apply to the current selection or area.

Using Toolbar Features

The toolbar in the desktop provides easy access to frequently used operations. Some other tools also provide toolbars. The following illustration shows some key features of the desktop toolbar.
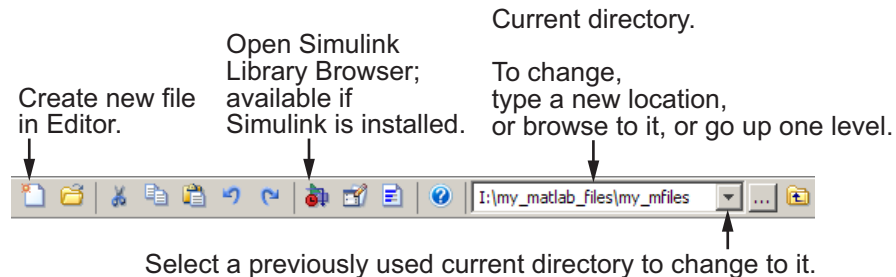
These are the major toolbar features:

- ToolTips — Position the pointer over a button for 1 or 2 seconds and a ToolTip appears describing the item.

- Customizing — You can customize the toolbar to show or remove controls, and to rearrange the controls. Use **File > Preferences > Toolbars**. For details, click **Help** in the resulting dialog box.

- Toolbars in Tools — Some tools have their own toolbars, which are located within the tool's own window. For example, the Current Directory browser has its own toolbar. When you undock one of these tools, the undocked tool includes the toolbar.

- Hiding Toolbars — To hide a toolbar, or to show it again after previously hiding it, select **Desktop > Toolbars**, and select the toolbar of interest. As an alternative, right-click a toolbar or menu bar and select a toolbar from the context menu to hide or show it. For a figure window, use figure window's **View** menu, and from it, select the toolbar of interest.

- Repositioning Toolbars — If a tool has more than one toolbar, you can change the position of the toolbars. For example, in the Editor, the default is for the Editor toolbar to be above the Cell Mode toolbar. To move a toolbar, grab the toolbar anchor (at the left end) and drag the toolbar to a different location.

To move a toolbar, grab the toolbar anchor and drag the toolbar to a new location.
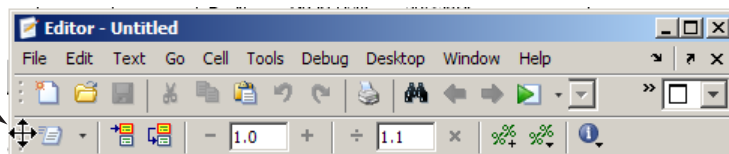
Here, the Cell Mode toolbar has been moved before the Editor toolbar.

Cell mode toolbar   Editor toolbar
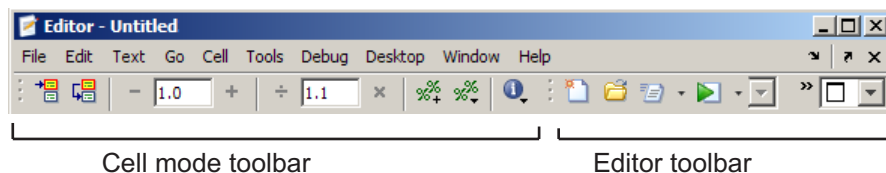
See also the "Shortcuts Toolbar" on page 2-62.

Viewing and Changing the Current Directory in the Desktop Toolbar

The current directory field in the desktop toolbar shows the current working directory in MATLAB. You can use this field to change the current directory. For more information, see "Using the Current Directory Field to View and Change the Current Directory" on page 5-58.

Viewing Status in the Status Bar

Along the bottom of the desktop is the status bar. It provides status information, such as when MATLAB is busy executing statements or when the Profiler is on. Some tools, display additional status information, such as the Editor, which displays the current line number. Not all status information appears on the status bar — many MATLAB functions and tools provide status information that is reported in dialog boxes or other places and is not reported to the status bar.

You can construct your own functions to provide status information. One method is the timer function. Use the Help browser search feature to find other specific terms describing the status you want.

Sizing, Arranging, and Sorting Columns in Desktop Tools

Some desktop tools present information in columns, such as the Current Directory browser. The following table describes how you can resize and reposition the columns, as well as sort the information in the columns.

To...	Do This...
Change the column width	Drag the separator bar between two column headings.
View all the information in a column that is too narrow to show all of it	Position the pointer over an item to view the full value for that item. It displays like a ToolTip
Rearrange the columns	Drag a column header to a different position.
Sort the information by a particular column	Click the column header. For example, in the Current Directory browser, click the **Last Modified** date to sort the items in date order. In some columns, you also can reverse the sort order—click the column header again. A small gray arrow in the header indicates the current sort order—for example, an up arrow in the **Date Modified** column header indicates an ascending sort order, meaning the oldest files are at the top of the list.

To reorder columns,
drag a column header
to a new position.

To sort by a column, click
a column heading. To reverse the
sort order, click the heading again.

To see the entire
value for a field,
hold the pointer over it.
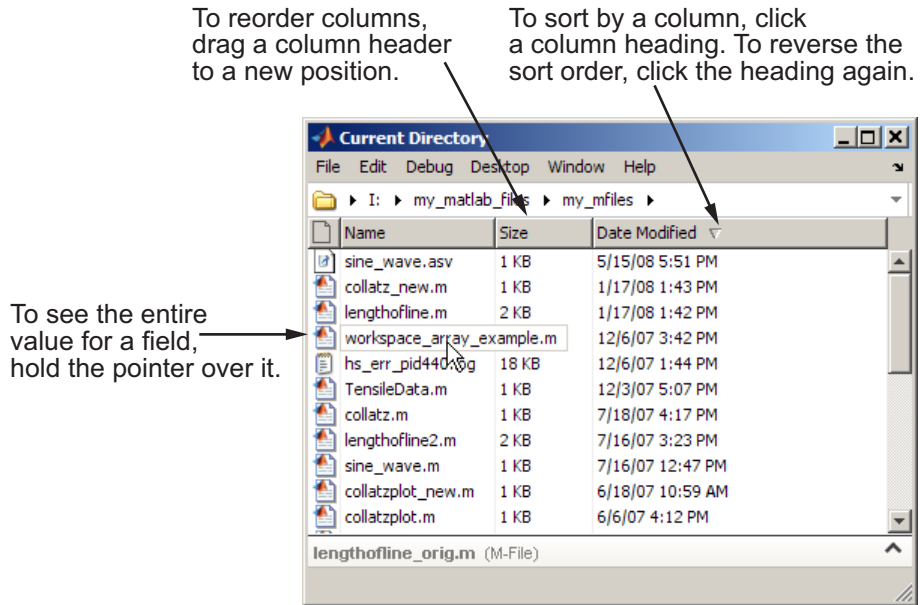
Selecting Multiple Items

In many desktop tools, you can select multiple items, and then select an
action to perform on all the selected items. Select multiple items using the
standard practices for your platform.

For example, if you run on a Microsoft Windows platform, do the following to
select multiple items:

1 Click the first item you want to select.

2 Hold the **Ctrl** key, and then click the next item you want to select.
Repeat this step until you have selected all the items you want. To select
contiguous items, select the first item, hold the **Shift** key, and then select
the last item.

Now you can perform an action on the selected items, such as delete.

To clear one of multiple selected items, **Ctrl**+click that item. To clear all
selected items, click outside of the selection.

Cut, Copy, Paste, and Move

You can cut and copy a selection from a desktop tool to the clipboard, and then paste it from the clipboard into another tool or application. You can use the **Edit** menu, toolbar, context menus, or standard keyboard shortcuts. For example, you can copy a selection of statements from the Command History window and paste them into appropriate MATLAB desktop tools, such as the Editor.

Use **Paste** to move items copied to the clipboard from other applications. The **Paste to Workspace** item in the **Edit** menu opens the selection on the clipboard in the Import Wizard. You can use this to copy data from another application, such as the Microsoft® Excel® application, into MATLAB. For details, see the "Using the Import Wizard" topic.

When editing in the Command Window and the Editor, you can move text to a new location by selecting the text and dragging it. To copy text, press **Ctrl** and drag the selected text to the new location.

To undo the most recent cut, copy, or paste command, select **Undo** from the **Edit** menu. Use **Redo** to reverse the **Undo**. For some tools, you can undo multiple times in succession.

See also the `clipboard` function.

Drag and Drop

You also can move or copy a selection from one tool to another by dragging the selection. For example, make a selection in the Command History window and drag it to the Command Window, which pastes it there. Edit the lines in the Command Window, if needed, and then press the **Enter** key to run the lines from the Command Window.

Another example is to open a file in the Editor by dragging the file name from the Current Directory browser to the Editor. If you drag editable text (for example, text in the Editor), the text is cut rather than copied. Use **Ctrl** and drag to copy rather than cut editable text.

On Windows platforms, you can drag items from external applications into MATLAB. For example, dragging text from a document created using the Microsoft Word application into the Editor cuts and pastes it into the open

file. Dragging an M-file from Windows Explorer tool to the Command Window runs the file. Similarly, you can drag selections from desktop tools to other applications. For example, you can drag text from the Editor to the Word application.

Macintosh Platform — Differences

GUI Conventions in the Documentation and Macintosh Platforms

MATLAB on the Apple Macintosh platform sometimes uses conventions that are standard for the Macintosh platform, but might be different from what is stated in the MATLAB documentation, because the documentation typically presents conventions for Microsoft Windows platforms. The intended action for the Macintosh platform is typically obvious. For example, the documentation might instruct you to select **File > Save** and to then select an option from the Save dialog box, either **Yes**, **No**, or **Cancel**, which is the convention on Windows platforms. However, on Macintosh platforms, the Save dialog box that appears presents the options **Don't Save** and **Save**.

Pointer Device Instructions and Macintosh Platforms

The standard mouse for Macintosh platforms is a single-button device. Other platforms use a mouse with more than one button. MATLAB takes advantage of these buttons. The documentation does not usually present the equivalent instructions for the Macintosh platform. When the documentation instruction is right-click, use **Ctrl**+click on the Macintosh platform. When the documentation instruction is middle-click, use **Command**+click on the Macintosh platform.

Using File Browser GUIs on Macintosh Platforms to Navigate Within the MATLAB Root Directory

On Macintosh platforms, you cannot use a file browser GUI to directly navigate to a file or directory within the MATLAB root directory (also called *matlabroot*, which is the directory where MATLAB is installed). When you use the Macintosh Finder or a file browser GUI in MATLAB and select Applications/R2008b_MATLAB, no contents appear. This is because on Macintosh platforms, the MATLAB root directory is R2008b_MATLAB.app, and

the Macintosh operating system does not display the contents of applications (items with the `.app` extension).

Here are some ways to view or open the contents of the MATLAB root directory via a file browser GUI:

- In the Macintosh Finder, right-click (or **Ctrl**+click) `MATLAB_R2008b`, and from the context menu, select **Show Package Contents**.

- In a MATLAB GUI where you cannot access the contents of `MATLAB_R2008b`, press **Command+Shift+G**, which opens the Go To Folder dialog box. In the Go To Folder dialog box, enter the full path to *matlabroot*, for example, `/Applications/MATLAB_R2008b.app/`. While typing the path in the Go To Folder dialog box, you can use autocomplete, that is, type the first few characters in the path or file name, and pause; the remaining characters matching an existing name appear. For example, type `/App` and autocompletion displays `/Applications`. Then add `/MAT`, and autocompletion displays `/Applications/MATLAB_R2008b.app/`. After entering the path, press **OK** in the Go To Folder dialog box. The MATLAB GUI then displays the contents of the MATLAB root directory.

- Use an alternative to the GUI. For example, instead of using **File > Open** to open an M-file in the Editor, use the `edit` function.

- In the Command Window, change the current directory to *matlabroot* by running `cd(matlabroot)`, and then open the GUI. Some GUIs then display the contents of *matlabroot*.

Printing and Page Setup Options for Desktop Tools

You can print from all desktop tools, except the Current Directory browser, but there are some differences in usage.

To print, select **File > Print** from the tool. A Print dialog box opens. The **Properties** button in the Print dialog box is enabled for the Web and Help browsers and the Profiler, but is not enabled for the other desktop tools.
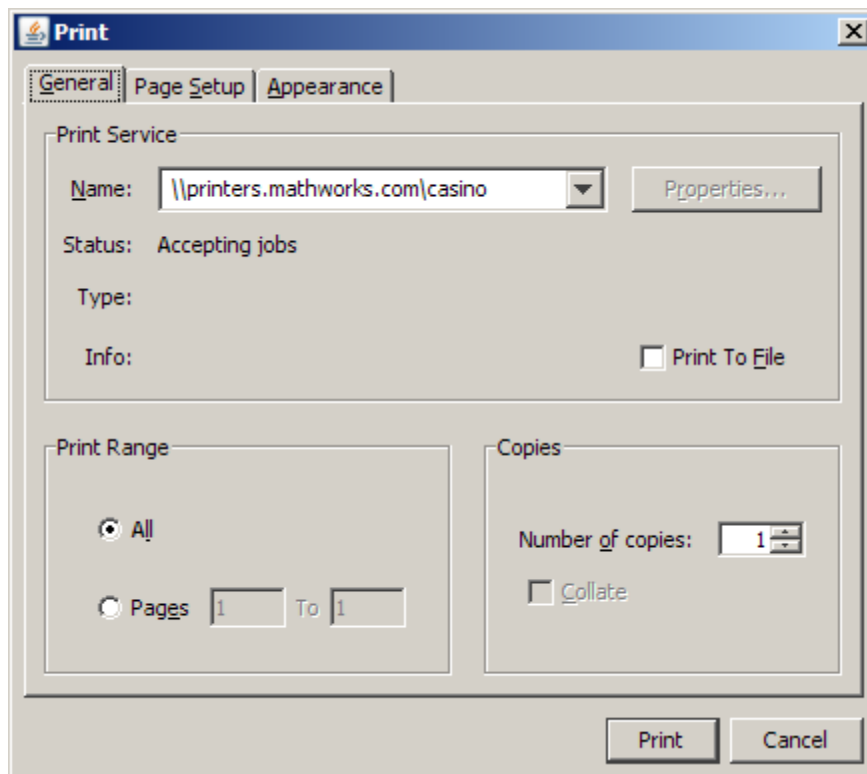
To specify standard page setup options for your platform when you print from the Command History, Workspace browser, and Variable Editor, select **File > Page Setup**. A standard page setup dialog box for your platform opens.

MATLAB provides special page setup options for printing from the Command Window and Editor. The setup options are essentially the same for both tools, with minor variations. This section covers their use:

- "Specifying Page Setup Options" on page 2-91
- "Layout Options for Page Setup" on page 2-92
- "Header Options for Page Setup" on page 2-92
- "Fonts Options for Page Setup" on page 2-92

Specifying Page Setup Options

To specify page setup options, perform these steps:

1 In the tool you want to print from, for example, the Command Window, select **File > Page Setup**.

The Page Setup dialog box opens for that tool.
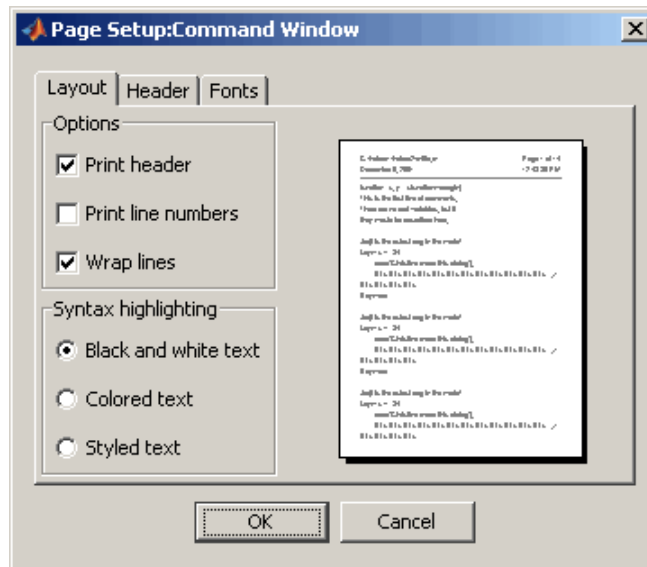
2 Click the **Layout**, **Header**, or **Fonts** tab in the dialog box and set those options for that tool, as detailed in subsequent sections.

3 Click **OK**.

4 After specifying the options, select **File > Print** in the tool you want to print from, for example, the Command Window.

The contents from the tool are printed, using the options you specified in Page Setup.

Layout Options for Page Setup

You can specify the following layout options. A preview area shows you the effects of your selections.

- **Print header** — Print the header specified in the **Header** pane.

- **Print line numbers** — Print line numbers.

- **Wrap lines** — Wrap any lines that are longer than the printed page width.

- **Syntax highlighting** — For keywords and comments that are highlighted in the Command Window, specify how they are to appear in print. Options are black and white text (that is, no highlighting), colored text (for use with a color printer), or styled text. For styled text, keywords appear in bold, comments appear in italics, and all other text appears in the normal style. Only keywords and comments you input in the Command Window are highlighted; output is not highlighted.

Header Options for Page Setup

If you want to print a header, select the **Layout** tab and then select **Print header**. Next, select the **Header** tab and specify how the elements of the header are to appear. A preview area shows you the effects of your selections:

- **Page number** — Format for the page number, for example `# of n`

- **Border** — Border style for the header, for example, `Shaded box`

- **Layout** — Layout style for the header. For example, `Standard one line` includes the date, time, and page number all on one line

Fonts Options for Page Setup

Specify the font to be used for the printed contents:

1 From **Choose font**, select the element, either `Body` or `Header`, where `Body` text is everything except the `Header`.

2 Select the font to use for that element. For example, select **Use Command Window font** for `Body` text if you want the printed text to be the same as the font that appears in the Command Window. This is the font specified in **File > Preferences > Fonts > Custom** for the `Command Window`.

3 Repeat for the other element. If you did not select **Print header** on the **Layout** pane, you do not need to specify the Header font. As an example, for Header text, select **Use custom font** and then specify the font characteristics—type, style, and size. After you specify a custom font, the **Sample** area shows how the font will look.

Displaying Documents and Web Sites in Web Browsers
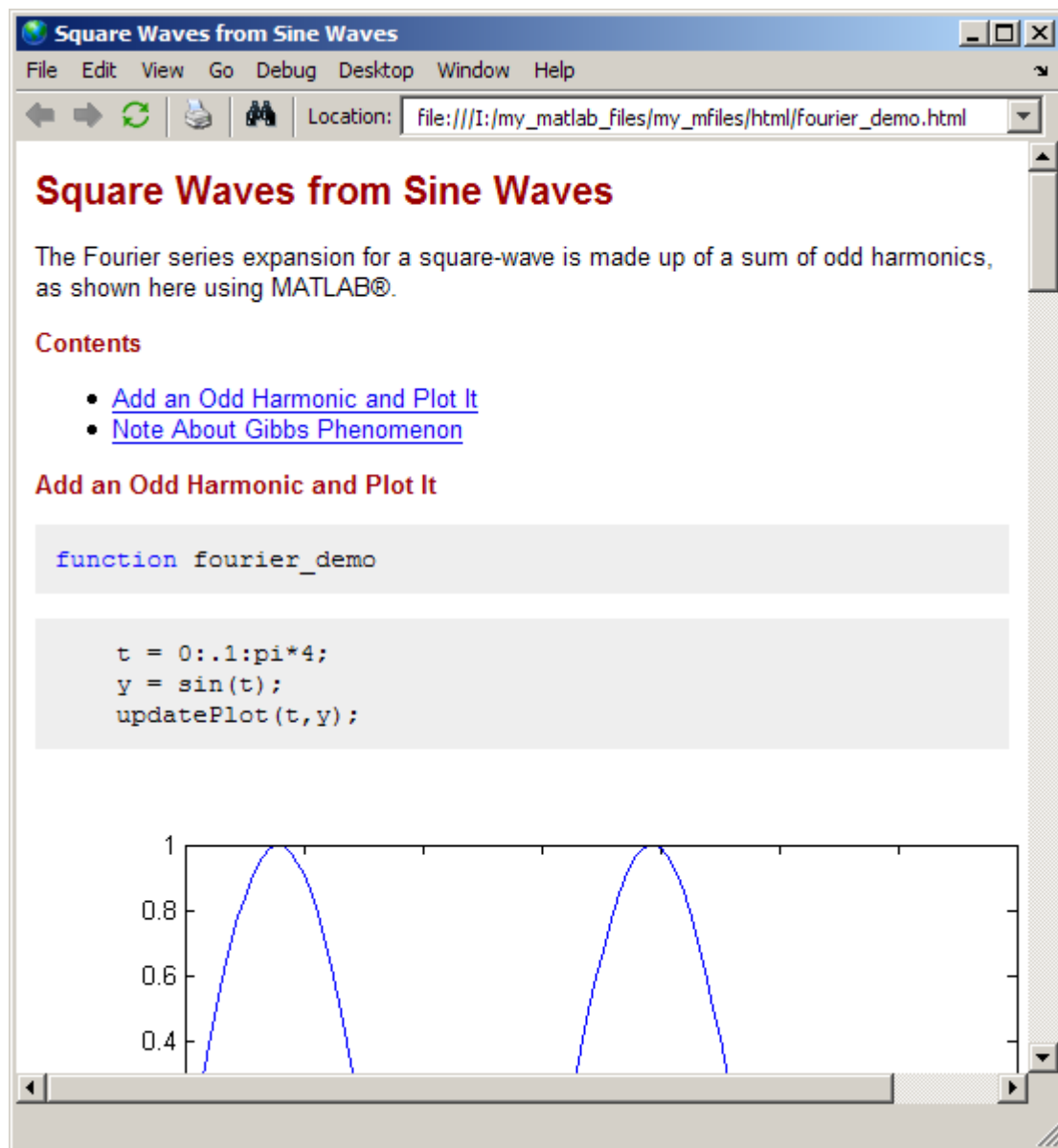
- "About the MATLAB Web Browser, the Help Browser, and Your System Web Browser" on page 2-93

- "How You Can Display Pages in Web Browsers" on page 2-95

- "Web Preferences" on page 2-95

About the MATLAB Web Browser, the Help Browser, and Your System Web Browser

From MATLAB, HTML and related documents and Web sites display in the MATLAB Web browser, the Help browser, or your system Web browser, such as Mozilla® Firefox®.

Here is how tools in MATLAB use the different types of browsers:

- In the Help browser, links to other documentation pages display in the Help browser.

- Links to Web sites from the Help browser and from other tools, such as the **Start** button links to the MathWorks Web site, display in your system browser. On Microsoft Windows and Apple Macintosh platforms, the system browser MATLAB uses is the default browser for your operating system. On UNIX platforms, the default system browser MATLAB uses is Mozilla Firefox, but you can specify a different system browser for MATLAB with Web preferences.

- HTML files display in the MATLAB Web Browser, such as an M-file published to HTML using the Editor's cell features, as shown in the following figure.

How You Can Display Pages in Web Browsers

You can display Web sites and your own HTML and related files from MATLAB in these ways:

- To display an HTML document in the MATLAB Web Browser, double-click the document name in the Current Directory browser.

- To open the MATLAB Web Browser without a document in it, select **Desktop > Web Browser**. Then, to go a Web site or an HTML page, type the URL or the full path to a file name in the **Location** field.

- In the MATLAB Web browser, to open a link in a separate window, use the middle mouse button, if you have one.

- Use the web function to open a browser in MATLAB. You can specify a URL or file to display and the browser to use. You also can use the web function in conjunction with methods that operate on a specified browser, such as a method to close a browser.

Because the MATLAB Web and Help browsers are desktop tools, you can perform desktop operations on them, such as docking the tools. The toolbar buttons and menu items in the Web Browser are similar to those found in the Help browser display pane. For more information, see "Viewing Documentation in the Help Browser" on page 4-28.

Limitations of the MATLAB Web Browser. Like any Web browser, the MATLAB Web Browser might not support all of the HTML or related features used in a particular Web site or HTML page. For example, the MATLAB Web Browser does not support the display of .bmp (bitmap) image files. Instead use .gif or .jpeg formats for image files in HTML pages.

Web Preferences

You use Web preferences to specify characteristics related to accessing the Internet, either from the MATLAB Web browser or through other features, like the sendmail function:

- "Specifying Proxy Server Settings for Accessing the Internet from MATLAB" on page 2-96

- "Specifying the System Browser MATLAB Uses for UNIX Platforms" on page 2-98

• "Specifying Fonts for the MATLAB Web Browser" on page 2-100

Specifying Proxy Server Settings for Accessing the Internet from MATLAB. If your network uses a firewall or another means of protection that restricts Internet access, and you want to access the Internet from MATLAB, you need to provide information about your proxy server to MATLAB.

The following MATLAB features access the Internet and rely on the proxy server settings specified in Web preferences:

• MATLAB Web browser and `web` function, as well as features that use them, such as links to the Web from the Help browser, and links to the MathWorks Web site from the **Help** menu and **Start** button.

• `sendmail` and features that use it, such as the Error Log Reporter, which reports abnormal termination incidents to The MathWorks.

• `urlread` and `urlwrite` functions.

• `ftp` functions.

• `createClassFromWsdl` function.

To specify the proxy server settings:

1 Select **File > Preferences > Web**.

2 Set or clear the **Use a proxy server to connect to the Internet** check box.

   • By default, the check box is not selected. If you have a direct connection to the Internet, be sure the that check box is cleared.

   • If you use a proxy server, select the check box, and specify values for **Proxy host** and **Proxy port**. These are examples of acceptable formats for the host: `172.16.10.8` and `ourproxy`. For the port, you can enter only an integer, such as `22`. (These values are provided for illustrative purposes only and are not valid.) If you do not know the values for your proxy server, ask your system or network administrator for the information.

3 If your proxy server requires you to provide a user name and password, select the **Use a proxy with authentication** check box, and enter your proxy user name and password.

The password is stored without encryption in your `matlab.prf` file.

4 To ensure your settings are working, click the **Test connection** button, which attempts to connect to `http://www.mathworks.com`:

- If MATLAB is able to access the Internet, **Success!** appears next to the button.

- If MATLAB cannot access the Internet, **Failed!** appears next to the button. Correct the values you entered and try again. If you still cannot connect, try using the values you used when you authenticated your MATLAB license.

5 Click **OK** to accept the changes.
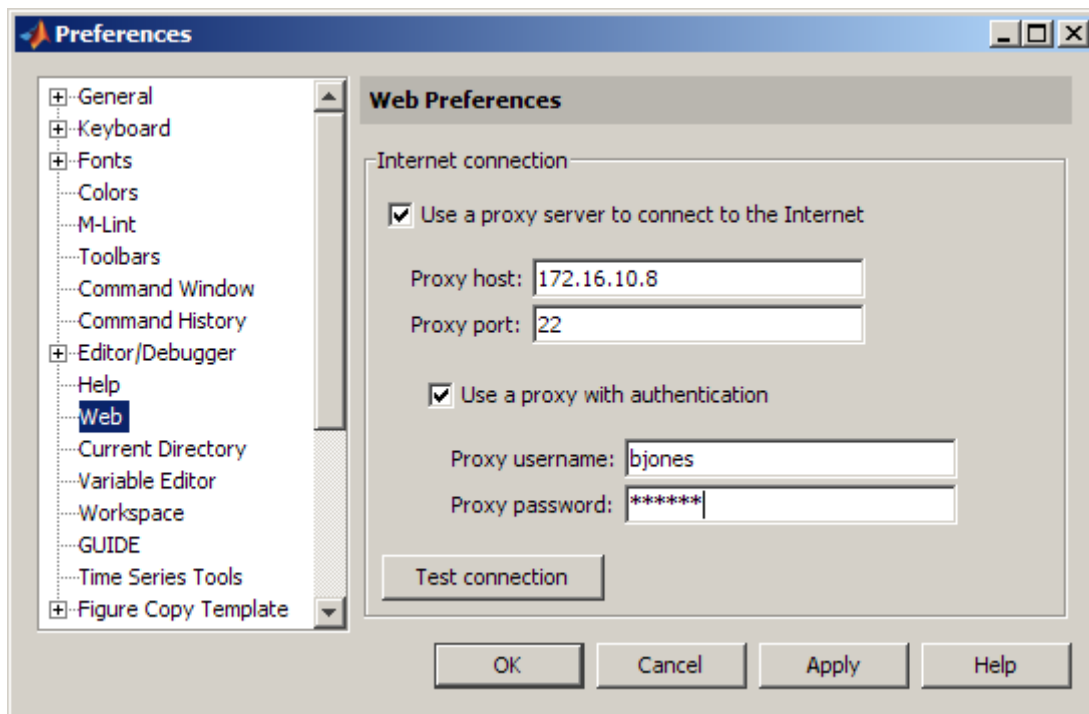
After specifying the preference, you can access the Internet from MATLAB through your proxy server.

These are limitations of specifying the proxy server settings:

- You cannot specify the proxy server settings using a script.

- You cannot automatically provide the proxy server settings your system browser uses to MATLAB.

Specifying the System Browser MATLAB Uses for UNIX Platforms.
On UNIX platforms (not including the Apple Macintosh platform), when MATLAB uses a system browser, by default it uses the Mozilla Firefox browser. If you do not have Firefox, or if you want to use a different system browser for MATLAB, use Web preferences as follows:

1 Select **File > Preferences > Web**. The Preferences dialog box opens to
the Web pane.
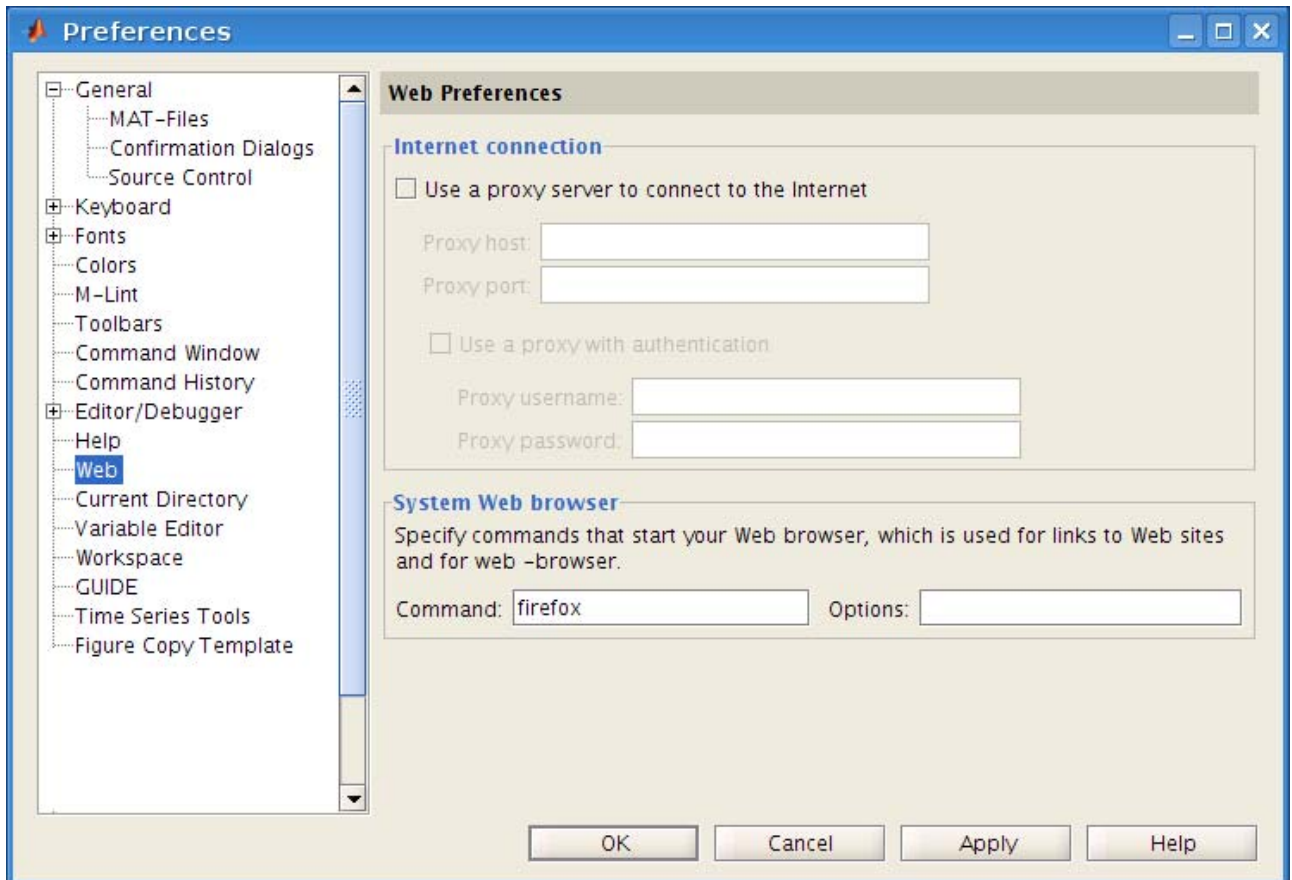
Note The **System Web browser** preference is only for UNIX platforms
(except Macintosh) and does not appear in the preferences pane for other
platforms.

2 For **System Web browser**, in the **Command** field, specify the system command to open the browser, for example, `opera`, which opens the Opera Web browser.

3 To instruct your system browser to open using options that it supports, enter them in the **Options** field. For example, `geometry 1064x860` specifies the size of the window for Opera.

4 Click **OK**.

The next time MATLAB displays documents in a system browser, it will use the browser specified in Web preferences.

For background information, see "About the MATLAB Web Browser, the Help Browser, and Your System Web Browser" on page 2-93.

Specifying Fonts for the MATLAB Web Browser. To modify the font used in the MATLAB Web Browser, select **File > Preferences > Fonts**. The Web browser uses the font settings that you specify for the HTML Proportional Text tool. For more information about setting fonts, click the **Help** button in the preference pane for **Fonts**.

Accessing The MathWorks on the Web

You can access popular pages on the MathWorks Web site from the MATLAB desktop.

To download trial versions of products that you do not have, select **Help > Get Product Trials**.

For access to other popular Web site pages, select one of the following items from the **Help > Web Resources** menu. The selected Web page opens in your system Web browser:

- **The MathWorks Web Site** — Home page of the MathWorks Web site (`http://www.mathworks.com`).

- **Products & Services** — MathWorks Products and Services page (`http://www.mathworks.com/products/`) with information about the full family of products.

- **Support** — MathWorks Support page
  (`http://www.mathworks.com/support`) where you can look for solutions to problems, or report new problems.

- **Training** — List of courses for learning to use MathWorks products
  (`http://www.mathworks.com/services/training/courses/`).

- **MathWorks Account** — Login page for MathWorks Account
  (`http://www.mathworks.com/accesslogin/`). If you are registered, your main account page displays. Otherwise, you are directed to a page where you register online. Registration allows you to view your product registration and license information and helps you stay up to date on the latest developments for MATLAB.

- **MATLAB Central** — MATLAB Central Web site
  (`http://www.mathworks.com/matlabcentral/`) for the user community for MATLAB. It includes contests for MATLAB and results, and a screen saver with the logo for MATLAB.

- **MATLAB File Exchange** — Code library of files contributed by MathWorks customers and employees, available for free download and use with MathWorks products.

- **MATLAB Newsgroup Access** — Provides access to the Usenet newsgroup for MATLAB and related products, `comp.soft-sys.matlab`, where you can post and answer questions, as well as view the archives.

- **MATLAB Newsletters** — Access to online versions of News and Notes and MATLAB Digest. News and Notes is published twice a year and contains feature articles, technical notes, and product information for users of MATLAB. MATLAB Digest, an electronic bulletin consisting of technical notes, solutions, and timely announcements to the user community, is issued more frequently. See `http://www.mathworks.com/company/newsletters`.

Managing Your Licenses

You can use the MATLAB licensing features to perform license management activities, such as activating licenses, deactivating licenses, or updating licenses. You also can visit the License Center at the MathWorks Web site to perform other license-related activities.

To access the licensing feature:

1 Select **Help > Licensing**.

2 Select the activity you want to perform from the **Licensing** menu. The following table describes the options. Depending on your license type, the **Licensing** menu on your system might not include all options.

Note Some options require an Internet connection. If your Internet connection requires a proxy server, use MATLAB Web preferences to specify the server host and port. See "Specifying Proxy Server Settings for Accessing the Internet from MATLAB" on page 2-96 for more information.

Option	Description
Activate Software	Starts the activation application, which walks you through the activation process. Answer the questions on each dialog box, select the license you want to activate, and click **Activate**.
Deactivate Software	Displays a list of all your MathWorks licenses on this computer, with their current status. When you select a license and click **Deactivate Selected License**, MATLAB deactivates all releases on this computer associated with the license, and updates the licensing information at the MathWorks Web site. You will not be able to use MathWorks software with that license on this computer.

If you are not connected to the Internet, MATLAB deactivates the licences on your computer but cannot update the corresponding license information stored at the MathWorks Web site. In this scenario, MATLAB returns a *deactivation string*. To complete deactivation, save a copy of this string, go to a computer with an Internet connection, and visit the License Center at the MathWorks Web site. There you can login to your MathWorks Account and enter the deactivation string. |

Option	Description
Update Current Licenses	Displays a list of all your MathWorks licenses on this computer, with their current status. When you select a license and click **Update Selected License**, MATLAB contacts The MathWorks to retrieve the most current version of the License File for the license. The update process overwrites the current License File on your system. You will need to restart MATLAB.
Manage Licenses	Starts a Web browser, opening the My Licenses page associated with your MathWorks Account. You can use this page, called the License Center, to perform many licensing activities.

Check for Updates

To determine if more recent versions of your MathWorks products are available, and to view latest version numbers for all MathWorks products, use the Check for Updates feature.

To access the Check for Updates feature, you must have an active Internet connection. Then, follow these steps:

1 Select **Help > Check for Updates**. The Check for Updates dialog box displays.

2 From the **Select View** list, choose to view the latest version numbers for all MathWorks products installed on your system, or all MathWorks products. The latest versions are displayed.

3 Click any column heading to sort or reverse the sort order by that column.

4 Use the What's New column to access the release notes for a product. Release notes document new features and changes, bug reports, and compatibility considerations.

5 To upgrade to the most recent version, click **Download Products at MathWorks.com**, which links to the Downloads area of the MathWorks Web site. If you do not want to upgrade at this time, click **Close**.

Specifying Options for MATLAB Using Preferences

In this section...

Setting Preferences for MATLAB

Use preferences to specify options for MATLAB tools, as follows:

1 Select **File > Preferences**. Alternatively, click the Preferences button
on the desktop toolbar; if the button is not on the toolbar, you can add
it—for information, see "Setting Toolbars Preferences for Desktop Tools" on
page 2-138.

2 From the left pane of the Preferences dialog box, choose a tool or product
and click the **+** to display more preferences for that item. From the
expanded list, select the entry you want. The right pane shows the
preferences for that item.

3 Change settings. Click **Apply** or **OK** to set the preferences. Preferences
take effect immediately. They remain persistent across sessions of
MATLAB.

Note that some tools allow you to control these settings from within the tool
without setting a preference. Use that method if you want the change to apply
only to the current session.

Function Alternative

Open the Preferences dialog box using the preferences function.

Summary of Preferences

Preference	What You Can Specify
General Preferences	Toolbox path caching, figure window printing, `delete` function behavior, MAT-file save formats, confirmation dialogs, and source control.
Keyboard	Key bindings, tab completion, and delimiter matching for the Command Window and the Editor.
Fonts	Font type, style, and size for desktop tools. Customize for any tool.
Colors	Colors for text, background, syntax highlighting, and hyperlinks in desktop tools.
M-Lint	Show or hide M-Lint messages in the Editor M-Lint automatic code analyzer and in the M-Lint Code Check Report.
Toolbars	Remove, add, and rearrange controls on toolbars for some desktop tools.
Command Window	Numeric format and display, accessibility, and tab size.
Command History	Display, filtering, and saving.
Editor/Debugger	Editor type, startup options, display, tab size and indenting, language, including M-Lint messages, and autosave.
Help	Product filter and synchronization.
Web	Internet proxy server settings.
Current Directory	Number of entries in history and display options.
Variable Editor	Numeric format, use of **Enter** key, and decimal separator.
Workspace	Statistical calculation options.
GUIDE	Display options for GUI-building tool.
Time Series Tools	Property Editor dialog and x-axes warning dialog. For details, click the **Help** button in the Preferences dialog box.
Figure Copy Template	Application, text, line, uicontrols, axis, format, background color, and size.
Other products	Preferences for other installed MathWorks products.

Preferences File — matlab.prf

MATLAB and other MathWorks products store their preferences in the file
`matlab.prf`. Type `prefdir` in the Command Window to see the full path for
the directory where `matlab.prf` is located, called the preferences directory.
The preference directory also contains other related files.

On Apple Macintosh platforms, the directory might be in a hidden folder, for
example, `myname/.matlab/R2009a`. To access the directory, select **Go > Go to
Folder** in the Apple Mac OS® Finder tool. In the resulting dialog box, type
the path returned by `prefdir` and press **Enter**.

The `matlab.prf` file loads when you start MATLAB. When you make changes
to preferences while using MATLAB, it makes the changes to `matlab.prf`.
When you closeMATLAB, it saves those changes to `matlab.prf`.

The exact name of the preferences directory that MATLAB uses depends on
the release. After you install a new version of MATLAB and startMATLAB,
it tries to use your existing preferences from the previous version, where
possible. For more information on the preference directory name and the
preference migration process, see the `prefdir` reference page.

Setting General Preferences for the MATLAB Application

In this section...

General Preferences

Select **File > Preferences > General** from any desktop tool to access
General Preferences.

These preferences apply to all relevant tools in the MATLAB application.
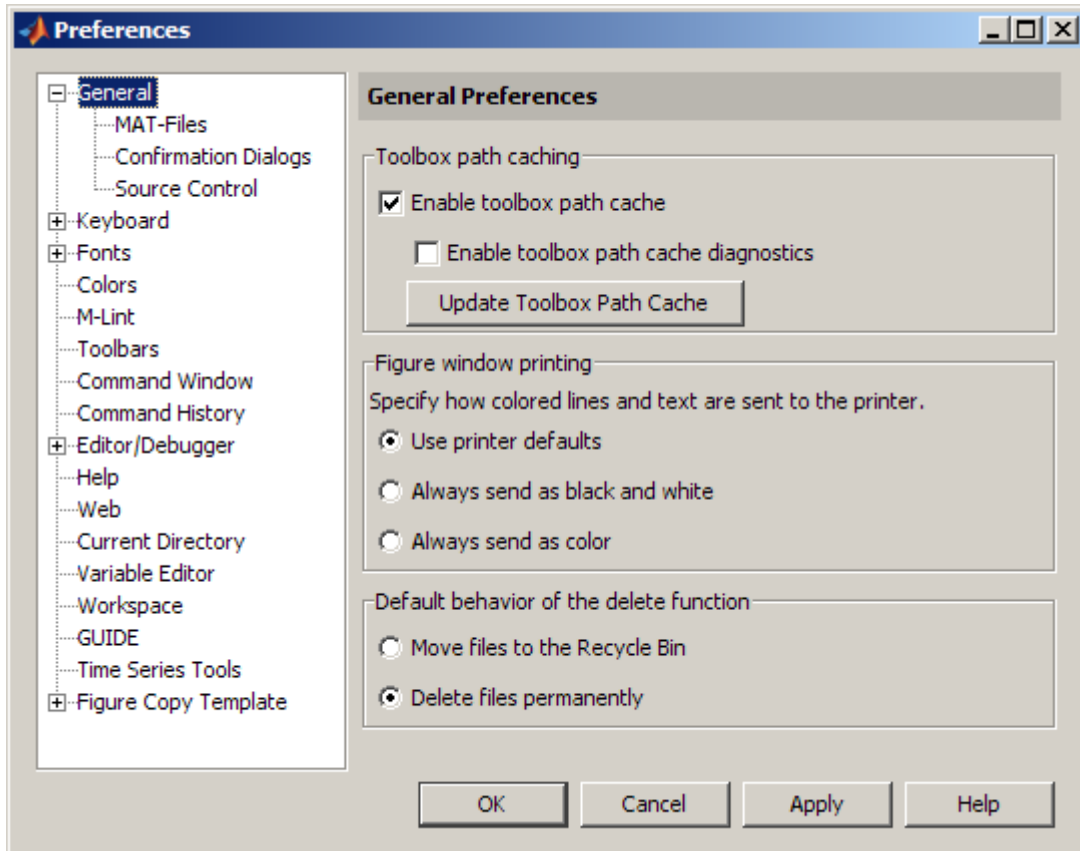
Toolbox Path Caching

For information, see "Toolbox Path Caching in the MATLAB Program" on page 1-22.

Figure Window Printing

For information, see "Printing and Exporting" in MATLAB Graphics documentation.

Default Behavior of the Delete Function

Files you delete using the `delete` function are permanently removed by default. There is no opportunity to retrieve them.

You can use this preference to instead move deleted files to the Recycle Bin on Microsoft Windows, to the Trash Can on Apple Macintosh, or to a `tmp` directory on UNIX[7] platforms. Then, you can recover any accidentally deleted files from these locations. Deleted files in these locations are not automatically removed; you must remove them using operating system features, such as **Empty Recycle Bin** on the Windows platform. When you select this preference, `delete` might run slower.

Function Alternative. The delete preference for MATLAB actually sets the state of the `recycle` function upon startup and when you change the preference. You can override the behavior of the preference by setting the `recycle` function state. For example, regardless of the preference setting, when you run

```
recycle('off')
delete('thisfile.m')
```

MATLAB permanently removes `thisfile.m` from the current directory. Files you subsequently remove using `delete` are also permanently removed, unless you reapply the preference to recycle or run `recycle('on')`. Regardless of the state of the `recycle` function when you end a session, the next time you start MATLAB, the setting for the preference is honored. For more information, see the `recycle` and `delete` reference pages.

Note that this preference and the `recycle` function do not apply to files you delete using the Current Directory browser. For more information, see "Cutting and Deleting Files and Directories Using the Current Directory Browser" on page 5-75.

MAT-Files Preferences

The **MAT-file save format** sets the default version compatibility option MATLAB uses when saving MAT-files. Use these options if you use multiple

---

7. UNIX is a registered trademark of The Open Group in the United States and other countries.

versions of MATLAB or share MAT-files with others who run a different version of MATLAB. The setting applies when you use the `save` function as well as when you use **Save** menu items for MAT-files, such as **File > Save Workspace As** from any desktop tool.

The MAT-file preference also applies to saving FIG-files, which include plots, as well as GUIs you create with GUIDE.
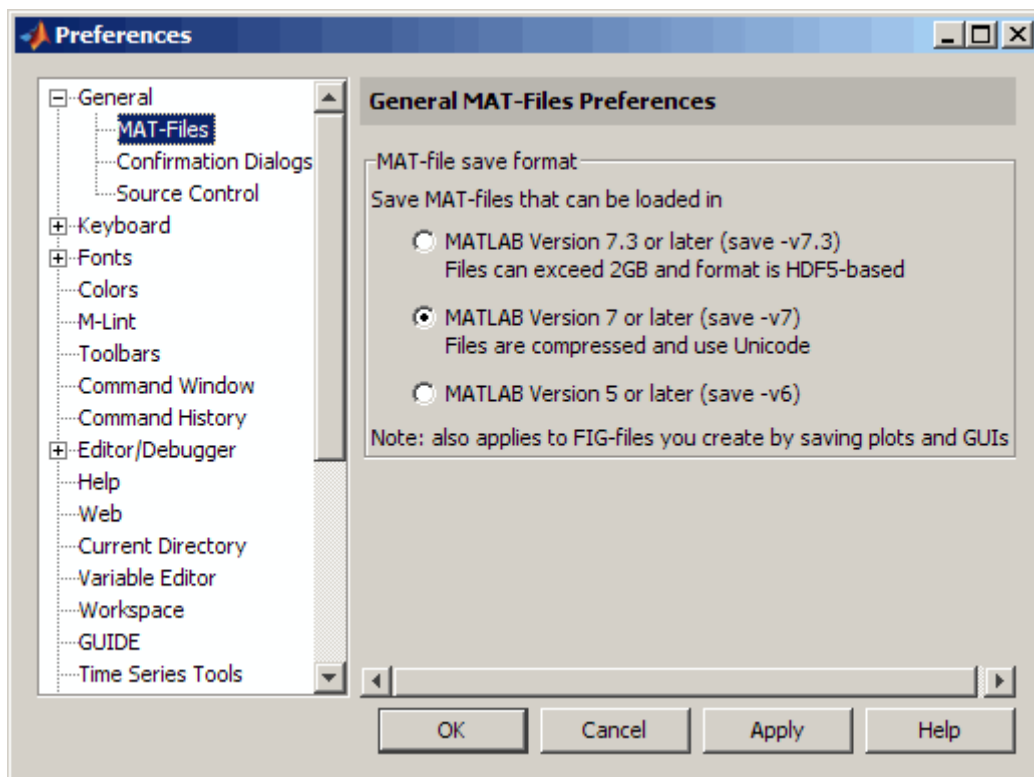
Options are

- **MATLAB Version 7.3 or later (save -v7.3)** — Starting in MATLAB Version 7.3, you can save data that is larger than 2 GB on platforms that allow it, which is the primary purpose of this option. Using this option is equivalent to running `save -v7.3`. This format of the resulting MAT-file is

HDF5-based. You cannot load these MAT-files into any versions prior to MATLAB Version 7.3; in those cases, use one of the other two options.

- **MATLAB Version 7 or later (save -v7)** — Starting in MATLAB Version 7, MATLAB compresses the data when saving a MAT-file, thereby reducing the storage space required. When you load the MAT-file, MATLAB automatically uncompresses the data. In addition, MATLAB uses Unicode® character encoding for strings when you save a MAT-file, making the data accessible to other users of MATLAB, regardless of the default character encoding scheme used by their systems. MAT-files saved with this option work in all MATLAB 7 versions. Using this option is equivalent to running `save -v7`.

- **MATLAB Version 5 or later (save -v6)** — Releases of MATLAB prior to Version 7 did not save compressed MAT-files. They also did not use Unicode character encoding, which sometimes prevented the exchange of MAT-files among users, particularly when they used localized systems. Specify this option to save MAT-files for use with versions prior to MATLAB Version 7. Using this option is equivalent to running `save -v6`.

Like other preferences, the **MAT-file save format** preference gets its initial value from the preference file for the previous installed version. For example, if the setting in your MATLAB 7.6 preference is `-v6`, when you upgrade from MATLAB Version 7.6 (R2008a) to MATLAB Version 7.8 (R2009a), the initial value in Version 7.8 is `-v6`.

If you upgrade from a version prior to MATLAB Version 7.3, or if you do not have a previous MATLAB version installed, the initial value is `-v7`.

Note For more information about MAT-file save formats, including restrictions, see Version Compatibility Options and Remarks in the `save` reference page.

Function Alternative

You can override the **MAT-file save format** preference by using the `save` function with a specified version compatibility option. For occasional use, this might be more convenient than changing the preference. For example, use

save with the -v6 option to ensure compatibility with MATLAB versions prior to Version 7. For more information, see the save reference page.

Confirmation Dialogs Preferences

These preferences instruct MATLAB to display or not display specific confirmation dialog boxes.
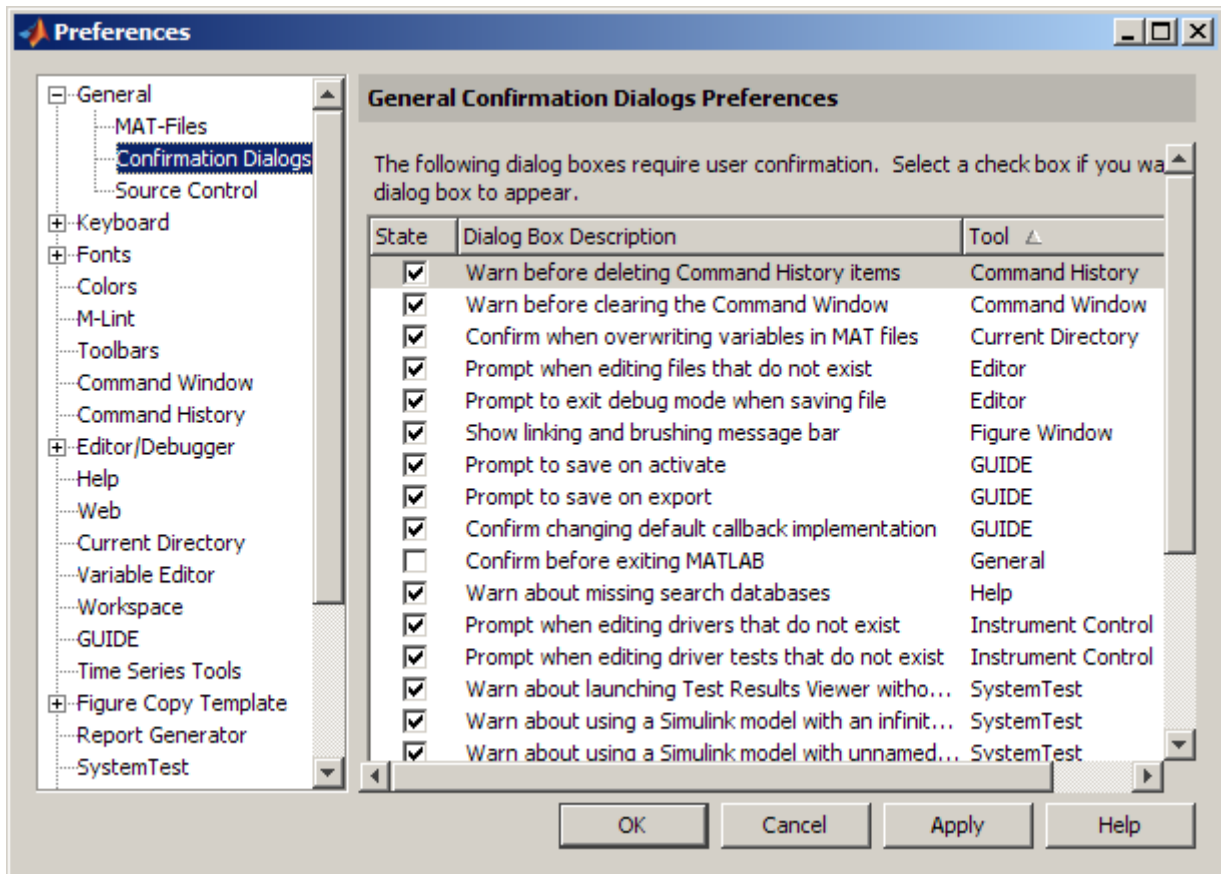
When the check box for a confirmation dialog is selected and you perform the action it refers to, the confirmation dialog box appears. If you clear that check box, the dialog box does not appear when you perform the action.

When the confirmation dialog box does appear, it includes a **Do not show this prompt again** check box. If you select the check box in the dialog box, it automatically clears the check box for the confirmation preference.

For example, select the check box **Warn before deleting Command History items**. Then select **Edit > Clear Command History**. MATLAB displays the following confirmation dialog box.
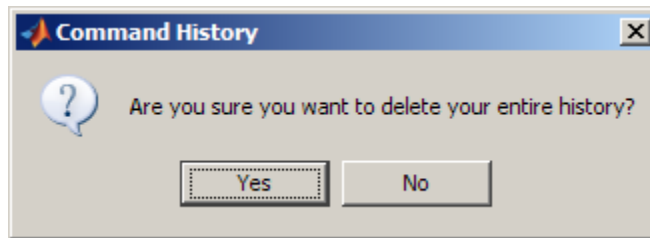
If a confirmation dialog box includes a **Do not show this prompt again** check box and you click **OK**, the confirmation dialog box will not appear the next time you perform the action. In addition, the check box in the **Confirmations Dialogs** preferences pane is cleared.

The following table summarizes the confirmation dialog boxes for MATLAB. There might be additional confirmation dialog boxes listed for other products you have installed.

Confirmation Dialogs Check Box Item	Description of the Confirmation Dialog Box	For More Information
Warn before deleting Command History items	Appears when you delete entries from the Command History window.	"Deleting Entries from the Command History Window" on page 3-74
Warn before clearing the Command Window	Appears when you clear the Command Window content using menu items. Does not appear when you use the `clc` function.	"Clearing the Command Window" on page 3-51

Confirmation Dialogs Check Box Item	Description of the Confirmation Dialog Box	For More Information
Confirm when overwriting variables in MAT-files	Appears when you save variables by dragging them from the Workspace browser onto a MAT-file in the Current Directory browser.	"Creating and Updating MAT-Files in the Current Directory Browser" on page 5-70
Prompt when editing files that do not exist	Appears when you type `edit filename`, if `filename` does not exist in the current directory or on the search path.	"Function Alternative for Creating New Files" on page 6-9
Prompt to exit debug mode when saving file	Appears when you try to save a modified file while in debug mode.	"Ending Debugging" on page 6-171
Show linking and brushing message bar	Displays the message bar in a figure window that provides links to help for data brushing and linked plots.	"Exploring Data in Graphs"
Prompt to save on activate	Appears when you have unsaved changes to a figure and M-file, and then activate the GUI, by clicking the **Run** button, for example.	"GUIDE Preferences" in the MATLAB Creating Graphical User Interfaces documentation
Prompt to save on export	Appears when you have unsaved changes to a figure and M-file, and then select **File > Export**.	"GUIDE Preferences" in the MATLAB Creating Graphical User Interfaces documentation
Confirm changing default callback implementation	Appears after you have modified a callback signature in GUIDE.	"Changing Callbacks Assigned by GUIDE" in the MATLAB Creating Graphical User Interfaces documentation

Confirmation Dialogs Check Box Item	Description of the Confirmation Dialog Box	For More Information
Confirm before exiting MATLAB	Appears when you quit MATLAB.	Quitting MATLAB
Warn about missing search databases	Appears if you have help files in the Help browser for non-MathWorks products and the search database for those files has not been updated for the version of MATLAB you are running.	Contact the provider of the help files to obtain the correct version of the search database. Without the most current version, you can use the help files in the Help browser, but the Help browser search will not include those files in search results.
Confirm when deleting variables	Appears when you delete variables from the workspace using menu items. Does not appear with the `clear` function.	"Deleting Workspace Variables" on page 5-9

Source Control Preferences

For information, see Chapter 10, "Source Control Interface".

Customizing the Desktop Using Preferences

In this section...

Setting Keyboard Preferences for Desktop Tools

- "Overview of Keyboard Preferences" on page 2-116
- "Command Window Key Bindings Preferences" on page 2-117
- "Editor/Debugger Key Bindings Preferences" on page 2-118
- "Tab Completion Preferences" on page 2-118
- "Tabs and Indents Preferences" on page 2-119
- "Function Hints Preferences" on page 2-119
- "Delimiter Matching Preferences" on page 2-119

Overview of Keyboard Preferences

To set key binding, tab completion, delimiter matching, and function hints preferences for the Command Window and the Editor, select **File > Preferences** and then select **Keyboard** in the left pane of the Preferences dialog box.
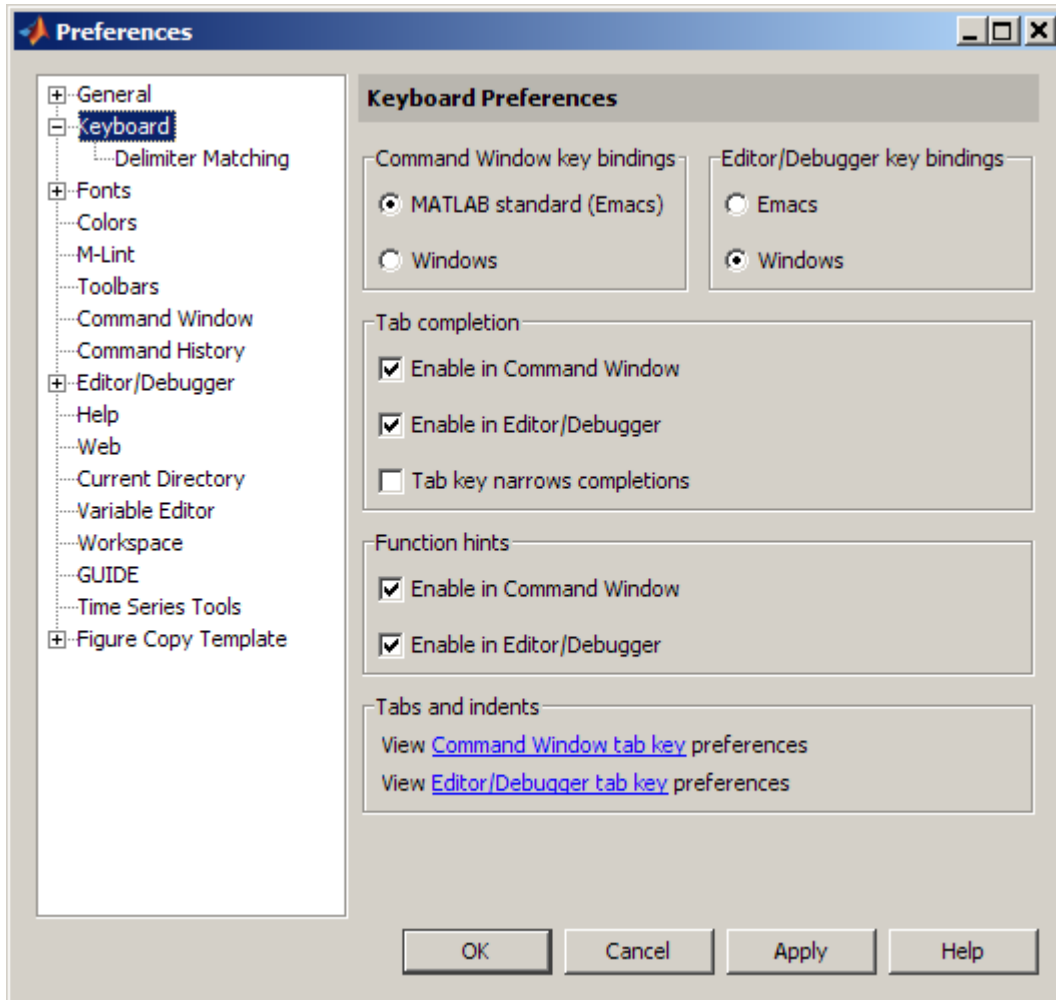
Command Window Key Bindings Preferences

Specify the keyboard shortcuts (key bindings) to be used at the command line.

MATLAB standard (Emacs). Allows you to use the control keys listed in "Keyboard Shortcuts in the Command Window" on page 3-19, which should be familiar to existing users of MATLAB and Emacs software. For example, **Ctrl+A** moves the cursor to the beginning of the line.

Windows. Allows you to use standard control keys for Microsoft Windows platforms. For example, **Ctrl+A** is the shortcut for **Edit > Select All**, which selects the entire contents of the Command Window.

Macintosh. This option is available only on Apple Macintosh platforms. It allows you to use keys found on Macintosh keyboards, such as the **Command** key instead of the **Ctrl** key.

Editor/Debugger Key Bindings Preferences

Specify the keyboard shortcuts (key bindings) to be used by the Editor and for debugging. The Editor/Debugger key bindings are also used by other tools, for example, the **Callback** field in the Shortcut Editor dialog box.

Select **Windows**, **Emacs**, or **Macintosh** (available only on Macintosh platforms), depending on which convention you want the Editor to follow for accelerators and shortcuts. The accelerators on the menus change after you change this option.

For example, when you select key bindings for Windows platforms, the shortcut to paste a selection is **Ctrl+V**. When you select Emacs key bindings, the shortcut to paste a selection is **Ctrl+Y**. When you select key bindings for Macintosh platforms, the shortcut to paste a selection is **Command+V**. You can see the accelerator on the **Edit** menu for the **Paste** item.

Tab Completion Preferences

Enable in Command Window. Select the check box to use tab completion when typing functions in the Command Window—for more information about the feature, see "Completing Statements in the Command Window — Tab Completion" on page 3-25. Clear the check box if you do not want to use the tab completion feature. With the tab completion preference cleared, when you press the **Tab** key, MATLAB moves the cursor to the next tab stop rather than completing a function—see also the preference for "Tab size" on page 3-64.

Enable in Editor/Debugger. Select the check box to use tab completion when typing functions in the Editor—for more information about the feature, see "Completing Statements in the Command Window — Tab Completion" on page 3-25. Clear the check box if you do not want to use the tab completion feature. With the tab completion preference cleared, when you press the **Tab** key, MATLAB moves the cursor to the next tab stop rather than completing a function. For related information, select **File > Preferences > Editor/Debugger > Tab**, and click **Help**.

Tab key narrows completions. Select this check box to narrow the list of possible completions shown by typing another character and pressing **Tab**. For details, see "Narrowing Completions Shown" on page 3-28.

Tabs and Indents Preferences

The links go to the panes where you can view and set preferences for

- **Tab** key size in the Command Window, which is used when the tab completion preference is not set

- **Tab** key size and indenting preferences in the Editor/Debugger

Function Hints Preferences

To show function hints in the Command Window and Editor, select the function hints check boxes. If you do not want to use function hints, clear the check boxes. Function hints are a reminder of the syntax for a function that you use while entering a statement. The hints appear in a temporary pop-up window when you enter the opening parenthesis after a function name. For more information, see "Viewing Function Syntax While Entering a Statement — Function Hints" on page 3-34.

Delimiter Matching Preferences

To set these preferences, select **File > Preferences > Keyboard > Delimiter Matching** . These preferences apply to the Command Window and the Editor.

With these preferences selected, MATLAB alerts you to matched and unmatched delimiters based on the MATLAB language syntax rules. For

example, when you type a parenthesis or another delimiter, MATLAB highlights the matched parenthesis or delimiter in the pair.

Delimiter pairs are parentheses ( ), brackets [ ], and braces { }. For the Editor, paired language keywords are also matched. Paired language keywords include for, if, while, else, and end statements.

In the following illustration, MATLAB underlines the left parenthesis in the pair when you move over the right parenthesis using an arrow key.

```
len(nl) = sum(sqrt(dot(temp',temp')));
```

If the matching delimiter is not visible on the screen, a pop-up window appears and shows the line containing the matching delimiter. In the Editor, the line number is included. Click in the pop-up window to go to that line.
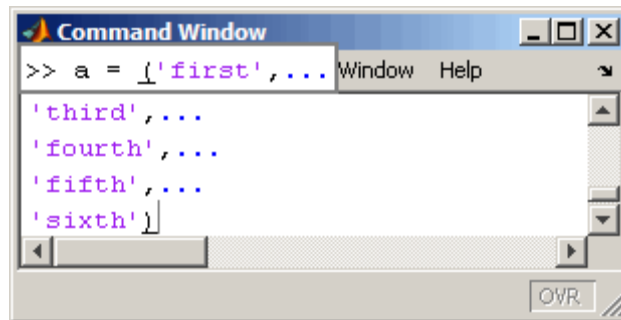
Match while typing. Select the check box if you want to be alerted to matches and mismatches in pairs of delimiters as you type them. Then choose how you want MATLAB to alert you to matches by selecting an entry from **Show match with**. When you type a closing (or opening) delimiter in the Command Window or Editor, MATLAB alerts you based on the option you choose:

- Balance — The corresponding delimiter is highlighted briefly.

- Underline — Both delimiters in the pair are underlined briefly.

- Highlight — Both delimiters in the pair are highlighted briefly.

Choose how you want MATLAB to alert you to mismatches using **Show mismatch with**. When you type a closing delimiter that does not have an opening match, MATLAB alerts you based on the option you choose:

- `Beep` — MATLAB beeps.
- `Strikethrough` — The delimiter you typed is briefly crossed out.
- `None` — There is no action.

Match on arrow key. Select the check box if you want to be alerted to matches and mismatches in pairs of delimiters when you use an arrow key to move the cursor over a delimiter. Then choose how you want MATLAB to alert you to matches by selecting an entry from **Show match with**. When you move the arrow over a closing (or opening) delimiter in the Command Window or Editor, MATLAB alerts you based on the option you choose:

- `Underline` — Both delimiters in the pair are underlined briefly.
- `Highlight` — Both delimiters in the pair are highlighted briefly.

Choose how you want MATLAB to alert you to mismatches by selecting an entry from **Show mismatch with**. When you move an arrow key over a delimiter that does not have a match, MATLAB alerts you based on the option you choose:

- `Beep` — MATLAB beeps.
- `Strikethrough` — The delimiter is briefly crossed out.
- `None` — There is no alert.

Setting Fonts Preferences for Desktop Tools

You can specify your preferences for fonts that the desktop tools use. The first time MATLAB needs to use or display the list of available fonts, it gets the operating system's font list. If a font exists, but MATLAB cannot display it, then MATLAB excludes it from its list. The system fonts are installed in one of the following locations:

- The operating system's standard location

  Ask your system administrator where this is on your system.

- The /jre/lib/fonts directory where Java software is installed on your system.

See the following sections for details on setting fonts preferences:

- "Desktop Fonts Preferences" on page 2-122
- "Custom Fonts Preferences" on page 2-127
- "Changing the Font — Example" on page 2-129
- "Antialiasing for Desktop Fonts on Linux and UNIX Platforms" on page 2-131
- "Making Fonts Available to MATLAB Tools" on page 2-131

Desktop Fonts Preferences

Use desktop font preferences to specify the font characteristics for MATLAB desktop tools. The font characteristics are

- Name (also called family or type), for example, select SansSerif
- Style, for example, select bold
- Size in points, for example, type 11 points

Select **File > Preferences > Fonts** to set fonts for desktop tools. You can specify the font to be used by all tools that primarily display code such as the Command Window, and specify the font to be used by other desktop tools that display text. The font to be used for HTML Proportional text, which is used in the Help display pane and the MATLAB Web browser, is a custom font that you can modify. If you want, you can separately specify the font for each desktop tool.

Select the font characteristics from the lists shown. For font size, not all entries are shown. You can type a size, including one not shown.

You can set some font options differently for printing — see "Printing and Page Setup Options for Desktop Tools" on page 2-89.

For information about making additional fonts available to MATLAB, see "Making Fonts Available to MATLAB Tools" on page 2-131.

With the code and text font styles, you can easily apply the same font to all desktop tools that display code or text, respectively.

Type a font size if it is not in the list

Preferences

Fonts Preferences

- ⊞ General
- ⊞ Keyboard
- ⊟ Fonts
  - Custom
  - Colors
  - M-Lint
  - Toolbars
  - Command Window
  - Command History
- ⊞ Editor/Debugger
  - Help
  - Web
  - Current Directory
  - Variable Editor
  - Workspace
  - GUIDE
  - Time Series Tools
- ⊞ Figure Copy Template

Desktop code font

Currently used by: Command Window, Command History, Editor

Monospaced ▼ Plain ▼ 10 ▼

Sample

The quick brown fox jumps over the lazy dog.
1234567890

Desktop text font

Currently used by: Help Navigator, Current Directory, Workspace, Variable Editor

☑ Use system font

Tahoma ▼ Plain ▼ 8 ▼

Sample

The quick brown fox jumps over the lazy dog. 1234567890

Custom fonts

Currently used by: HTML Proportional Text

To set a custom font for any desktop tool, go to the Custom Fonts preferences.

OK   Cancel   Apply   Help

Use Custom fonts to apply a font to the Help display pane
and MATLAB Web browser.

Desktop Code Font and Desktop Text Font. You specify separate font characteristics for tools that primarily display code (**Desktop code font**), such as the Command Window, and tools that primarily display text (**Desktop text font**), such as the Current Directory browser. (For other tools, such as the Help display pane and the MATLAB Web Browser you use custom fonts—see "Custom Fonts Preferences" on page 2-127.)

Many users prefer that code display in a monospace font to provide better alignment, and to distinguish it from other text information. With the desktop code font preference, you set just one preference to apply a monospace style to all tools that display code (except the Help and Web Browsers).

Typically users specify a proportional font for tools that display little or no code. You use the desktop text font preference to set just one preference that applies to all tools that display little or no code. If you want to use the system font as the desktop text font, select **Use system font**.

The following illustrations show how the Editor looks using a monospace font compared to a proportional font. A monospace font is useful when you care about alignment, while a proportional font uses less space.

With a monospaced font, all characters are the same width. Here, the font is 10 pt. Monospace. Note the 10th character in each line aligns with the Editor's right-hand text limit, which is set to column 10.
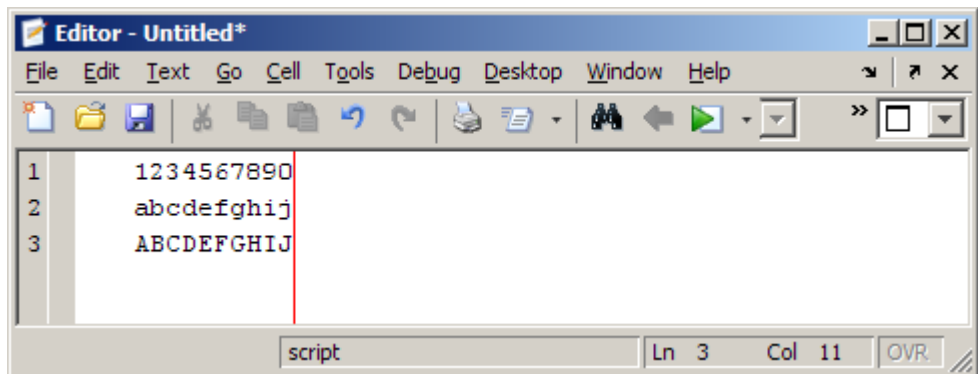
With a proportional font, characters are different widths. Here, the font is 10 pt. SanSerif. Each line contains 10 characters, but the length of each line differs. The Editor's right-hand text limit, at column 10, is not relevant.
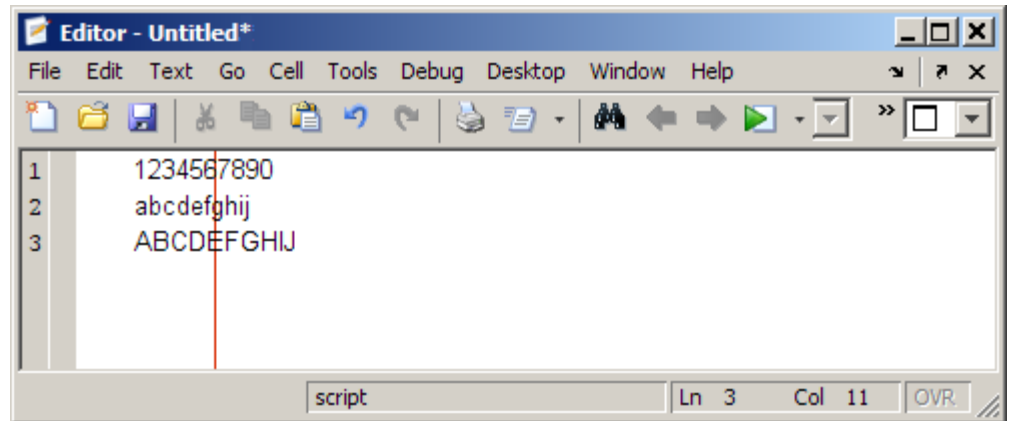
When you change a font characteristic for **Desktop code font**, the characteristic takes effect for all tools that use the desktop code font. The same is true when you change a font characteristic for **Desktop text font**.

After changing a font characteristic, a sample in the dialog box shows how it will look. Click **Apply** or **OK** to make the change take effect in the desktop tools.

Factory Default Font Settings

The following table lists the factory default code and text font settings, and the tools that use those font settings. If you have made changes but want to revert to the default font characteristics, make changes using the values in the table.

Font Type	Factory Default Characteristics and Sample	Tools Using Font Type by Factory Default
Desktop code font	Monospaced, Plain, 10 point `Sample text font`	• Command History • Command Window • Editor (which also applies to the Shortcuts Editor)
Desktop text font	Your system's current font.	• Current Directory browser (which also applies to the Path browser) • Function Browser in the Command Window and Editor • Help Navigator • Workspace browser • Variable Editor

See Also

"Specifying Options for MATLAB Using Preferences" on page 2-104

Custom Fonts Preferences

Use custom font preferences if you want to specify the font settings for tools that use the HTML Proportional Text font, including the MATLAB Help display pane and Web browser. You also can use custom font preferences if you do not want to use the current settings for "Desktop Code Font and Desktop Text Font" on page 2-125 for all other tools. Select **File > Preferences > Fonts > Custom**. The **Fonts Custom Preferences** pane appears.

In the illustration shown, the Command Window uses the **Desktop code font**, which was defined in the **Fonts** pane as described in "Desktop Fonts Preferences" on page 2-122.

Use custom fonts preferences to specify the tools that use the code style font and the tools that use the text style font. You can also apply a custom font to any tool.
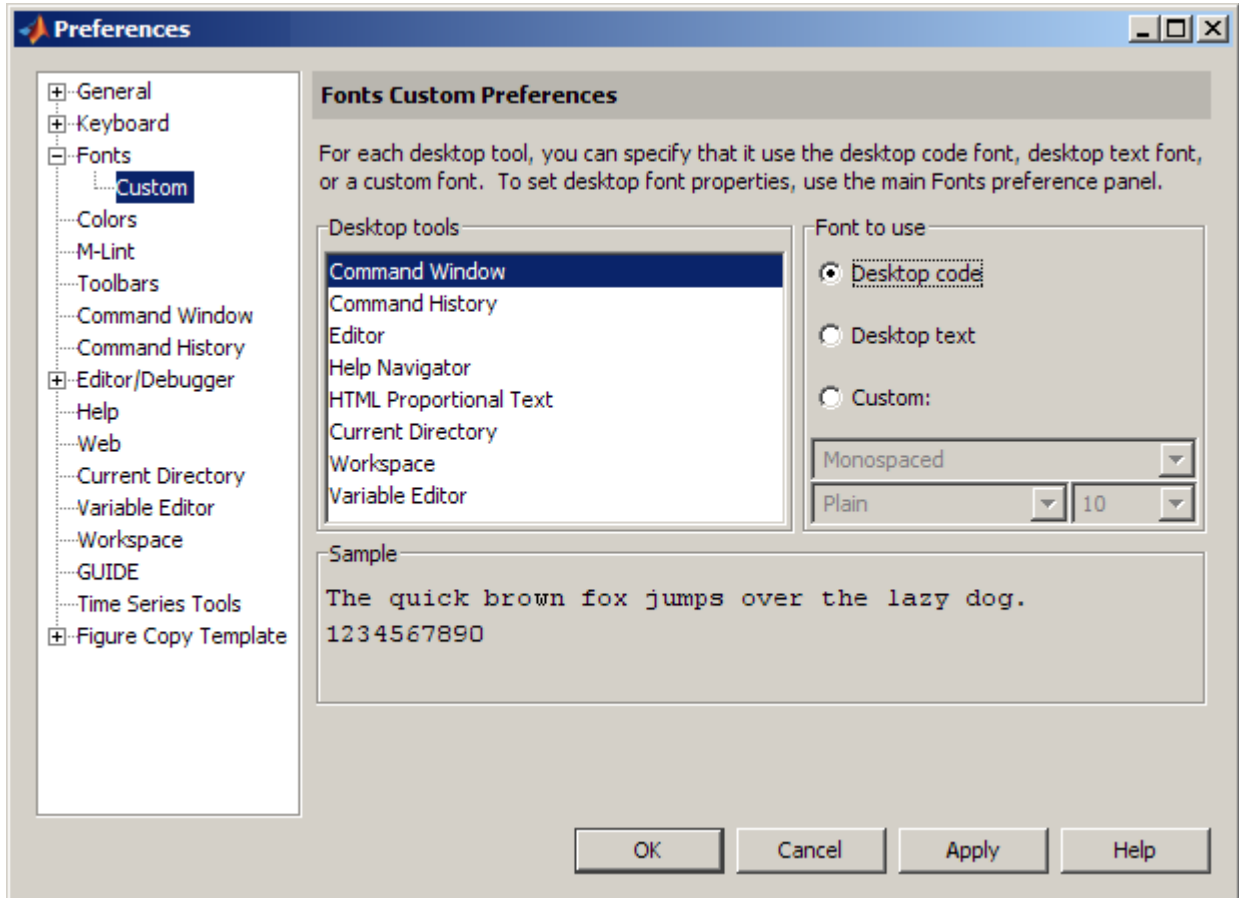
Select a tool from the **Desktop tools** list. The type of font the tool currently uses appears under **Font to Use**.

To change the type of font that a selected tool uses, for **Font to Use**, select **Desktop code** or **Desktop text**. You can instead select **Custom** and then specify the font characteristics.

Note Tools that use **HTML Proportional Text** include the MATLAB Web browser (which displays the HTML output generated from publishing), the Profiler, the Help browser display pane, and the small help window used by Help on Selection. If you change the font style (for example, to bold or italic) for HTML Proportional Text, it has no effect. If you change the font size, it affects both noncode and code text for tools using the HTML Proportional Text font.

Changing the Font — Example

This example changes the default settings (see Factory Default Font Settings on page 126) for the desktop code font, changes the Command History font preference so that it uses the desktop text font instead of the code font, and specifies a custom font for the Current Directory browser:

1 Change the characteristics for the desktop code font.

On the **Fonts Preferences** pane, set the **Desktop code font** to Times New Roman, Plain, 14 point.

2 Use the system default font for the desktop text font.

On the **Fonts Preferences** pane, under **Desktop text font** select **Use system font**.

3 Click **Apply**.

4 Make the Command History window use the desktop text font:

   a Click the **Custom Fonts** link.

   b From **Desktop tools**, select Command History.

   c Select the **Desktop text** radio button.

5 Apply a custom font to the Current Directory browser.

    a From **Desktop tools**, select `Current Directory`.

    b Select the **Custom** radio button.

    c Select `Arial Narrow` and `Plain`, and then type `11` in the size field.

    d Click **OK**.

The following table details the results of the changes.

Tool	Font Type	Font Characteristics
Command Window	Desktop code	Monotype Imaging Times New Roman® font, Plain, 14 point
Command History	Desktop text	Same as your current system font, which appears in the dimmed fields below the **Use system font** check box.
Editor	Desktop code	Times New Roman font, Plain, 14 point
Help Navigator	Desktop text	Same as your current system font, which appears in the dimmed fields below the **Use system font** check box.
HTML Proportional Text	Custom	SansSerif, Plain, 10 point
Current Directory browser	Custom	Monotype Corporation Arial® Narrow font, Plain, 11 point
Workspace browser	Desktop text	Same as your current system font, which appears in the dimmed fields below the **Use system font** check box.
Variable Editor	Desktop text	Same as your current system font, which appears in the dimmed fields below the **Use system font** check box.

Notice that on the Fonts preferences pane, the descriptive text reflects your changes. For example, under **Desktop text font**, the text reads, `Currently used by: Command History, Help Navigator, Workspace, Variable Editor`.

See Also. For information about how MATLAB stores preferences, and to get help for other preferences, see "Specifying Options for MATLAB Using Preferences" on page 2-104.

Antialiasing for Desktop Fonts on Linux and UNIX Platforms

To give the desktop a smoother appearance on Linux[8] and UNIX[9] platforms, select the antialiasing preference on the **Preference > Fonts** pane. The preference applies to all fonts.

Note The antialiasing option is not necessary on Microsoft Windows or Apple Macintosh platforms, because MATLAB follows the operating system's font settings on these platforms.

Making Fonts Available to MATLAB Tools

On Windows platforms, desktop components (such as the Command Window and Workspace browser), figure windows, and uicontrols support only Apple TrueType® and Microsoft OpenType® fonts. Some graphics objects, such as `xlabel`, `ylabel`, `title`, and `text`, also can render bitmapped fonts.

To make a new compatible font available to MATLAB, install the font by selecting **Start > Control Panel > Fonts** in the Windows desktop, and then selecting **File > Install New Font**. Restart MATLAB so that it can use the font.

Setting Colors Preferences for Desktop Tools

- "Setting Colors Used in Desktop Tools" on page 2-132

---

8. Linux is a registered trademark of Linus Torvalds.

9. UNIX is a registered trademark of The Open Group in the United States and other countries.

- "Desktop Tool Colors" on page 2-133

- "M-File Syntax Highlighting Colors" on page 2-135

- "Other Colors" on page 2-137

- "See Also" on page 2-138

Setting Colors Used in Desktop Tools

Desktop color preferences specify the colors used in MATLAB desktop tools and the colors that convey syntax highlighting. Select **File > Preferences > Colors** to set color preferences for desktop tools. You can set some color options differently for printing — see "Printing and Page Setup Options for Desktop Tools" on page 2-89.

To set colors for text and the background, clear the **Use system colors** check box and then select colors from the palettes.

Specify the color of hyperlinks in the Command Window and Help browser Index pane.
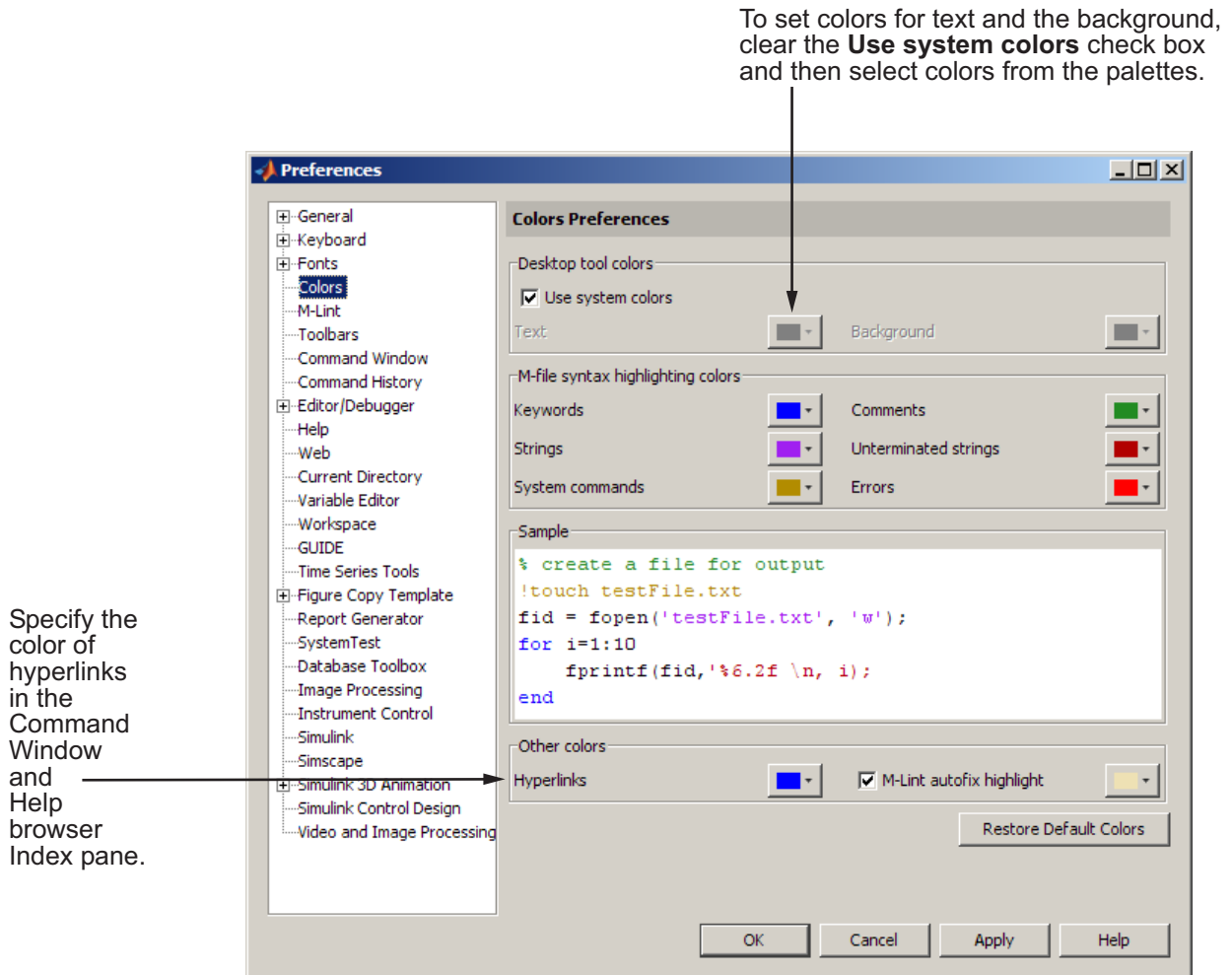
Desktop Tool Colors

Use **Desktop tool colors** to change the color of the text and background in the desktop tools. The colors also apply to the Import Wizard. The colors do not apply to the HTML display pane nor to the Web Browser.

Select the check box **Use system colors** if you want the desktop to use the same text and background colors that your platform (for example, Microsoft Windows) uses for other applications.

To specify different text and background colors, follow these steps:

1 Clear the **Use system colors** check box.

2 Click the arrow next to the **Text** color and choose a new color from the palette shown.

   When you choose a color, the **Sample** area in the dialog box updates to show you how it will look.

3 Click the arrow next to the **Background** color and choose a new color.

   If you use a gray background color, a selection in an inactive window will not be visible.

4 Click **Apply** or **OK** to see the changes in the desktop tools.

Click **Restore Default Colors** to return to the default settings for desktop tool colors, as well as for syntax highlighting colors.

The following illustration shows how the Current Directory browser looks with blue-green text and a beige background. These colors are only discernible in the online version of this documentation.

Background color   Text color
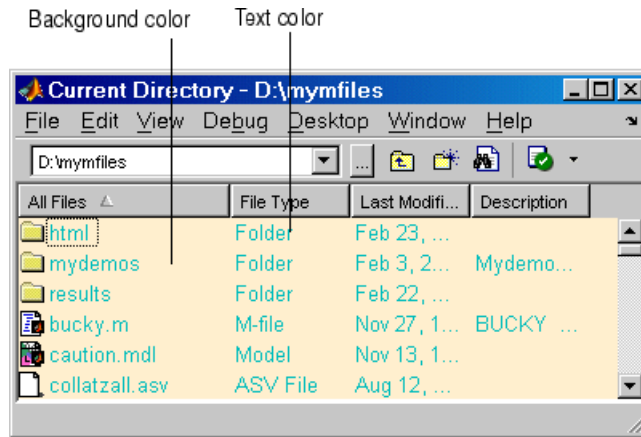
Gray Background Color. For some UNIX[10] platforms, there is a gray background color for desktop tools, such as the Editor. This occurs when the preference for **Desktop tool colors** is set to **Use system colors**, and the system's window manager uses gray as the background color default. To change the color, clear the check box for **Use system colors** and then select a new **Background** color from the palette.

M-File Syntax Highlighting Colors

In the Command Window, Command History, Editor, and Shortcuts callback area, MATLAB conveys syntax information via different colors to help you easily identify elements, such as if/else statements. This is known as syntax highlighting.

---

10.   UNIX is a registered trademark of The Open Group in the United States and other countries.

In the Command Window, only the input you type is highlighted; the output from running MATLAB functions is not highlighted.

Note To set syntax highlighting colors for files you create for the TLC, C, C++, or Sun Microsystems Java languages, or for HTML, and XML, use the Editor/Debugger language preferences by selecting **File > Preferences > Editor/Debugger > Language**. Click the **Help** button to get information on "Setting Language Preferences" on page 6-21 in the online documentation.

When you choose a color under the **M-file syntax highlighting colors** area, the **Sample** area in the dialog box updates to show you how it will look.
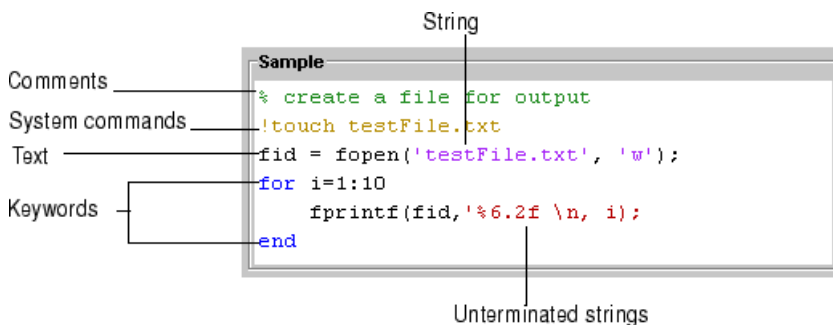
The default colors are listed here:

- **Keywords** — Flow control functions, such as for and if, as well as the continuation ellipsis (...), are colored blue.

- **Comments** — All lines beginning with a %, designating the lines as comments in MATLAB, are colored green. Similarly, the block comment symbols, %{ and %}, as well as the code in between, appear in green. Text following the continuation ellipsis on a line is also green because it is a comment.

- **Strings** — Type a string and it is colored maroon. When you complete the string with the closing quotation mark ('), it becomes purple. Note that for functions you enter using command syntax instead of function syntax, the arguments are highlighted as strings. This is to alert you that in command

notation, variables are passed as literal strings rather than as their values. For more information, see "MATLAB Command Syntax" in the MATLAB Programming Fundamentals documentation.

- **Unterminated strings** — A single quote without a matching single quote, and whatever follows the quote, are colored maroon. This might alert you to a possible error.

- **System commands** — Commands such as ! (shell escape) are colored gold.

- **Errors** — Error text that appears after you run code, including any hyperlinks, is colored red.

Click **Restore Default Colors** to return to the default settings for syntax highlighting colors and desktop tool colors.

Note Syntax highlighting for M-files is enabled by default. If you find it is disabled, follow these steps to reenable it:

1 Select **File > Preferences**, and then click **Language**.

   The Editor/Debugger Language Preferences dialog box opens.

2 In the **Language** drop-down menu, select MATLAB.

3 In the **Syntax** area, select **Enable syntax highlighting**.

   Note that from this area, you can access the Colors Preferences dialog box, by clicking **Set syntax colors**.

4 Click **Apply**.

Other Colors

Specify the color for **Hyperlinks**, which applies to links in the Command Window and Help browser **Index** pane. If you use a dark background color for those tools, be sure to use a light or other contrasting color for hyperlinks so that you can see them.

With the **M-Lint autofix highlight** preference selected, code that M-Lint can automatically correct is highlighted in the Editor. Use the palette to change the highlight color. For more information, see "Using M-Lint Automatic Code Analyzer in the Editor" on page 6-124.

See Also

For information about other preferences and how MATLAB stores preferences, see "Specifying Options for MATLAB Using Preferences" on page 2-104.

Setting Toolbars Preferences for Desktop Tools

You can customize some toolbars in the MATLAB application using Toolbars preferences. You can add and remove buttons and other controls, as well as change their position on the toolbar.

To access Toolbars preferences, select **File > Preferences** and select **Toolbars**.

The following figure summarizes how to modify toolbars.

1. Select the toolbar you want to customize.

3a. To move a control, first select it.

3b. Then select a move button to reposition it.
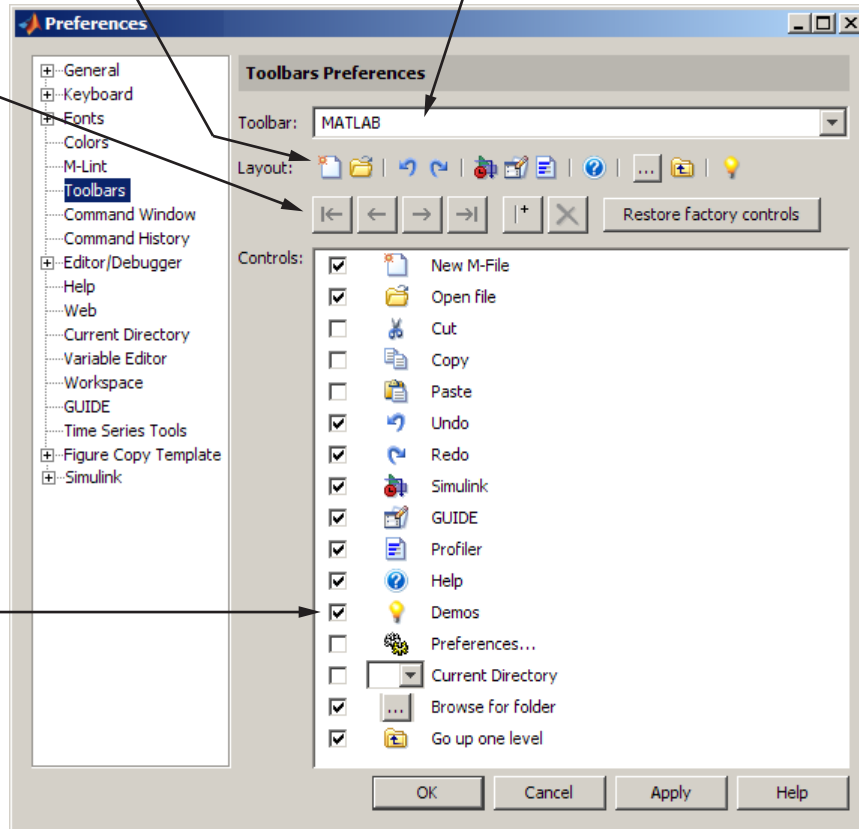
2. Select the controls you want to have on the toolbar.

Following is an example of a customized MATLAB desktop toolbar.

Example of MATLAB desktop toolbar customized using preferences.

To customize a toolbar, follow these steps:

1 Select **File > Preferences > Toolbars**. You also can access Toolbars Preferences by right-clicking a toolbar and selecting **Customize** from the context menu.

2 In the resulting Toolbars preferences pane, choose the toolbar to modify by selecting it from the **Toolbar** list:

- MATLAB — the toolbar in the MATLAB desktop
- Editor — the toolbar in the MATLAB Editor
- Editor Cell Mode — a specialized toolbar in the Editor. For more information, see "Using Cells for Rapid Code Iteration and Publishing Results" on page 6-185.
- Workspace, the toolbar in the Workspace browser
- Current Directory, the toolbar in the Current Directory browser

The controls for the selected toolbar appear in the **Layout** and **Controls** sections of the Toolbars Preferences pane.

3 Customizing controls:

- To add controls — choose a control to add to the toolbar by selecting its check box in the **Controls** section of the dialog box. For example, select the Demos check box to add its button to the toolbar.
- To remove controls — choose a control to remove from the selected toolbar by clearing its check box in the **Controls** section of the dialog box. For example, to remove the Cut, Copy, and Paste toolbar buttons, clear the check box for each.
- Restoring defaults — if you want to show the controls that appeared on the selected toolbar and in the same order as when MATLAB was first installed, click **Restore Factory Controls**.

The **Layout** section of the dialog box displays the controls you chose for the toolbar, in the order they will appear.

4 Rearrange the order of the controls and separator bars on the selected toolbar using the **Layout** area:

- Drag a control or separator bar to another position.

- Select a control or separator bar; then use one of the move buttons. For example, select the Demos button 💡, and then the Move to the End button ➡️. The Demos button moves to the right end.

- Add a separator bar after the selected control using this button: ⊞. To remove a separator bar, select it, and then use the Remove button ✖. You can use the Remove button to delete any control selected in the **Layout** section of the dialog box.

The **Layout** area displays the controls in the order you specified.

5 Click **Apply** or **OK**. The toolbars in the desktop and Editor update to reflect the changes you made.

For information about hiding, showing, and moving toolbars, see "Using Toolbar Features" on page 2-83.

Accessibility

In this section...
"Software Accessibility Support" on page 2-142
"Documentation Accessibility Support" on page 2-143
"Assistive Technologies" on page 2-144
"Installation Notes for Accessibility Support" on page 2-145
"Troubleshooting" on page 2-148

Software Accessibility Support

MathWorks products includes a number of modifications to make them more accessible to all users. Software accessibility support for blind and visually impaired users includes:

- Support for screen readers and screen magnifiers, as described in "Assistive Technologies" on page 2-144

- Command-line alternatives for most graphical user interface (GUI) options

- Keyboard access to GUI components

- A clear indication of the current cursor focus

- Information available to assistive technologies about user interface elements, including the identity, operation, and state of the element

- Nonreliance on color coding as the sole means of conveying information about working with a GUI

- Noninterference with user-selected contrast and color selections and other individual display attributes, as well as noninterference for other operating system-level accessibility features

- Consistent meaning for bitmapped images used in GUIs

- HTML documentation that is accessible to screen readers

Keyboard access to the user interface includes support for "sticky keys," which allow you to press key combinations (such as **Ctrl+C**) sequentially rather than simultaneously.

Except for scopes and real-time data acquisition, the MathWorks software does not use flashing or blinking text, objects, or other elements having a flash or blink frequency greater than 2 Hz and lower than 55 Hz.

The MathWorks believes that its products do not rely on auditory cues as the sole means of conveying information about working with a GUI. However, if you do encounter any issues in this regard, please report them to the MathWorks Technical Support group.

```
http://www.mathworks.com/contact_TS.html
```

Documentation Accessibility Support

Documentation is available in HTML format for all MathWorks products in this release.

Accessing the Documentation

To access the documentation with a screen reader, go to the documentation area on the MathWorks Web site at

```
http://www.mathworks.com/access/helpdesk/help/helpdesk.html
```

Navigating the Documentation

Note that the first page that opens lists the products. To get the documentation for a specific product, click the link for that product.

The table of contents is in a separate frame. You can use a document's table of contents to navigate through the sections of that document.

Because you will be using a general Web browser, you will not be able to use the search feature included in the MATLAB Help browser. You will have access to an index for the specific document you are using. The cross-product index of the MATLAB Help browser is not available when you are using a general Web browser.

Products

The documentation for all products is in HTML and can be read with a screen reader. However, for most products, most equations and most graphics are not accessible.

The following product documentation has been modified (as described below) to enhance its accessibility for people using a screen reader such as the JAWS® application software from Freedom Scientific BLV Group:

- MATLAB (many sections, but not the function reference pages (however, M-file help is accessible))

- Spreadsheet Link™ EX

- Optimization Toolbox™

- Signal Processing Toolbox™

- Statistics Toolbox™

Documentation Modifications

Modifications to the documentation include the following:

- Describing illustrations in text (either directly or via links)

- Providing text to describe the content of tables (as necessary)

- Restructuring information in tables to be easily understood when a screen reader is used

- Providing text links in addition to any image mapped links

Equations

Equations that are integrated in paragraphs are generally explained in words. However, most complex equations that are represented as graphics are not currently explained with alternative text.

Assistive Technologies

Note To take advantage of accessibility support features, you must use MathWorks products on a Microsoft Windows platform.

Tested Assistive Technologies

The MathWorks has tested the following assistive technologies:

- The JAWS screen reader application software 5.0, 6.0, and 7.0 for Windows platforms from Freedom Scientific

- Built-in accessibility aids from Microsoft, including the Magnifier and "sticky keys"

Use of Other Assistive Technologies

Although The MathWorks has not tested other assistive technologies, such as other screen readers or ZoomText® Xtra screen magnifier from Ai Squared, The MathWorks believes that most of the accessibility support built into its products should work with most assistive technologies that are generally similar to the ones tested.

If you use other assistive technologies than the ones tested, The MathWorks is very interested in hearing from you about your experiences.

Installation Notes for Accessibility Support

Note If you are not using a screen reader such as the JAWS application software , you can skip this section.

This section describes the installation process for setting up your MATLAB environment to work effectively with the JAWS software.

Use the regular MATLAB installation script to install the products for which you are licensed. The installation script has been modified to improve its accessibility for all users.

Note Java Access Bridge 2.0 software from Sun Microsystems is installed automatically when you install MATLAB.

After you complete the product installation, there are some additional steps you need to perform to ensure the JAWS software works effectively with MathWorks products.

Setting Up JAWS Software

Make sure that the JAWS application software is installed on your machine. If it is, there is probably a shortcut to it on the Windows desktop.

Setting up JAWS software involves these tasks:

1 Add the Access Bridge to your Windows path (for networked installations only).

2 Create the `accessibility.properties` file.

These tasks are described in more detail below.

(For Networked Installations Only) Add Java Access Bridge Software to Your Path. If you are running MATLAB in a networked installation environment (that is, if the MATLAB Installer was not run on your machine), you need to take the following steps to add Java Access Bridge to your Windows path.

Note This procedure assumes the **Start** button in your Windows preferences is set to Classic mode. To set Classic mode, from the **Start** button, select **Settings**. Next select **Task Bar and Menu**. Then select the **Start Menu** tab and make sure the `Classic Start Menu` option is enabled. Click **OK** and you are done.

---

1 From the **Start** button, select **Settings**, next select **Control Panel**. Scroll down and click the **System** icon to display the System Properties dialog box.

2 In the **System Properties** dialog box, select the **Advanced** tab.

3 Click **Environment Variables**.

4 Under **System variables**, select the `Path` option.

5 Click the **Edit** button.

6 To the start of the `Path` environment variable, add the directory that contains `matlab.exe`; for example:

```
C:\matlab\bin\win32;
```

Be sure to include that semicolon between the end of this directory name and the text that was already there.

7 Click **OK** three times.

8 If the JAWS software is already running, exit and restart.

Note The JAWS software must be started with these path changes in effect to work properly with MATLAB.

Create the accessibility.properties File.

1 Create a text file that contains the following two lines:

```
screen_magnifier_present=true
assistive_technologies=com.sun.java.accessibility.AccessBridge
```

2 Use the filename `accessibility.properties`.

3 Move the `accessibility.properties` file into

matlabroot`\sys\java\jre\win32\jre1.5.0_07\lib\`

Pronunciation Dictionary for the JAWS Software. As a convenience, The MathWorks provides a pronunciation dictionary for the JAWS application software. This dictionary is in a file called `MATLAB.jdf`.

During installation, the file is copied to your system under the root directory for MATLAB at `sys\Jaws\matlab.jdf`.

To use the dictionary, you must copy it to the `\SETTINGS\ENU` folder located beneath the root installation directory for the JAWS software.

You need to restart the JAWS software and MATLAB for the settings to take effect.

Testing

After you install the JAWS software and set up your environment as described above, you should test to ensure the JAWS software is working properly:

1 Start the JAWS software.

2 Start MATLAB.

The JAWS software should start talking to you as you select menu items and work with the user interface for MATLAB in other ways.

Troubleshooting

This section identifies workarounds for some possible issues you may encounter related to accessibility support in MathWorks products.

JAWS Software Does Not Detect When Installation of the MATLAB Software Has Started

When you select `setup.exe`, the Windows copying dialog box opens and you are informed. After the files have been copied, the installation splash screen opens, and then the installer starts. However, the JAWS software does not inform you that the installer has begun: the installer either starts up below other windows or applications or it is minimized. Since the installer is not an active item, nothing is read.

Therefore, check the Windows applications bar for the installer. After you go to the installer, you can use the JAWS software to perform the installation.

JAWS Software Stops Speaking

When many desktop components are open, the JAWS software sometimes stops speaking for MATLAB.

If this happens, close most of the desktop components, exit MATLAB, and restart.

Command Output Not Read

In the MATLAB Command Window, the JAWS software does not automatically read the results of commands.

To read command output, first select **File** > **Preferences** > **Command Window**, select the option **Use arrow keys for navigation instead of command history recall**, and click **OK**. Then, in the Command Window, press the arrow keys to move to the command output and use keystrokes for the JAWS software to read the output.

With this preference set, you cannot use arrow keys to recall previous commands. Instead use the following key bindings:

- Key bindings for Windows platforms:
  - Previous history: **Ctrl+up arrow**
  - Next history: **Ctrl+down arrow**
- Emacs key bindings:
  - Previous history: **Ctrl+p**
  - Next history: **Ctrl+n**

To return to using the up and down arrow keys to recall previous commands, clear the preference.

Some GUI Menus Are Treated as Check Boxes

For some GUIs (for example, the figure window), menus are treated by the JAWS software as though they are check boxes, whether or not they actually are.

You can choose a menu item for such GUIs by using accelerator keys (e.g., **Ctrl+N** to select **New Figure**), if one is associated with a menu item. You can also use mnemonics for menu navigation (e.g., **Alt+E**).

Note that check boxes that you encounter by tabbing through the elements of a GUI are handled properly.

Text Ignored in Some GUIs

For some dialog boxes, the JAWS software reads the dialog box title and any buttons, but ignores any text in the dialog box.

Also, in parts of some GUIs, such as some text-entry fields, the JAWS software ignores the label of the field. However, the JAWS software will read any text in the text box.

Running Functions — Command Window and History

If you have an active Internet connection, you can watch the Working in the Development Environment video demo and the Command History video demo for an overview of the major functionality. The Command Window is where you run (execute) MATLAB language statements, while the Command History is a log of the statements you have run.

- "The Command Window" on page 3-2

- "Running Functions and Programs, and Entering Variables" on page 3-5

- "Entering Statements in the Command Window" on page 3-14

- "Assistance While Entering Statements" on page 3-24

- "Controlling Output in the Command Window" on page 3-48

- "Finding Text in the Command Window" on page 3-53

- "Preferences for the Command Window" on page 3-60

- "Command History Window" on page 3-65

- "Preferences for Command History" on page 3-76

The Command Window

In this section...
"About the Command Window" on page 3-2
"Opening the Command Window" on page 3-2
"Command Window Prompt" on page 3-3
"Changing the Way the Command Window Looks" on page 3-4

About the Command Window

The Command Window is one of the main tools you use to enter data, run MATLAB functions and other M-files, and display results. If you have an active Internet connection, you can Working in the Development Environment video demo for an overview of the major functionality.

Opening the Command Window

When the Command Window is not open, access it by selecting **Command Window** from the **Desktop** menu. Alternatively, open the Command Window with the commandwindow function.

If you prefer a simple command-line interface without the other MATLAB desktop tools visible, select one of the following:

- **Desktop > Desktop Layout > Command Window Only**.

  The desktop contains only the Command Window, as shown in the image that appears after this list. MATLAB appears similar to how it looked in older versions.

- **Desktop > Desktop Layout > All but Command Window Minimized**.

  All tools open and are minimized in the desktop, except the Command Window, which is maximized. The Editor also remains maximized if it was open and contained a document at the time you chose the **All but Command Window Minimized** menu item.

For more information, see "Opening and Arranging Desktop Tools" on page 2-5.
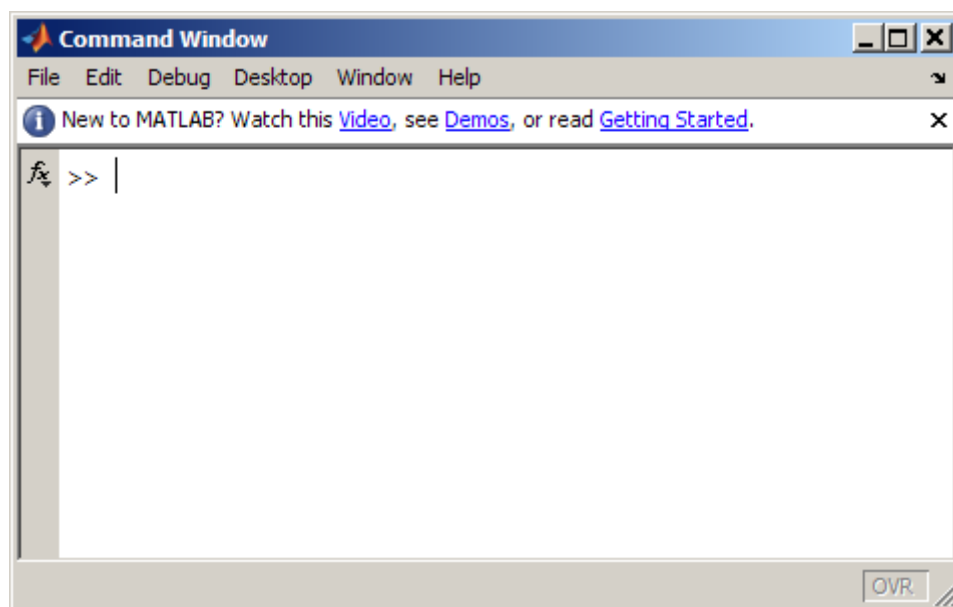
To restore the desktop to the factory default arrangement, select
Desktop > Desktop Layout > Default.

Command Window Prompt

The Command Window prompt, >>, is where you enter statements. For
example, you can enter a MATLAB function with arguments, or assign
values to variables. The prompt indicates that MATLAB is ready to accept
input from you. When you see the prompt, you can enter a variable or run a
statement. This prompt is also known as the command line.

When MATLAB displays the K>> prompt in the Command Window, MATLAB
is in debug mode. Type dbquit to return to normal mode. For more
information, see Chapter 6, "Editing and Debugging M-Files"

MATLAB displays the EDU>> prompt for the MATLAB Student Version.

Changing the Way the Command Window Looks

The Command Window is a desktop tool so you can move it, resize it, and change its font the same way you would any desktop tool. For more information, see "Opening and Arranging Desktop Tools" on page 2-5and "Setting Fonts Preferences for Desktop Tools" on page 2-121.

There are other ways you can change the display of the Command Window, including hiding the message bar and the Function Browser button. To make changes, select **File > Preferences > Command Window** and use the **Display** options. For more information, click the **Help** button in the Preferences dialog box.

Running Functions and Programs, and Entering Variables

Running Statements at the Command Line Prompt

Entering Variables and Running Functions

At the prompt, enter data and run functions. For example, to create A, a 3-by-3 matrix, type

```
A = [1 2 3; 4 5 6; 7 8 10]
```

When you press the **Enter** or **Return** key after typing the line, the MATLAB software responds with

```
A =

    1    2    3
    4    5    6
    7    8   10
```

To run a function, it must be in the current directory or in a directory on the search path. By default, functions included with MATLAB are on the search path. Therefore, you do not need to do anything special to run functions provided with The MathWorks products.

Type the function including all arguments and press **Enter** or **Return**. MATLAB displays the result. For example, type

```
magic(2)
```

and MATLAB returns

```
ans =
  1     3
  4     2
```

To determine the name of the M-file currently running, use `mfilename`.

To find the name of a function you want to run that was provided by The MathWorks, use the function browser—see "Finding Functions Using the Function Browser" on page 3-41.
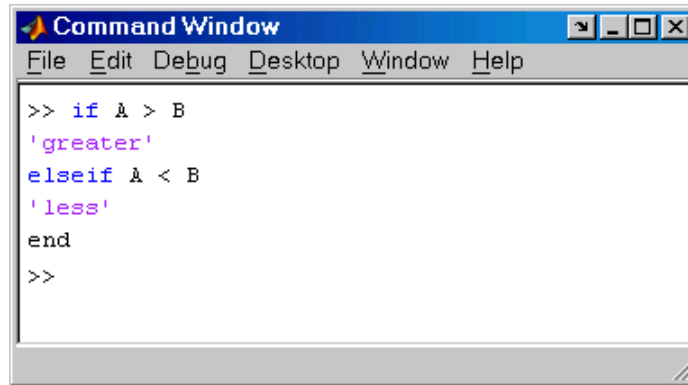
What Is a Statement?. All the information you type before pressing **Enter** or **Return** is known as a statement. Statements can include:

- Variable assignments: For example, `a = 3`

- Functions and their arguments: M-files that can accept input arguments and return output arguments, for example, `magic`.

- Commands: M-files provided with MATLAB or toolboxes that do not accept input arguments, for example, `clc`, which clears the Command Window.

- Scripts: M-files (MATLAB program files) you write that do not take input arguments or return output arguments, for example, `myfile.m`.

Some functions support a form that does not require an input argument, thereby operating as commands. For convenience, the term function is used to refer to both functions and commands.

When you enter program control statements, such as `if ... end`, the prompt does not appear until you complete the set of functions. In the following example, you press **Enter** at the end of each line, but the prompt does not appear until you complete the set of statements with `end`.

Running M-Files That Were Not Provided By The MathWorks

You can run M-files, files that contain code in the MATLAB language, that you created or that were created by other users of MATLAB. The M-file must be in the current directory or on the search path before you run it or you get an `Undefined function or variable` error or a `File not found` error. For more information, see "How To Ensure MATLAB Software Can Access Files You Want To Use" on page 5-37. When the M-file is in the current directory or on the search path, you run it the same way that you would run functions supplied with MATLAB. Type the name of the M-file in the Command Window, complete the statement by adding arguments, and press **Enter** or **Return**.

For an M-file script, you can also use the `run` function and specify the full pathname to an M-file script.

Examining Errors

If an error message appears when you run an M-file, click the underlined portion of the error message, or position the cursor within the file name and press **Ctrl+Enter**. The offending M-file opens in the Editor, scrolled to the line containing the error.

Processing Order

In MATLAB, you can only run one process at a time. If MATLAB is busy running one function, any further statements you issue are buffered in a queue. The next statement runs when the previous statement finishes.

Stopping Execution

You can stop execution of whatever is currently running by pressing **Ctrl+C** or **Ctrl+Break** at any time. On Apple Macintosh platforms, you can also use **Command+.** (the Command key and the period key) to stop the program. For certain operations, stopping the program might generate errors in the Command Window.

For M-files that run a long time, or that call built-ins or MEX-files that run a long time, **Ctrl+C** does not always effectively stop execution. Typically, this happens on Microsoft Windows platforms rather than UNIX[11] platforms. If you experience this problem, you can help MATLAB break execution by including a `drawnow`, `pause`, or `getframe` function in your M-file, for example, within a large loop. Note that **Ctrl+C** might be less responsive if you start MATLAB with the `-nodesktop` option.

Running External Programs

The exclamation point character, !, sometimes called bang, is a *shell escape* and indicates that the rest of the input line is a command to the operating system. Use it to invoke utilities or call other executable programs without quitting MATLAB. On UNIX platforms, for example,

```
!vi yearlystats.m
```

invokes the vi editor for a file named `yearlystats.m`. After the external program completes or you quit the program, the operating system returns control to MATLAB. Add `&` to the end of the line, such as

```
!dir &
```

---

11. UNIX is a registered trademark of The Open Group in the United States and other countries.

on Windows platforms to display the output in a separate window or to run the application in background mode. For example

```
!excel.exe &
```

opens Microsoft Excel software and returns control to the Command Window so you can continue running MATLAB language statements.

The maximum length of the argument list provided as input to the bang (!) command is determined by any restrictions maintained within the operating system. If you are running the Microsoft Windows XP operating system, for example, the length of the argument list input to the bang command cannot exceed 8189 characters.

See the reference pages for the `unix`, `dos`, and `system` functions for details about running external programs that return results and status.

Note To execute operating system commands with specific environment variables, include all commands to the operating system within the `system` call. Separate the commands using `&` (ampersand) for DOS, and `;` (semicolon) for UNIX platforms. This applies to the MATLAB `!` (bang), `dos`, `unix`, and `system` functions. Another approach is to set environment variables before starting MATLAB.

On Macintosh platforms, you cannot run AppleScript® (from Apple) directly from MATLAB. However, you can run the Apple Mac OS X `osascript` function from the MATLAB `unix` or `!` (bang) function to run AppleScript from MATLAB.

UNIX Platforms System Path for Running UNIX Programs from the MATLAB Software

To run a UNIX program from MATLAB if its directory is not on the UNIX system path MATLAB uses, take one of the actions described here.

Change the Current Directory in MATLAB. Change the current directory in MATLAB to the directory that contains the program you want to run.

Modify the UNIX System Path that MATLAB Uses. Add the directories to the system path from the shell. The exact steps depend on your shell. This is an example using sh:

1 At the system command prompt, type

```
export PATH="$PATH:<mydirectory>"
```

where `<mydirectory>` is the directory that contains the program you want to run.

2 Start MATLAB.

3 In the MATLAB Command Window, type

```
!echo $PATH
```

The directory containing the file is added to the system path that MATLAB uses. This change applies only to the current session of the terminal window.

Automatically Modify the System Path When the MATLAB Starts. If you want to add a directory to the `PATH` environment variable each time you start MATLAB, perform these steps:

1 In a text editor, open the file `MATLAB/bin/matlab`. This file is used to start MATLAB.

2 Add this line to the beginning of the `matlab` file

```
export PATH="$PATH:<mydirectory>"
```

where `<mydirectory>` is the directory you want to add to the path.

If you run a tsch shell instead of a bash shell, use `setenv` instead of `export`.

3 Save the file.

The `matlab` file will modify the `PATH` environment variable, and then start MATLAB.

Evaluating or Opening a Selection

Make a selection in the Command Window and press **Enter** or **Return**. The selection is appended to whatever is at the prompt, and MATLAB executes it.

Similarly, you can select a statement from any MATLAB desktop tool, right-click, and select **Evaluate Selection** from the context menu. Alternatively, after making a selection, use the shortcut key, **F9**, or for some tools, press **Enter** or **Return**. For example, you can scroll up in the Command Window, select a statement you entered previously, and then press **Enter** to run it. If you try to evaluate a selection while MATLAB is busy, for example, running an M-file, execution waits until the current operation is done.

You can open a function, file, variable, or Simulink model from the Command Window. Select the name in the Command Window, and then right-click and select **Open Selection** from the context window. This runs the `open` function for the item you selected so that it opens in the appropriate tool:

- M-files and other text files open in the Editor.
- Figure files (`.fig`) open in a figure window.
- Variables open in the Variable Editor.
- Models open in Simulink software.

See the `open` reference page for details about what action occurs if there are name conflicts. If no action exists to work with the selected item, **Open selection** calls `edit`.

Function Alternative

Use `open` or `edit` to open a file in the Editor. Use `type` to display the M-file in the Command Window.

Displaying Hyperlinks in the Command Window

You can use MATLAB functions to create hyperlinks in the Command Window. The created hyperlink can:

- Open an HTML page in a MATLAB Web browser using an `href` string.
- Transfer files via the file transfer protocol (FTP).

- Run a MATLAB M-file using the `matlabcolon` (`matlab:`) command.

Hyperlinks to Web Pages

When creating a hyperlink to a Web page, append a full hypertext string on a single line as input to the `disp` or `fprintf` command. For example, the command

```
disp('<a href = "http://www.mathworks.com">The MathWorks Web Site</a>')
```

displays the hyperlink

```
The MathWorks Web Site
```

in the Command Window.

When you click this link, a MATLAB Web browser opens and displays the requested page.

Transferring Files via FTP

To create a link to an FTP site, enter the site address as input to the `disp` command as shown below.

```
disp('<a href = "ftp://ftp.mathworks.com">The MathWorks FTP Site</a>')
```

This command displays

```
The MathWorks FTP Site
```

as a link in the Command Window.

When you click this link, a MATLAB browser opens and displays the requested FTP site.

Clicking a Hyperlink to Run MATLAB Functions

Use `matlab:` to run a specified statement when you click a hyperlink in the Command Window. For example

```
disp('<a href="matlab:magic(4)">Generate magic square</a>')
```

displays

Generate magic square

When you click the link Generate magic square, MATLAB runs magic(4). Alternatively, you can press **Ctrl+Enter** if the cursor is positioned in the link text. You can use the disp, error, fprintf, or warning function with this feature. Change the hyperlink color using **Colors Preferences** — see "Setting Colors Preferences for Desktop Tools" on page 2-131. For more information, including examples, see the matlabcolon (matlab:) reference page.

Entering Statements in the Command Window

Case and Space Sensitivity

Uppercase and Lowercase for Variables

With respect to case, the MATLAB language requires an exact match for variable names. For example, if you have a variable a, you cannot refer to that variable as A.

Uppercase and Lowercase for Files and Functions

With respect to functions, file names, objects, and classes on the search path or in the current directory, MATLAB prefers an exact match with regard to case. MATLAB runs a function if you do not enter the function name using the exact case, but displays a warning the first time you do this.

To avoid ambiguity and warning messages, always match the case exactly. It is a best practice to use lowercase only when running and naming functions.

This is especially useful when you use both Microsoft Windows and UNIX[12] platforms because their file systems behave differently with regard to case.

Note that if you use the `help` function, function names are shown in all uppercase, for example, `PLOT`, solely to distinguish them. Some functions for interfacing toSun Microsystems Java software do use mixed case and the M-file help and documentation accurately reflect that.

Examples. The directory `first` is at the top of the search path and contains the file `A.m`. If you type `a` instead of `A`, MATLAB runs `A.m` but issues a warning. When you type `a` again during that session, MATLAB runs `A.m` but does not show the warning.

Add the directory `second` after `first` on the search path, with the file `a.m` in `second`. The directory `first` contains `A.m`, while `second` contains `a.m` Type `a`. MATLAB runs `a.m` but displays a warning the first time you do this.

Spaces in Expressions

Blank spaces around operators such as `-`, `:`, and `( )`, are optional, but they can improve readability. For example, MATLAB interprets the following statements the same way.

```
y = sin (3 * pi) / 2
y=sin(3*pi)/2
```

Cut, Copy, Paste, and Undo Features

Use the **Cut**, **Copy**, **Paste**, **Undo**, and **Redo** features from the **Edit** menu when working in the Command Window. You can also access some of these features in the context menu for the Command Window.

Undo applies to some of the actions listed in **Edit** menu. You can undo multiple times in succession until there are no remaining actions to undo. Select **Edit > Redo** to reverse an undo.

Press the **Esc** (escape) key to clear everything you entered on the current line.

---

12. UNIX is a registered trademark of The Open Group in the United States and other countries.

If you use **Enter**, you cannot edit a line after entering it, even though you have not completed the flow. In that event, use **Ctrl+C** to end the flow, and then enter the statements again.

Entering Multiple Lines Without Running Them

To enter multiple statements on multiple lines before running any of them, use **Shift+Enter** or **Shift+Return** after typing a statement on a line. The cursor moves down to the next line, which does not show a prompt, where you can type the next statement. Continue for more lines. Then press **Enter** or **Return** to run all of the lines. This allows you to edit any of the statements before you pressing **Enter** or **Return**. This is an example:

```
>> a=1 % Press Shift+Enter to advance without executing this statement.
b=2    % Press Shift+Enter to advance without execution. You can edit this or the above line.
c=a+b  % Press Enter to execute all three statements.
```

MATLAB executes all three lines and returns the following:

```
a =
     1
b =
     2
c =
     3
>>
```

When you enter a paired keyword statement on multiple lines, such as for and end, you do not need to use **Shift+Enter**. You can use the typical process of pressing **Enter** after each line in the set to advance to the next line. MATLAB executes the keyword statement after complete it on the last line. For example

```
>> for r=1:5 % Press Enter. MATLAB advances a line where you continue the paired keyword statement.
a=pi*r^2     % Press Enter. MATLAB advances a line where you continue the paired keyword statement.
end          % Press Enter to execute the paired keyword statement.
```

MATLAB executes all three lines and returns the following:

```
a =
   3.141592653589793
a =
  12.566370614359172
a =
  28.274333882308138
```

Entering Multiple Functions in a Line

To enter multiple functions on a single line, separate the functions with a comma (,) or semicolon (;). Using the semicolon instead of the comma will suppress the output for the command preceding it. For example, put three functions on one line to build a table of logarithms by typing

```
format short; x = (1:10)'; logs = [x log10(x)]
```

and then press **Enter** or **Return**. The functions run in left-to-right order.

Entering Multiple-Line (Long) Statements — Line Continuation

If a statement does not fit on one line, enter three periods (...) , also called dots, stops, or an ellipsis, at the end of the line to indicate it continues on the next line. Then press **Enter** or **Return**. Continue typing the statement on the next line. You can repeat the ellipsis to add a line break after each line until you complete the statement. When you finish the statement, press **Enter** or **Return**.

For items in single quotation marks, such as strings, you must complete the string in the line on which it was started. For example, completing a string as shown here

```
headers = ['Author Last Name, Author First Name, ' ...
'Author Middle Initial']
```

results in

```
headers =
Author Last Name, Author First Name, Author Middle Initial
```

MATLAB produces an error when you do not complete the string, as shown here:

```
headers = ['Author Last Name, Author First Name, ...
Author Middle Initial']

??? headers = ['Author Last Name, Author First Name, ...
Error: Missing variable or function.
```

Note that MATLAB ignores anything appearing after the `...` on a line, and continues processing on the next line. This effectively creates a comment out of the text following the `...` on a line. For more information, see "Commenting Out Part of a Statement" on page 6-41.

Recalling Previous Lines in the Command Window

You can recall, edit, and reuse functions you typed earlier by using the arrow, tab, and control keys on your keyboard. For example, suppose you mistakenly enter

```
rho = (1+ sqt(5))/2
```

Because you misspelled `sqrt`, MATLAB responds with

```
Undefined function or variable 'sqt'.
```

Instead of retyping the entire line, press the up arrow ↑ key. The previously typed line is redisplayed. Use the left arrow key to move the cursor, add the missing `r`, and press **Enter** or **Return** to run the line. Repeated use of the up arrow key recalls earlier lines, from the current and previous sessions. Using the up arrow key, you can recall any line maintained in the Command History window.

Similarly, specify the first few characters of a line you entered previously and press the up arrow key to recall the previous line. For example, type the letters `plo` and then press the up arrow key. This displays the last line that

started with `plo`, as in the most recent `plot` function. Press the up arrow key again to display the next most recent line that began with `plo`, and so on. Then press **Enter** or **Return** to run the line. This feature is case sensitive.

If the up arrow key moves the cursor up but does not recall previous lines, clear the accessibility preference. For more information, see "Accessibility" on page 3-64.

Another way to view and access commands from the current and previous sessions of MATLAB is with the Command History window — see "Command History Window" on page 3-65.

Keyboard Shortcuts in the Command Window

Following is the list of arrow and control keys that serve as shortcuts for using the Command Window. In addition to these shortcut keys (sometimes called hot keys), you can use shortcuts for menu items, which you can view on the menus, as well as general desktop shortcuts described in "Performing Desktop Tasks Using Keyboard Shortcuts" on page 2-67. If you select the **Emacs (MATLAB standard)** preference for key bindings (see "Command Window Key Bindings Preferences" on page 2-117 for an explanation), you can also use the **Ctrl+key** combinations shown in the table.

Key or Mouse Action for Windows Preference	Control Key for MATLAB standard (Emacs) Preference	Key or Mouse Action for Macintosh Preference	Operation
↑	**Ctrl+P**	↑	Recall previous line — for details, see "Recalling Previous Lines in the Command Window" on page 3-18. See also "Command History Window" on page 3-65, which is a log of previously used functions, and "Keeping a Session Log" on page 3-51.
			With the **Accessibility** preference selected, moves the cursor up a line when it is above the prompt. In that event, use **Ctrl+↑** to recall previous lines for key

Key or Mouse Action for Windows Preference	Control Key for MATLAB standard (Emacs) Preference	Key or Mouse Action for Macintosh Preference	Operation
			bindings for Microsoft Windows and Apple Macintosh platforms.
↓	**Ctrl+N**	↓	Recall next line — for details, see "Recalling Previous Lines in the Command Window" on page 3-18. Works only after using the up arrow or **Ctrl+P**.
			With the **Accessibility** preference selected, moves the cursor down a line when it is above the prompt. In that event, use **Ctrl+↓** to recall previous lines for key bindings for Windows and Macintosh platforms.
Ctrl+Home	None	**Home**	Move to top of Command Window.
Ctrl+End	None	**End**	Move to end of Command Window.
None	None	**Cmd+Home**	Move cursor and scroll to top of Command Window.
None	None	**Cmd+End**	Move cursor and scroll to end of Command Window.
None	None	**Shift+Cmd+Home**	Select to top of Command Window.
None	None	**Shift+Cmd+End**	Select to end of Command Window.
←	**Ctrl+B**	←	Move back one character.
→	**Ctrl+F**	→	Move forward one character.
Ctrl+←	None	**Option+←**	Move left one word.
Ctrl+→	None	**Option+→**	Move right one word.

Key or Mouse Action for Windows Preference	Control Key for MATLAB standard (Emacs) Preference	Key or Mouse Action for Macintosh Preference	Operation
Home	**Ctrl+A**	**Cmd+ ⬅**	Move to beginning of current statement. With key bindings for Macintosh platforms, move to beginning of current line.
End	**Ctrl+E**	**Cmd+ ➡**	Move to end of current statement. With key bindings for Macintosh platforms, move to end of current line.
Esc (escape)	**Ctrl+U**	**Esc**	Clear the command line when cursor is at the command line. Otherwise, move cursor to command line. If the function hints pop-up window is open, closes the window. In the Function Browser, press **Esc** up to three times: first to dismiss the search history, then to clear the search field, and lastly, to close the Function Browser.
Delete	**Ctrl+D**	Forward **Delete**	Delete character after cursor.
Backspace	**Ctrl+H**	**Delete**	Delete character before cursor.
None	**Ctrl+K**	None	Cut contents (kill) from cursor to end of current line.
Insert	None	None	Change to overwrite mode from insert mode, or change to insert mode from overwrite mode. View current mode in the status bar: OVR is gray for insert mode. In overwrite mode, what you type replaces existing text and the cursor is a wide block. (Not supported on Macintosh platforms.)

Key or Mouse Action for Windows Preference	Control Key for MATLAB standard (Emacs) Preference	Key or Mouse Action for Macintosh Preference	Operation
Double-click	None	Double-click	Select current word. To select additional words, hold mouse after second click and continue dragging left or right.
None	None	**Shift+Option+ ←**	Select to previous word.
None	None	**Shift+Option+ →**	Select to next word.
Triple-click	None	None	Select current line. To select additional lines, hold mouse after second click and continue dragging up or down.
Shift+Home	None	**Shift+Cmd+ ←**	Select from cursor to beginning of statement. With key bindings for Macintosh platforms, select to beginning of line.
Shift+End	None	**Shift+Cmd+ →**	Select from cursor to end of statement. With key bindings for Macintosh platforms, select to end of line.
Enter in selection	None	None	Append selection to statement at command line and execute it.
Ctrl+Enter in hyperlink	None	**Ctrl+Enter** in hyperlink	Open hyperlink displayed in Command Window. For example, in the hyperlink of an error message, opens the file in the Editor at that line number.

Navigating Above the Command Line

To look at or copy information in the Command Window that is above the command line (>> prompt), use the mouse and scroll bar, key combinations such as **Ctrl+Home**, and search features. By default, the up and down arrow keys recall statements, and so by default, you cannot use the keys to move the cursor when it is above the command line.

To use the up and down arrow keys to move the cursor when it is above the command line, select **File > Preferences > Command Window**, and select the **Accessibility** preference.

See Also

- "Assistance While Entering Statements" on page 3-24
- "Finding Text in the Command Window" on page 3-53

Assistance While Entering Statements

In this section...
"Highlighting Syntax to Help Ensure Correct Entries" on page 3-24
"Matching Delimiters (Parentheses)" on page 3-25
"Completing Statements in the Command Window — Tab Completion" on page 3-25
"Viewing Function Syntax While Entering a Statement — Function Hints" on page 3-34
"Getting Help for the Selected Function in the Command Window or Editor" on page 3-39
"Finding Functions Using the Function Browser" on page 3-41
"See Also" on page 3-47

Highlighting Syntax to Help Ensure Correct Entries
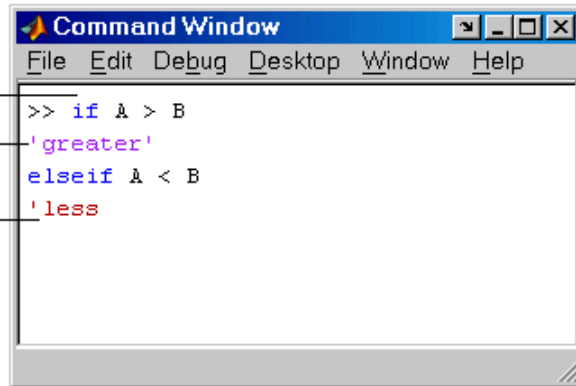
To help you better find elements, such as matching if/else statements, some entries appear in different colors in the Command Window. This is known as syntax highlighting. You can change the colors using preferences. Note that output in the Command Window does *not* appear with syntax highlighting, except for errors. Syntax highlighting is also available in other desktop tools. For more information, see "Setting Colors Preferences for Desktop Tools" on page 2-131.

Default colors are shown here—to change them, use **Preferences -> Colors**.

Keywords, like these for program control, are blue.

Closed strings are purple.

Unclosed strings are maroon.

```
>> if A > B
'greater'
elseif A < B
'less
```

Matching Delimiters (Parentheses)

You can instruct the MATLAB software to notify you about matched and unmatched delimiters. For example, when you type a parenthesis, bracket, or brace, MATLAB highlights the matched delimiter in the pair. To use the delimiter matching feature, select **File > Preferences > Keyboard > Delimiter Matching**. This feature is also available in the Editor.

For more information, see "Delimiter Matching Preferences" on page 2-119.

Completing Statements in the Command Window — Tab Completion

MATLAB helps you complete the names of known items as you type them in the Command Window so that you can avoid spelling mistakes and do not have to spend time looking up the information in other tools. These are the items for which MATLAB can complete the names:

- Functions or models on the search path or in the current directory

- MATLAB objects

- File names and directories, including object-oriented programming package and class directories

- Variables, including structures, in the current workspace
- Handle Graphics property names

Type the first few characters of the item name and then press the **Tab** key. To use tab completion, you must have the tab completion preference for the Command Window selected. For details, see "Setting Keyboard Preferences for Desktop Tools" on page 2-116.

Tab completion is also available in the Editor, but there are some slight differences in usage. See "Completing Statements in the Editor — Tab Completion" on page 6-43.

These examples demonstrate how to use tab completion in the Command Window:

- "Basic Example — Unique Completion" on page 3-26
- "Multiple Possible Completions" on page 3-27
- "Tab Completion for Directories and File names" on page 3-29
- "Tab Completion for Class Directories and File Names" on page 3-30
- "Tab Completion for Structures" on page 3-32
- "Tab Completion for Handle Graphics Properties" on page 3-33
- "Tab Completion for MATLAB Objects" on page 3-33

Basic Example — Unique Completion

This example illustrates a basic use for tab completion. After creating a variable, costs_march, type

```
costs
```

and press **Tab**. MATLAB automatically completes the name of the variable, displaying

```
costs_march
```

Then complete the statement, adding any arguments, operators, or options, and press **Return** or **Enter** to run it. In this example, if you just press

Enter, MATLAB displays the contents of `costs_march`. If MATLAB does not complete the name `costs_march` but instead moves the cursor to the right, you do not have the preference set for tab completion. If MATLAB displays `No Completions Found`, `costs_march` does not exist in the current workspace.

You can use tab completion anywhere in the line, not just at the beginning. For example, if you type

```
a = cost
```

and press **Tab**, MATLAB completes `costs_march`. You can also select `co` or position the cursor after `co` and press **Tab** to complete `costs_march`.

Multiple Possible Completions

If there is more than one name that starts with the characters you typed when you press the **Tab** key, MATLAB displays a list of all names that start with those characters. For example, type

```
cos
```

and press **Tab**. MATLAB displays
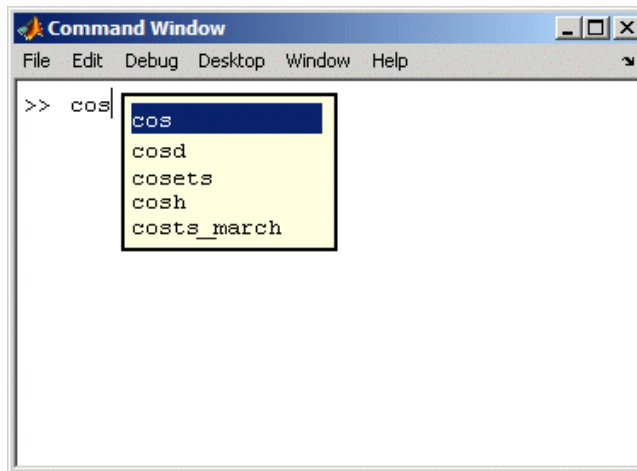
The resulting list of possible completions includes the variable name you created, `costs_march`, but also includes functions that begin with `cos`,

including `cosets` from the Communications Toolbox™ software, if it is installed on the system and on the search path in MATLAB. MATLAB completes variable names in the currently selected workspace, and the names of functions and models on the search path or in the current directory.

Continue typing to make your entry unique. For example, type the next character, such as `t` in the example. MATLAB selects the first item in the list that matches what you typed, in this case, `costs_march`. Press **Enter** (or **Return**) or **Tab** to select that item, which completes the name at the prompt. In the example, MATLAB displays `costs_march` at the prompt. Add any arguments, and press **Enter** again to run the statement.

You can navigate the list of possible completions using up and down arrow keys, and **Page Up** and **Page Down** keys. You can clear the list without selecting anything by pressing **Esc** (escape key). Note that the list of possible completions might include items that are not valid commands, such as private functions.

Narrowing Completions Shown. You can narrow the list of completions shown by typing a character and then pressing **Tab** if the Command Window preference **Tab key narrows completions** is selected. This is particularly useful for large lists. For example, type `cam` and press **Tab** to see the possible completions. There is a scroll bar with the list because there are too many completions to be seen at once.
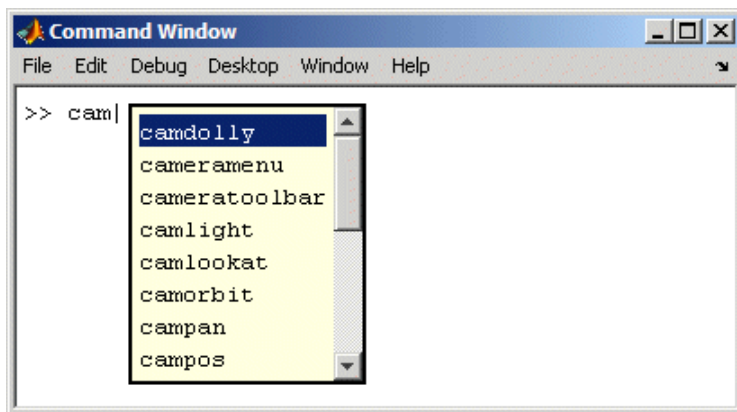
Type p and press **Tab** again. MATLAB narrows the list, showing only all possible `camp` completions.
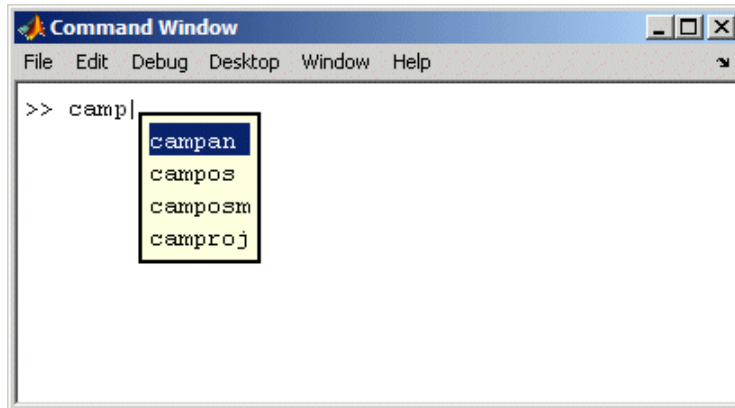
Continue narrowing the list in the same way. For the above example, type `o` and press **Tab** to further narrow the list. Press **Enter** or **Return** to select an item, which completes the name at the prompt.

Tab Completion for Directories and File names

Tab completion works for directories and file names in MATLAB functions.

For example, type

```
edit d:/
```

and press **Tab**.

MATLAB displays the list of directories and files in `d`, from which you can choose one. For example, type

```
mym
```

and press **Tab**.

MATLAB displays

```
edit d:/mymfiles/
```

where `mymfiles` is the only directory on your `d` drive whose name begins with `mym`. Continue using tab completion to display and complete directory names or file names until you finish the `edit` statement.

Tab completion for directories and file names is not supported for functions you write.

Tab Completion for Class Directories and File Names

Tab completion for class directories (including @-directories), package directories, and file names works the same as for standard directories.

For example, consider the `containers` package in the `matlabroot` directory, which is the directory in which MATLAB is installed. The `containers` package contains a `Map` @- directory and a `Map.m` file. The directory structure appears as follows:

```
+containers\@Map\Map.m
```

If you type:

```
edit cont
```

and press **Tab**, MATLAB adds characters to `cont` until MATLAB finds a character that is not common to any names on the MATLAB path. MATLAB displays a list from which to choose, such as the following

```
containers
contents.asv
Contents.m
contentsrpt.m
continue.m
contour.m
contour3.m
contourc.m
```

```
Command Window                          _ □ X
File  Edit  Debug  De                        ↘
  >>
  >>
  >>
fx >> edit cont
                                        OVR
```
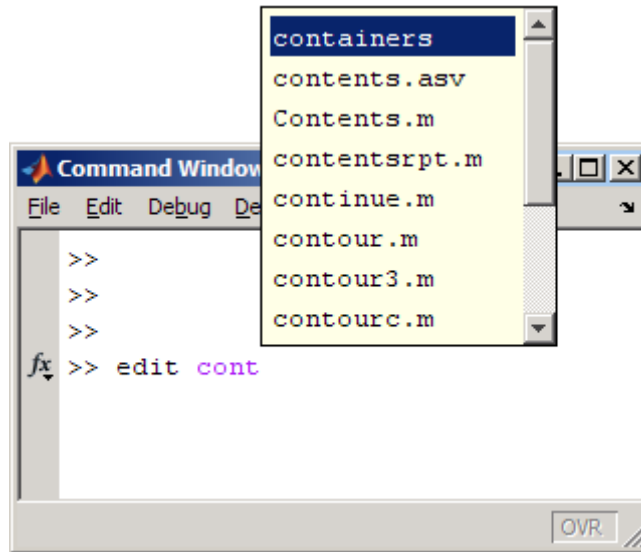
If you select `containers`, add a dot, and then press **Tab**, MATLAB displays

```
Command Window                          _ □ X
File  Edit  Debug  Desktop  Window  Help     ↘
  >>
  >>
  >>
fx >> edit containers.Map|
                                        OVR
```
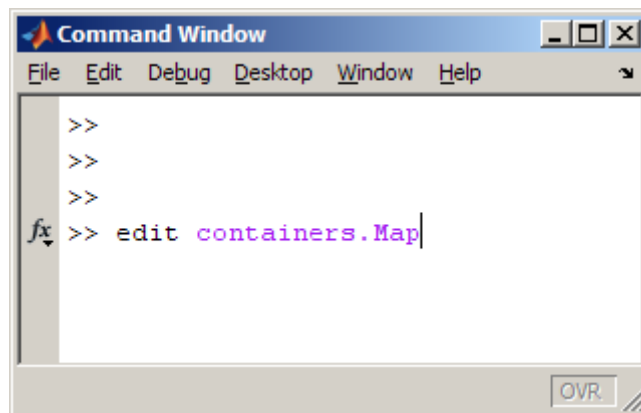
If you add a dot, press **Tab**, and scroll down a bit, MATLAB displays
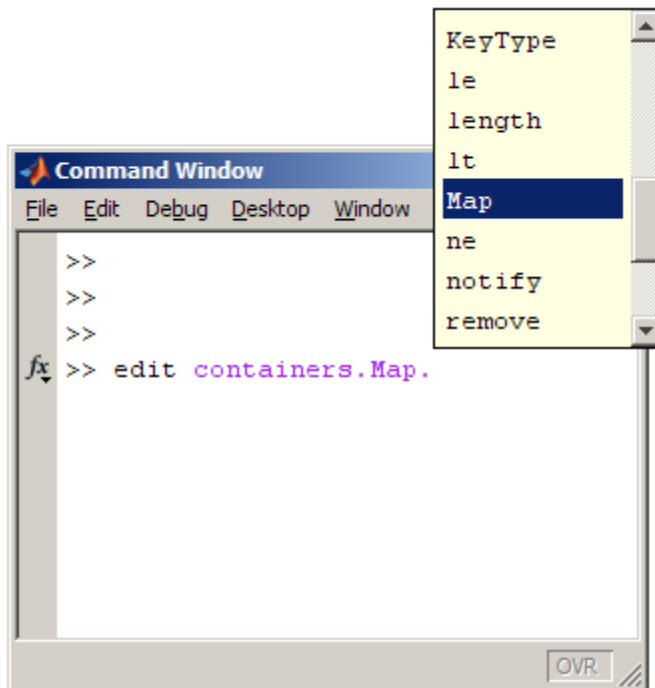
Tab Completion for Structures

For structures in the current workspace, after the period separator, press
Tab. For example, type

```
mystruct.
```

and then press **Tab** to display all fields of mystruct. If you type a structure
and include the start of a unique field after the period, pressing **Tab** completes
that structure and field entry.

For example, type

```
mystruct.n
```

and press **Tab**, which completes the entry mystruct.name, where mystruct
contains no other fields that begin with n.
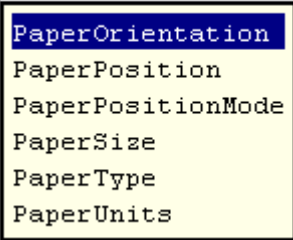
Tab Completion for Handle Graphics Properties

Complete the names of Handle Graphics properties using tab completion, as in this example. Here, f is a figure. Type

```
set(f, 'pap
```

and press **Tab**. MATLAB displays

```
set(f, 'paper
            PaperOrientation
            PaperPosition
            PaperPositionMode
            PaperSize
            PaperType
            PaperUnits
```

Select a property from the list. For example, type

```
u
```

and press **Enter**. MATLAB completes the property, including the closing quote:

```
set(f, 'paperunits'
```

Continue adding to the statement, as in this example:

```
set(f, 'paperunits', 'c
```

and press **Tab**. MATLAB automatically completes the property

```
set(f, 'paperUnits', 'centimeters'
```

because centimeters is the only possible completion.

Tab Completion for MATLAB Objects

You can use tab completion with MATLAB objects to select from available properties and methods.

For example, create a time series object, tsobj.

```
tsobj = timeseries
```

Then enter the object name followed by a period separator (.), such as

```
tsobj.
```

Press **Tab**. MATLAB displays a list of properties and methods for the object, tsobj.
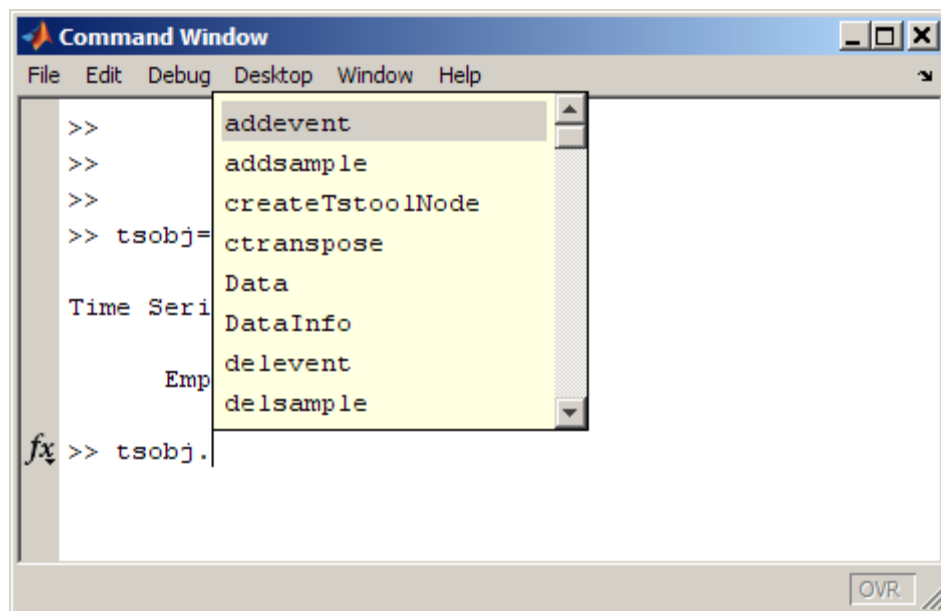
Viewing Function Syntax While Entering a Statement — Function Hints

- "What Are Function Hints and When Should You Use Them?" on page 3-35
- "How To Use Function Hints in the Command Window or Editor" on page 3-35
- "Actions You Can Perform While Using Function Hints" on page 3-38

• "Enabling or Disabling Function Hints" on page 3-38

What Are Function Hints and When Should You Use Them?

While entering a statement in the Command Window or Editor, you can view syntax for a function's input arguments in a pop-up (temporary) window. This feature is called *function hints*. Function hints are useful when you already know about a function, but want to be reminded of its syntax. You view function hints while in the context of entering the statement so you do not need to take your focus away to a help window.

If you start with function hints and then need to know more about a function than just its syntax, you can get more help while using them. Alternatively, you can choose to use the Function Browser or Help Browser instead of function hints.

To use function hints, you need to know the exact name of a function . If you do not know the exact name, use tab completion, the Function Browser or some other method to determine it.

Function hints do not display output argument syntax, but you can use function hints when entering statements that have output arguments—the hints show the input argument syntax only. If you need to know output argument syntax, you can get it from the reference page, which you can link to while using the function hints.

You can view function hints for all functions provided with MathWorks products, as well as for functions and classes you create. For MathWorks products, the syntax for the function hint comes from the function reference page. For a function or class you create, the function hint uses the first line of its M-file, if the M-file is on the search path or in the current directory.

How To Use Function Hints in the Command Window or Editor

To use functions hints, perform the following steps. The procedure uses an example for the `size` function being run in the Command Window:

1 Enter any output arguments and the equal sign, =. For example, enter

```
[m]=
```

2 Continue entering the statement—enter the function name and the opening (left) parenthesis, (, then pause. For example, enter

```
size(
```

A light yellow pop-up window appears near the statement you are entering, displaying the syntax options for the function. Arguments you can enter at that point appear in bold. For the example, two syntax options for the `size` function appear. If you are familiar with the `size` function, the syntax might be enough to remind you how to enter the arguments. For `size`, the arguments are the variable (**X**), and dimension for which you want to run `size`, which is `dim` in the syntax hints. In the pop-up window, X appears in bold and `dim` does not, indicating you can now enter the variable, but cannot yet enter the dimension.
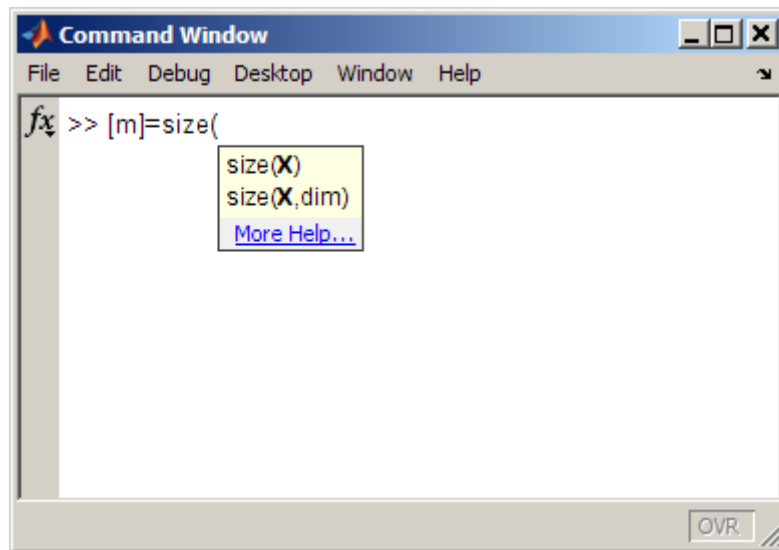
When you use function hints for an overloaded function name, the syntax for the method includes valid objects of the method currently in the workspace.

Some allowable arguments might not appear or might not be bold because MATLAB cannot always correctly determine them.

3 Enter input arguments:

a Enter the first input argument and the comma (,) after it. The syntax options shown in the pop-up window change, based on the argument you entered. For the example, my_data is the variable for which you want to run size, so enter.

```
my_data,
```

Now there is only one syntax option in the pop-up window. The dim argument is now bold while X is not, meaning you can now enter the dimension.

Do not enter the exact arguments shown in the pop-up window; they are for illustrative purposes, only. Similarly, you cannot select an entry from the pop-up window. Instead, enter your own variables that correspond to those shown.
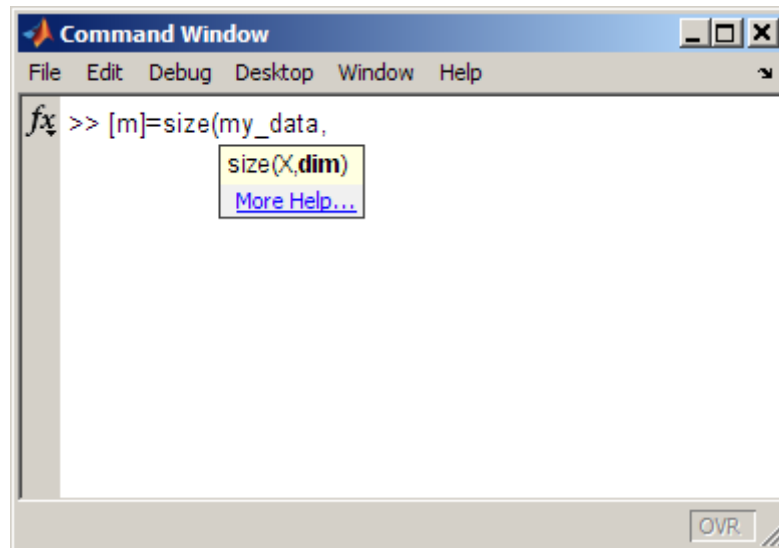
b Continue entering input arguments in the same way. For example, to run size for the third dimension, enter 3 after the comma.

c If there are no more arguments to enter, or if you enter the closing (right) parenthesis,), the pop-up window closes, completing the function

statement. For the example, after entering the 3 for dim, the pop-up
window closes.

Actions You Can Perform While Using Function Hints

You can perform these actions while using function hints:

- To get more information about the function while entering the arguments,
  including additional syntax options not shown or further description of the
  options, click the **More Help** link in the pop-up window. The full reference
  page for the function appears in a small help window, replacing the function
  hints pop-up window. The small help window and content are the same as
  for the Help on Selection feature—for more information, see "Getting Help
  for the Selected Function in the Command Window or Editor" on page 3-39.

- You can use function hints along with all other methods for getting
  assistance, including tab completion, the Function Browser, Help on
  Selection, and so on.

- To close the pop-up window, press the **Esc** (escape) key.

- After dismissing the pop-up window, you can see it again by clearing the
  arguments you entered and clearing the left parenthesis. Then enter the
  left parenthesis, (, and pause to see the hints.

- You can change the statement while the pop-up window is open. For
  example, you can use the Backspace **<–** key to clear characters you typed.

Enabling or Disabling Function Hints

To prevent function hints from appearing or to show them if they are not
appearing, use keyboard preferences.

1 Select **File > Preferences > Keyboard**.

2 In **Function hints**, select the check boxes for the tools where you want
   hints to appear, and clear the check boxes for the tools where you do not
   want hints to appear.

3 Click **OK**.

Getting Help for the Selected Function in the Command Window or Editor

While using the Command Window or the Editor, you can get full help for a function by displaying the reference page for a selected function. Follow these instructions:

1 Select a function or click the pointer within a function for which you want information.

2 Press **F1**, or right-click and select **Help on Selection** from the context menu.

The reference page for the function opens in a small help window. The following example illustration shows help on selection for the `corrcoef` function in the Command Window.
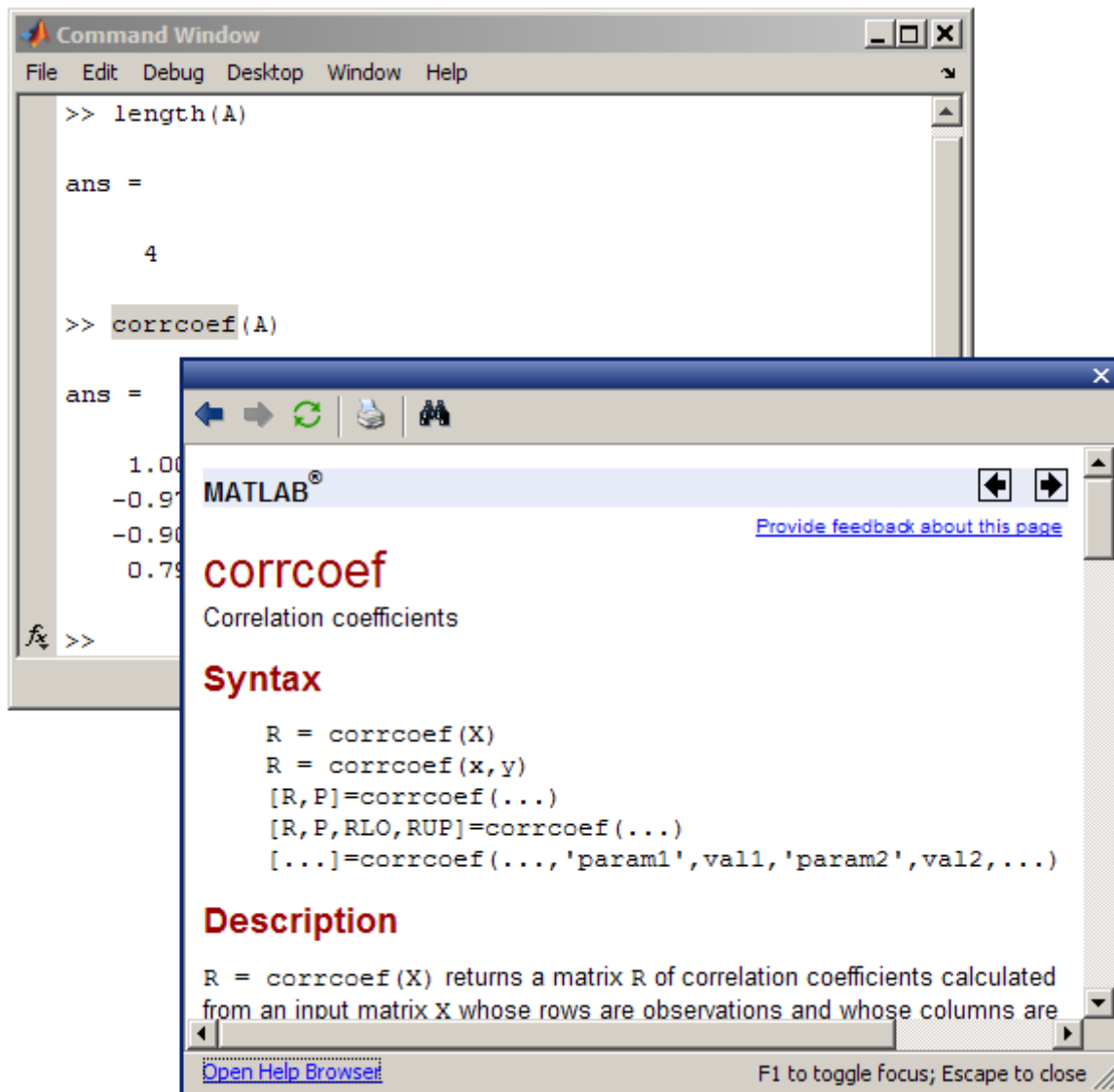
If the reference page for the function does not exist, such as for a function not provided by The MathWorks, M-file help appears instead.

3 Perform any of the following actions while using the small help window:

- Move or resize the window.

- Work in another tool, such as the Command Window or Editor; the small help window remains open.

- Toggle focus between the small help window and the Command Window or Editor by pressing the **F1** key.

- Open the function's reference page in the Help browser by clicking the **More Help** link at the bottom of the small help window; this closes the small help window.

  You can set a preference to specify that the Help on Selection option automatically opens the reference page in the Help browser instead of in the small help window. For instructions, see "Help on Selection Window — Specifying Where It Displays" on page 4-52. When Help on Selection opens in the Help browser, you can *not* toggle focus between the reference page and the Command Window or Editor.

- Display the small help window for a different function by repeating steps 1 and 2.

4 Close the small help window by clicking it to make it the active window, and then either press the **Esc** (escape) key or use the Close button.

To change the font used by the small help window, use the HTML Proportional Text Font. For more information, select **File > Preferences > Fonts > Custom**, and click the **Help** button in the Preferences dialog box.

Finding Functions Using the Function Browser

Use the Function Browser to find the names of and get help for functions while you work. The Function Browser is particularly useful in the Command Window and Editor but you can use it with other tools. The Function Browser uses the same names, syntaxes, and descriptions found in the function reference pages in the Help browser, but provides quick access to the information in context for a smoother workflow. The Function Browser does not show blockset reference pages or information for functions created by you or other users—use the Help browser for those reference pages.

1 Open the Function Browser by pressing **Shift+F1**.

When you click anywhere outside of the Function Browser, it automatically closes.

You can move the Function Browser by dragging its top edge. After you move it, you can change focus from the Command Window or Editor to the Function Browser by pressing **Shift+F1**. You can close the Function Browser by pressing **Esc** (escape).
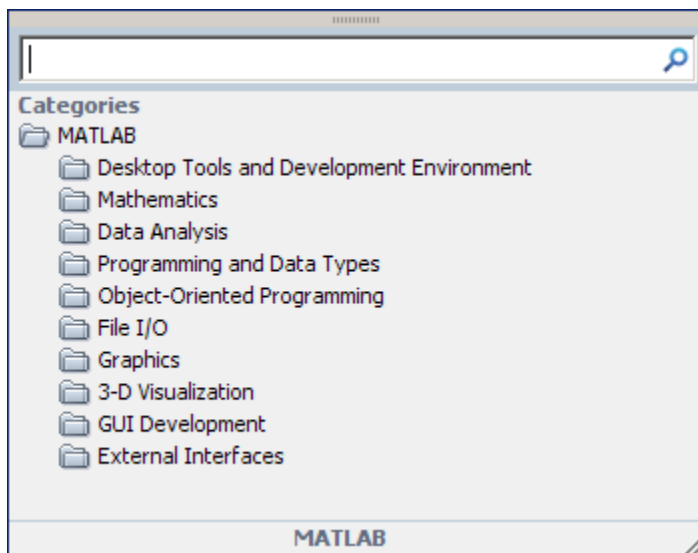
Other Ways to Open the Function Browser

- From the **Help** menu of any desktop tool.

- In the Command Window, by clicking the Function Browser button, $f_\star$, which is to the left of the prompt. To either show or hide the button, use a Command Window preference.

- In the Editor, by clicking the Function Browser button, $f_\star$, on the toolbar. To either show or hide the button, use Toolbars preferences for the Editor.

2 Specify the products that might contain the functions you are interested in by using the **Filter by Product** option in Help Preferences. You can

access the preference by clicking the product area at the bottom of the
Function Browser. You can change the product filter at any point while
using the Function Browser.

In this illustration, the filter is set so that the Function Browser looks for
functions in MATLAB and the Signal Processing Toolbox.

Click to specify
which products you
want the
Function Browser to
look in.
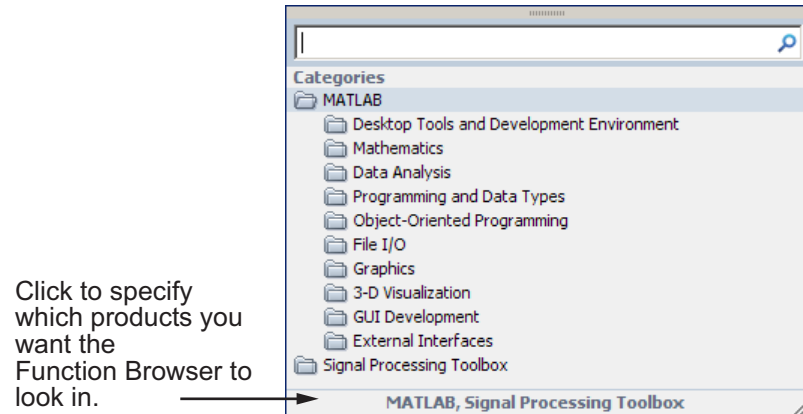
3 Find functions either by browsing the list of categories or by typing a search
term. You can switch between these two options at any point.

The following illustration shows how you browse for functions by expanding
categories of interest.

Browse categories
to find functions.
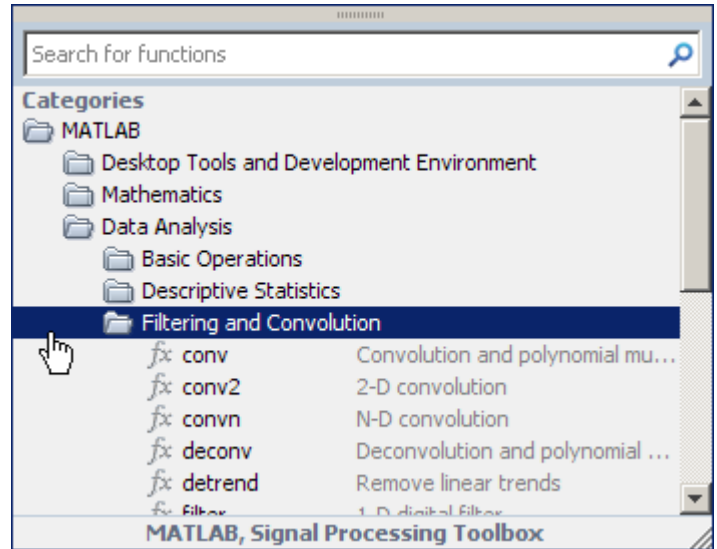
The following illustration shows results when you search for functions.

Find functions by searching,
for example, for "filter".

In the results, the product name is in parentheses.

MATLAB is assumed and is not in parentheses.
For an overloaded function, the name of the
directory containing the function is in parentheses.

filter is in MATLAB

filter (signal)is in the
Signal Processing Toolbox

filter (timeseries) is in the
timeseries directory in MATLAB.
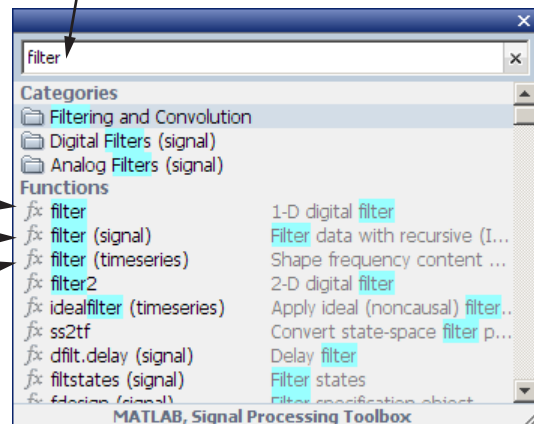
More About Searching in the Function Browser

- Type words or partial words (two or more letters) in the search field. For faster performance, when you type two letters, the Function Browser shows only functions containing an exact match.

- As you type, the Function Browser displays a history of similar search terms that you previously entered, just below the search field. You can choose an item by selecting it and pressing **Enter**. To view the entire search history for the current session, press the down arrow key when the search field is empty. To close the history, press **Esc** (escape).

- While you type, the Function Browser shows items matching your search term, highlighting the search term in blue. For results without any blue highlighting, the matching term is not part of the name, and is not in the syntax or brief description, but is somewhere else in the reference page.

  The product that contains the function appears in parentheses. No parentheses means the product is MATLAB. If there are two functions with the same name in MATLAB (overloaded functions), the directory where the overloaded function is located appears in parentheses.

- To clear the search field, press **Esc** (escape).

4 To view more information about a function, move the pointer over a function and a brief description for each of the syntax options displays in a yellow pop-up window.

The pop-up window automatically closes when you move your pointer to a new item in the results list. To keep the pop-up window open, drag its top edge to a different location.
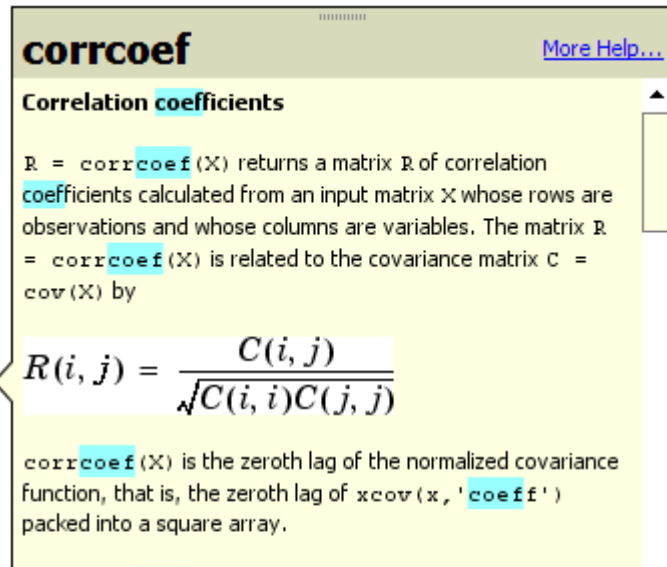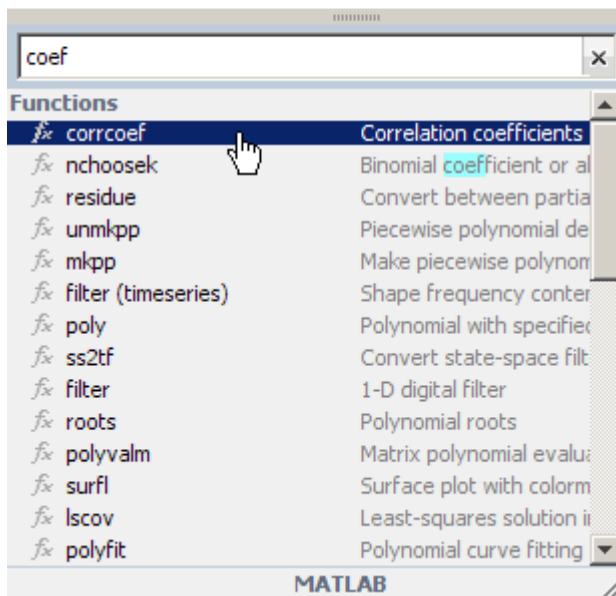
Viewing the Full Reference Page for Functions from the Function Browser

To view the complete reference page for the function in a small help window, click the **More Help** link in the pop-up window, or press **F1**. If you want the reference page to appear in the Help browser instead of the small window, set the preference in **File > Preferences > Help**.

5 After finding a function you want to use, double-click the function in the Function Browser results list to add it after the cursor in the Command Window or Editor. Alternatively, drag the function name from the Function Browser into any tool or application. For other options, right-click the function name in the Function Browser.

Function Browser — Related Topics

- To search in more than just reference pages, use the Help browser search feature—see "Searching Documentation and Demos with the Help Browser" on page 4-18.

- You cannot use the Function Browser to search for a function in a product that you do not have installed. Instead, look for help about the function in the documentation on the MathWorks Web site.

- You cannot use the Function Browser to find files that you created or that were provided by other users. Instead, see "Finding Files and Directories" on page 5-77.

- To change the font used in the Function Browser, change the preference for the desktop text font. For more information, select **File > Preferences > Fonts** and click the **Help** button in the dialog box.

See Also

- "Entering Statements in the Command Window" on page 3-14
- "M-Lint Code Check Report" on page 7-21
- Chapter 4, "Help, Demos, and Related Resources"
- The `lookfor` function

Controlling Output in the Command Window

Echoing Execution

To display each function within a statement as it executes, run `echo on`. For details, see the `echo` reference page.

Suppressing Output

If you end a statement with a semicolon (`;`) and then press **Enter** or **Return**, the MATLAB software runs the statement but does not display any output. This is particularly useful when you generate large matrices. For example, running

```
A = magic(100);
```

creates A but does not show the resulting matrix in the Command Window.

See also the `display` reference page.

Paging of Output in the Command Window

If output in the Command Window is lengthy, it might not fit within the screen and display too quickly for you to see it without scrolling back to it. To avoid that problem, use the `more` function to control the paging of output in the Command Window. By default, `more` is `off`.

After you type `more on`, MATLAB displays only a page (a screen full) of output, pauses, and displays

```
--more--
```

indicating there is more output to display. Press one of the following keys.

Key	Action
Enter or **Return**	To advance to the next line
Space Bar	To advance to the next page
q	To stop displaying the output

You can scroll in the Command Window to see input and output that are no longer in view. As an alternative to scrolling, you can use the up and down arrow keys if the Command Window Accessibility preference is selected.

Formatting and Spacing Numeric Output

By default, numeric output in the Command Window is displayed as 5-digit scaled, fixed-point values, called the short format. To change the numeric format of output for the current and future sessions, set the Command Window preference for text display. The text display format affects only how numbers are shown, not how MATLAB computes or saves them.

Function Alternative

Use the `format` function to control the output format of the numeric values displayed in the Command Window. The format you specify applies until you change it or until the end of the session. More advanced alternatives are listed in the "See Also" section of the `format` reference page.

Examples of Formats

Here are a few examples of the various formats and the output produced from the following two-element vector x.

```
x = [4/3 1.2345e-6]

format short
   1.3333    0.0000

format short e
   1.3333e+000  1.2345e-006

format +
++
```

A complete list and description of available formats is in the reference page for format. For more control over the output format, use the sprintf and fprintf functions.

Controlling Spacing

To control spacing in the output, use the Command Window preference for text display or the format function. Use

```
format compact
```

to suppress blank lines, allowing you to view more information in the Command Window. To include the blank lines, which can help make output more readable, use

```
format loose
```

Number of Characters in Command Window Display

The maximum line length for Command Window display is 25,000 characters. If the output from a statement exceeds this limit, the Command Window truncates the output and displays the following message at the end of the line of output:

```
Output truncated.  Text exceeds maximum line length of 25,000
characters for Command Window display.
```

Clearing the Command Window

Select **Clear Command Window** from the **Edit** menu or context menu
to clear it. This does not clear the workspace, but only clears the view.
Afterwards, you still can use the up arrow key to recall previous functions. A
confirmation dialog box appears if you select the preference for it; for more
information, see "Confirmation Dialogs Preferences" on page 2-112.

Function Alternative

Use `clc` to clear the Command Window. Similar to `clc` is the `home` function,
which moves the prompt to provide a clear screen, but does not clear the
text so you can still scroll up to see it.

Printing Command Window Contents

To print the complete contents of the Command Window, select **File > Print**.
To print only a selection, first make the selection in the Command Window
and then select **File > Print Selection**.

Specify printing options for the Command Window by selecting **File > Page
Setup**. For example, you can print with a header. For more information, see
"Printing and Page Setup Options for Desktop Tools" on page 2-89.

Keeping a Session Log

The diary Function

The `diary` function creates a copy of your session in MATLAB on a disk file,
including keyboard input and system responses, but excluding graphics. You
can view and edit the resulting text file using any text editor, such as the
MATLAB Editor. To create a file on your disk called `sept23.out` that contains
all the functions you enter, as well as output from MATLAB, enter

```
diary('sept23.out')
```

To stop recording the session, use

```
diary('off')
```

To view the file, run

```
edit('sept23.out')
```

Other Session Logs

There are two other means of viewing session information:

- The Command History window contains a log of all functions executed in the current and previous sessions—see "Command History Window" on page 3-65

- The `logfile` startup option—see "Startup Options" on page 1-17.

Finding Text in the Command Window

In this section...

Introduction

You can search for specified text that appears in the Command Window, where the text is either part of input you supplied, or output displayed by the MATLAB software. After finding the text, you can copy and paste it to the prompt in the Command Window to run it, or into an M-file or other file.

For techniques to reuse previous statements and navigate in the Command Window, see also "Recalling Previous Lines in the Command Window" on page 3-18, "Completing Statements in the Command Window — Tab Completion" on page 3-25, and "Keyboard Shortcuts in the Command Window" on page 3-19. To find files and text in files, see "Finding Files and Content Within Files in Any Directory" on page 5-82.

Finding Text Currently Displayed in the Command Window

To search for specified text currently displayed in the Command Window, follow these steps:

1 Select **Edit > Find** when the Command Window is active.

The Find dialog box opens.

.
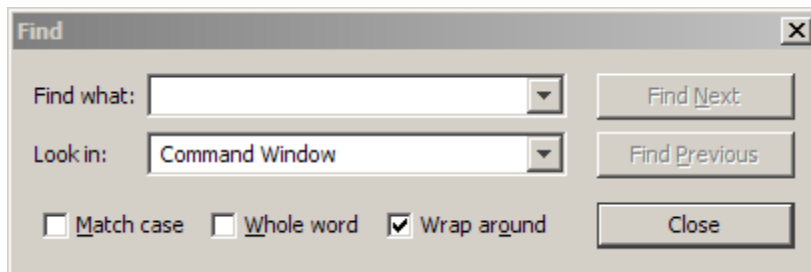
2 Complete the dialog box, and then click **Find Next** or **Find Previous**. Alternatively you can use the keyboard shortcuts **F3** to find the next occurrence and **Shift+F3** to find a previous occurrence.

The search begins at the current cursor position. MATLAB finds the text you specified and highlights it.

3 Repeat step 2 to find another occurrence.

MATLAB beeps when a search for **Find Next** reaches the end of the Command Window, or when a search for **Find Previous** reaches the top of the Command Window. If you have **Wrap around** selected, it continues searching after beeping.

To search for the specified text in other MATLAB desktop tools, change the selection in the **Look in** field.

Increasing the Amount of Information Available for Searching in the Command Window

To increase the amount of information displayed in the Command Window so that more text is available for searching, follow these steps:

1 Select **File > Preferences > Command Window**, and then increase the setting for the "Number of lines in command window scroll buffer" on page 3-64.

2 Do *not* clear the Command Window.

In other words, do not enter clc or select **Edit > Clear Command Window**.

Performing an Incremental Search in the Command Window

With the incremental search feature, the cursor moves to the next or previous occurrence of the specified text in the Command Window. It is similar to the Emacs search feature. To use the incremental search feature in the Command Window, follow these steps:

1 Position the cursor where you want the search to begin.

2 Begin the incremental search using one of the following, depending on your setting for the **Command Window key bindings**. (Available from **File > Preferences > Keyboard**.)

- If you set the preference to **MATLAB Standard (Emacs)**, press **Ctrl+S**

- If you set the preference to **Windows** , press **Ctrl+Shift+S**

3 To look for the previous occurrence, press **Ctrl+R** or **Ctrl+Shift+R** instead.

An incremental search field, **Inc Search**, appears at the bottom of the Command Window and is preceded by **F** for a forward search, or **R** when you are looking for the previous occurrence (reverse search).

Search begins at current cursor position.

Incremental search field.

4 In the **Inc Search** field, type the text you want to find. For example, look for `Boston`.

As you type the first letter, `b`, the first occurrence of that letter in the Command Window after the current cursor position is highlighted. For the example shown, the first occurrence of `b` is highlighted, the `b` in `Berlin`. Note that incremental search allows for case sensitivity — see "Case Sensitivity in Incremental Search" on page 3-58.

MATLAB finds the next b.
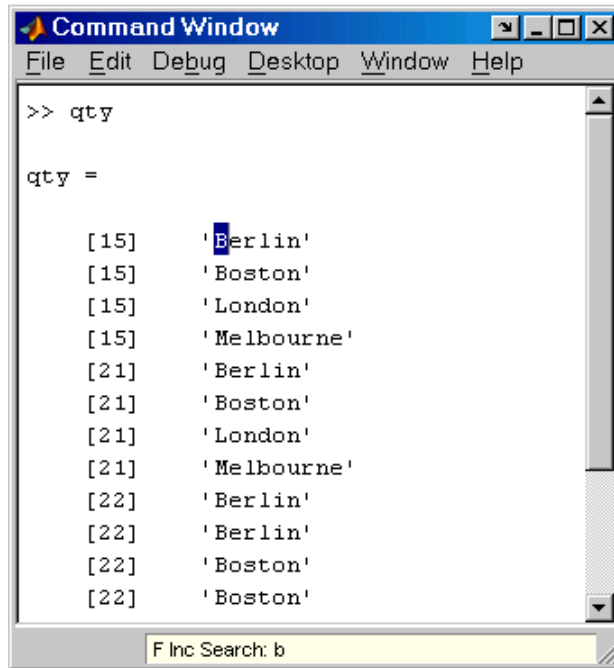
When you type the next letter, the first occurrence of the text becomes highlighted. In the example, when you add the letter o to the b so that the **Inc Search** field now has bo, the bo in Boston becomes highlighted.

- If you mistype in the **Inc Search** field, use the **Back Space** key to remove the last letters and make corrections.

- After finding the bo, you can press **Ctrl+W** to complete that word. In this example, Boston appears in the **Inc Search** field.

5 To find the next occurrence of Boston in the Command Window, press **Ctrl+S**. To find the previous occurrence of the text, press **Ctrl+R**

6 If MATLAB beeps, it means either that the text was not found, or the search wrapped past the end (or beginning) of the Command Window and continued at the beginning (or end).

3-57

- When the text is not found, **Failing** appears in the incremental search field. Modify the search term in the incremental search field and try again. Use **Ctrl+G** to automatically remove characters back to the last successful search. For example, if `plode` fails, **Ctrl+G** removes the `de` from the search term because `plo` does exist in the Command Window.

7 To end the incremental search, press **Esc** (escape) or **Enter**, or any other key that is not a character or number.

The **Inc Search** field no longer appears. The cursor is at the position where the text was last found, with the search text highlighted.
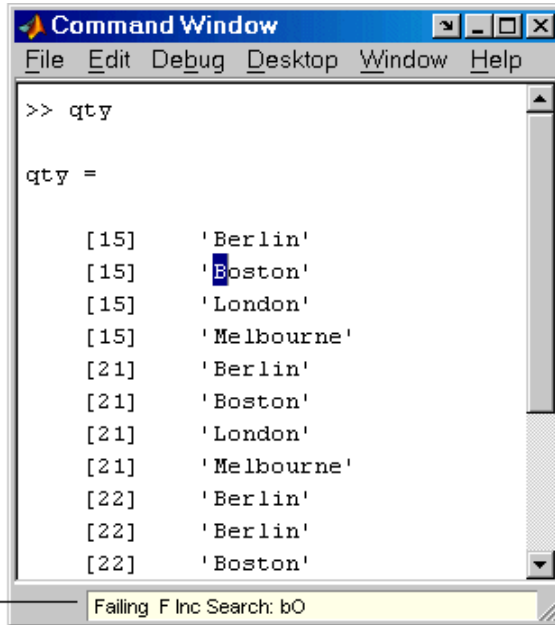
Incremental search is also available in the Editor — see "Incremental Search" on page 6-75.

Case Sensitivity in Incremental Search

When you enter lowercase letters in the **Inc Search** field, for example, `b`, incremental search looks for both lowercase and uppercase instances of the letters, for example `b` and `B`. However, if you enter uppercase letters, for example, `B`, incremental search only looks for instances that match the case you entered.

In the example, enter b0 in the **Inc Search** field and incremental search does not find any matching text.

Incremental search is case-sensitive when you enter uppercase letters. Here, no matches are found for b0.

```
>> qty

qty =

    [15]    'Berlin'
    [15]    'Boston'
    [15]    'London'
    [15]    'Melbourne'
    [21]    'Berlin'
    [21]    'Boston'
    [21]    'London'
    [21]    'Melbourne'
    [22]    'Berlin'
    [22]    'Berlin'
    [22]    'Boston'
```

Failing F Inc Search: bO

Preferences for the Command Window

In this section...
"Text, Display, Accessibility, and Tab Size Preferences" on page 3-60
"See Also" on page 3-64

Text, Display, Accessibility, and Tab Size Preferences

To set these preferences for the Command Window, select **File > Preferences** and then select **Command Window** in the left pane of the Preferences dialog box.
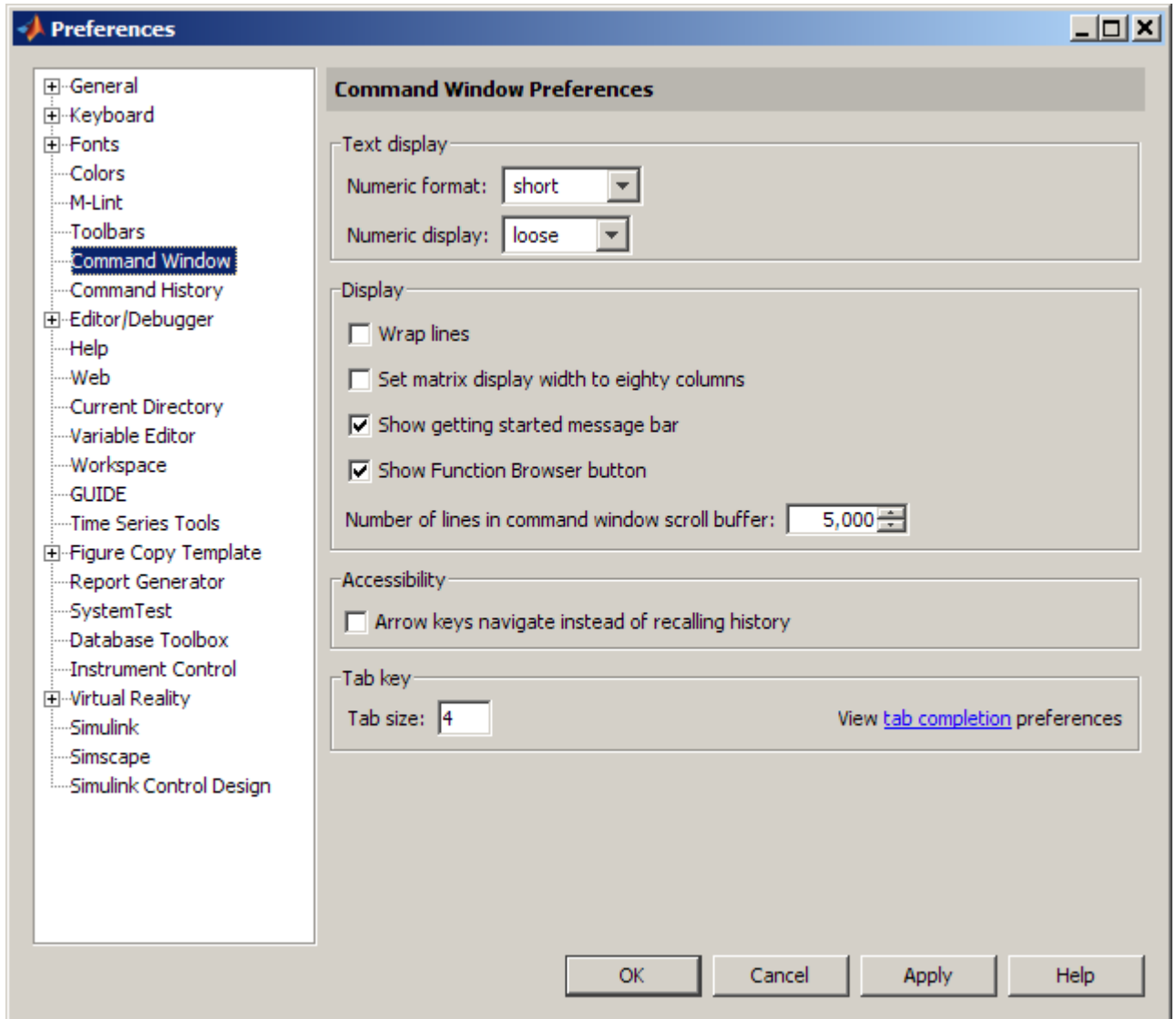
Text Display

Specify the format, that is, how output appears in the Command Window.

Numeric format. Specify the output format of numeric values displayed in the Command Window. This affects only how numbers are displayed, not how the MATLAB software computes or saves them. The format reference page includes the list of available formats, with examples.

Numeric display. Specify spacing of output in the Command Window. To suppress blank lines, use compact. To display blank lines, use loose. For more information, see the reference page for format.

Display

Wrap lines. Select to make a single line of input or output in the Command Window break into multiple lines in order to fit within the current width of the Command Window. This is useful for console mode. With this option selected, an entire line is visible without scrolling, and the horizontal scroll bar does not appear because it is not needed. With this option cleared, use the horizontal scroll bar to view the entire contents of the line.

Set matrix display width to eighty columns. When selected, MATLAB displays 80 characters of matrix output in a single row, and then continues displaying output in a new row, regardless of the width of the Command Window. Use the horizontal scroll bar if the width of the Command Window is less than 80 characters.

With the check box cleared, a row of matrix output fills the width of the Command Window, and then continues displaying output in a new row. Note that if the **Wrap lines** preference is also selected, and the width of the Command Window is less than 80 characters, each row of 80 characters of matrix output wraps to fit within the width of the Command Window.

To determine the number of characters and lines that will display in the Command Window, given its current size, use

```
get(0,'CommandWindowSize')
```

When the matrix display width preference is *not* selected, the number of characters for the width is based on the current width of the Command Window. For example, a result of 50, 25 means 50 characters will display

across the Command Window, and 25 lines will display. However, with the preference selected, the result for that same size Command Window is `80, 25`.

Show getting started message bar. The message bar in the Command Window includes links to a video, demos, and information on getting started with MATLAB. If you want to remove the message bar in the Command Window, click the Close box in the right corner of the bar. If you close the message bar, you can still access the documentation and demos it linked to—for more information, see Chapter 4, "Help, Demos, and Related Resources".

If you closed the message bar and want to show it again, select the **Show getting started message bar** check box in the Command Window **Display** preferences.

Getting started
message bar.
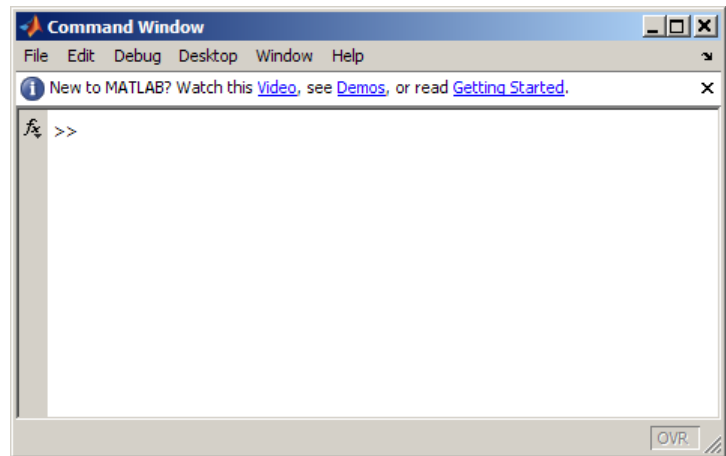
Show Function Browser button. The Function Browser button $f_{\bullet}$ appears to the left of the prompt in the Command Window. You use it to access the Function Browser. If you do not want the button to appear because of the space it requires, you can hide it by clearing the **Show Function Browser button** check box. When the button is not shown, you can access the Function Browser by pressing **Shift+F1** or by right-clicking in the Command Window and selecting **Function Browser** from the context menu. For more information about the Function Browser, see "Finding Functions Using the Function Browser" on page 3-41.

Number of lines in command window scroll buffer. Set the number of lines maintained in the Command Window, from 1,000 to 25,000. This is the number of lines you can see when you scroll vertically. A larger buffer means you can view more lines and it provides a larger base for search features, but requires more memory.

This preference setting does not impact the number of lines you can recall when you use the up arrow key in the Command Window. Using the up arrow key, you can recall all lines shown in the Command History window, regardless of how many lines you can see in the Command Window.

Accessibility

Select this option to use the up and down arrow keys to move the cursor when it is above the command line. With this preference selected, use the **Ctrl+** up arrow or down arrow key to recall statements using key bindings for Microsoft Windows and Apple Macintosh platforms, or **Ctrl+P** and **Ctrl+N** for **MATLAB standard (Emacs)** key bindings.

Clear this preference to use the up and down arrow keys to recall statements. Use the mouse and other features to move the cursor when above the command line.

Tab key

Tab size. Number of spaces assigned to a tab stop when displaying output. The default is four spaces, except on UNIX[13] platforms where the default is eight spaces. This does not apply when the tab completion preference is selected.

See Also

- "Setting Fonts Preferences for Desktop Tools" on page 2-121
- "Confirmation Dialogs Preferences" on page 2-112
- "Setting Keyboard Preferences for Desktop Tools" on page 2-116

---

13. UNIX is a registered trademark of The Open Group in the United States and other countries.

Command History Window

In this section...
"Overview of the Command History Window" on page 3-65
"Viewing Statements in the Command History Window" on page 3-66
"Using Statements from the Command History Window" on page 3-68
"Searching in the Command History Window" on page 3-69
"Printing the Command History Window" on page 3-74
"Deleting Entries from the Command History Window" on page 3-74

Overview of the Command History Window

The Command History window displays a log of the statements most recently run in the Command Window. If you have an active Internet connection, you can watch the Command History video demo for an overview of the major functionality.

To show or hide the Command History window, use the **Desktop** menu. Alternatively, use commandhistory to open the MATLAB Command History window when it is closed, or to select it when it is open. For details, see "Opening and Arranging Desktop Tools" on page 2-5.

Timestamp marks the start of each session. →

Select one or more entries and right-click to copy, evaluate, or create an M-file from the selection. →


```
%-- 6/8/07 10:42 AM --%
    clear all
    m=3
%-- 6/12/07 11:01 AM --%
    collatz(3)
    clear all
    dbquit
    collatzplot_new(3)
    dbquit
```
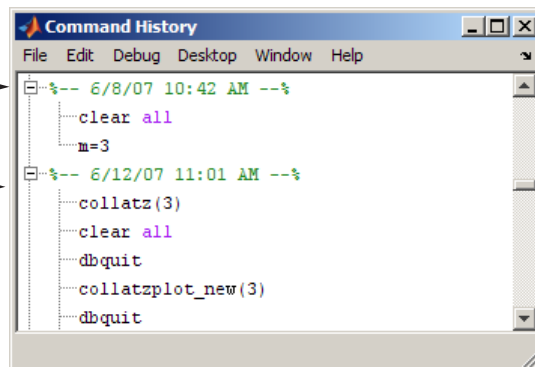
MATLAB provides other options for viewing a history of statements. See also the following sections:

- "Recalling Previous Lines in the Command Window" on page 3-18, which describes using the up arrow in the Command Window

- The `diary` function reference page

- "Startup Options" on page 1-17, which includes the `logfile` startup option

Command History File

The Command History window and the Command Window's feature for "Recalling Previous Lines in the Command Window" on page 3-18 both use the command history file, `history.m`, which is stored in the same directory as preferences for MATLAB. Type `prefdir` in the Command Window to see the location of the file. The command history file is loaded when MATLAB starts, and it stores a maximum of 20,000 bytes, deleting the oldest entries as needed to maintain that size.

Statements saved to the history are those that run in the Command Window. This includes statements you run using the **Evaluate Selection** item on context menus in tools such as the Editor, Command History, and Help browser. The history does not include every action taken in MATLAB, however, For example, if you run the statement

```
a = 1:10
```

and then modify the value of `a` in the Variable Editor, there is no record in the history that you modified the value of `a`.

MATLAB automatically saves the command history file throughout the session according to the **Saving** preference you specified. You can choose to automatically exclude certain statements from being written to the command history file with the **Settings** preference. For details, see "Preferences for Command History" on page 3-76.

Viewing Statements in the Command History Window

The Command History window lists statements you ran in the current session and in previous sessions. The time and date for each session appear at the top of the history of statements for that session. Use the scroll bar or the up and down arrow keys to move through the Command History window.

Click **-** to hide the history for a session, and click **+** to show it. Select a timestamp to select all entries for that session. With a timestamp selected, you can press the **+** or **-** key on the numeric keypad to show and hide entries.

Using Statements from the Command History Window

You can select entries in the Command History window and then perform the following actions for the selected entries.

Action	How to Perform the Action
Run statements in the Command Window	Double-click an entry (entries) in the Command History window to execute the statement(s) in the entries. For example, double-click `edit myfile` to open `myfile.m` in the Editor. You can also run the statements in an entry by right-clicking the entry and selecting **Evaluate Selection** from the context menu, or by selecting an entry and pressing **Enter** or **Return**.
Edit and run statements in the Command Window	Select an entry or entries and then select **Copy** from the context menu. Paste the selection into the Command Window. Alternatively, drag the selection to the Command Window. Then in the Command Window, edit the statements, and press **Enter** or **Return** to execute them.
Copy statements to another window	Select an entry or entries and then select **Copy** from the context menu. Paste the selection into an open M-file in the Editor or any application. Alternatively, drag the selection from the Command History window to an open M-file or another application.
Create an M-file from statement(s)	Select an entry or entries and then right-click and select **Create M-File** from the context menu. The Editor opens a new M-file that contains the statements you selected from the Command History window.
Create a shortcut from statement(s)	Select an entry or entries and then right-click and select **Create Shortcut** from the context menu. Alternatively, drag the selection to the **Shortcuts** toolbar. The Shortcut Editor opens and the selected statements appear in the **Callback** field. For more information, see "Using MATLAB Shortcuts to Easily Run a Group of Statements" on page 2-59.

Searching in the Command History Window

There are two types of search in the Command History window:

- "Finding Next Entry By Letter" on page 3-69
- "Finding Text" on page 3-73

After finding an entry, you can copy and paste it into an M-file or any file, or you can right-click and select **Evaluate Selection** to run the entry.

Finding Next Entry By Letter

Type a letter in the Command History window. The Command History window searches backwards to find the last previous entry that begins with that letter as illustrated in this example:

1 Position the cursor at anywhere in the Command History window.
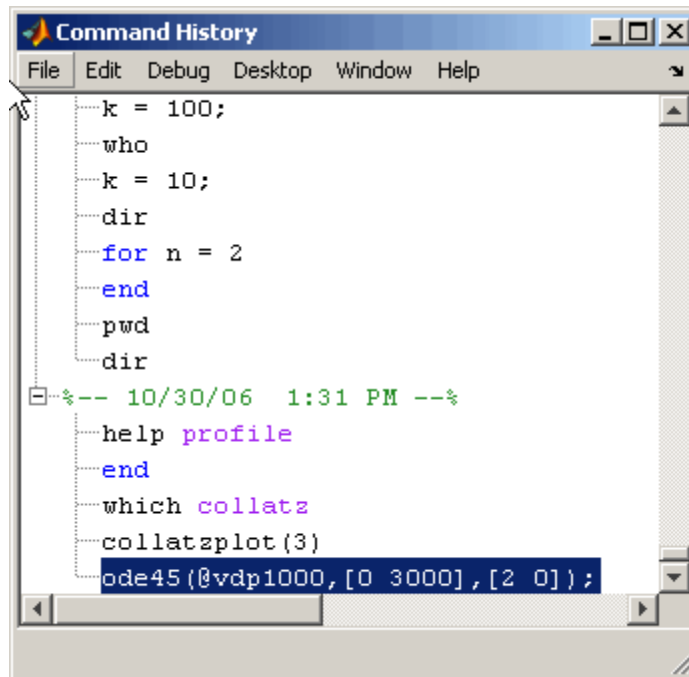
2 Type the first letters of the entry you want to find. For example, type `my`.

The Command History window searches backwards and selects the previous entry that begins with the letters you typed; in this example, you typed `my`, and the Command History finds `myfor`.

As you begin typing a ToolTip with the text: `Search history for:`, appears at the top of the Command History window. This ToolTip keeps track of your search target as you type additional letters to narrow the focus of your search.

If the search finds a matching entry in a session that is collapsed, it expands the session and selects the entry.

Incremental search target. Changes
as you type additional letters.
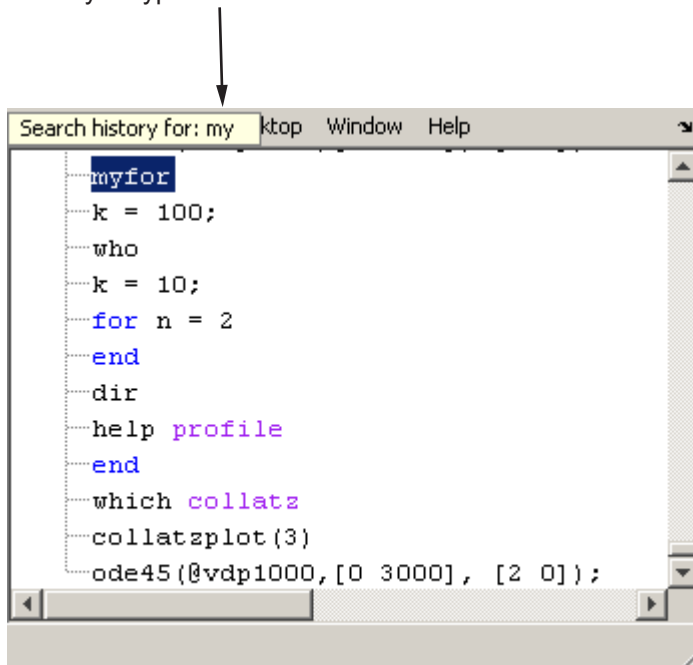
3 Now type an `s` to extend the search to `mys`. The Command History window continues to search backwards, stopping next at the function `mysurf`.
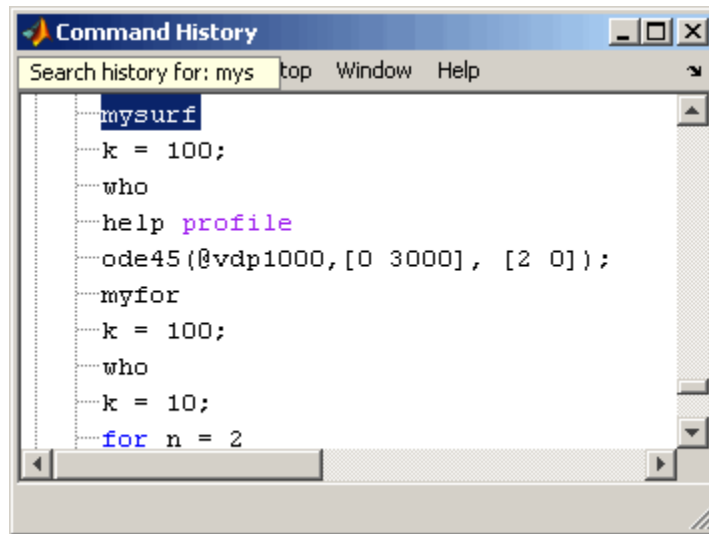
Finding Multiple Occurrences of the Entry. You can use the up and down arrow keys to search for the next or the previous occurrence of the entry you just found.

When you press **Ctrl** and the up or down arrow key, each occurrence of the entry remains highlighted while you search for additional instances.
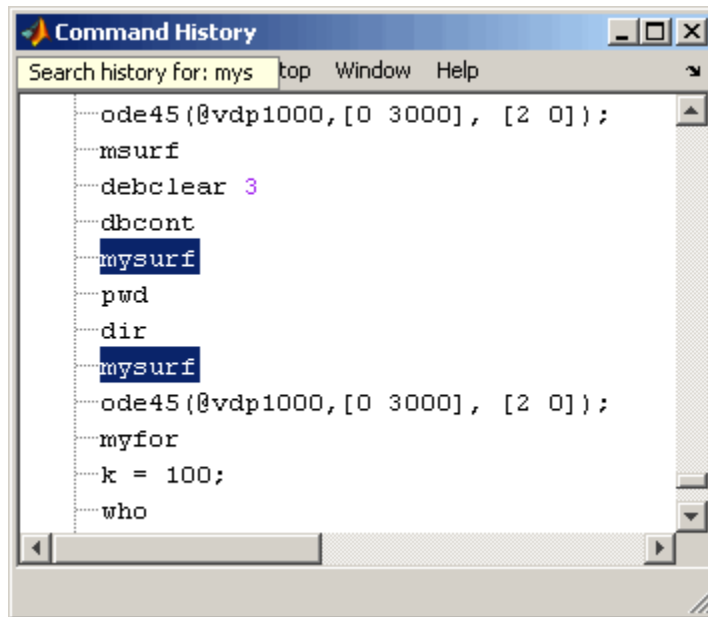
To highlight all instances of the entry, press **Ctrl+A**. In the example below, all instances of entries beginning with my are highlighted.
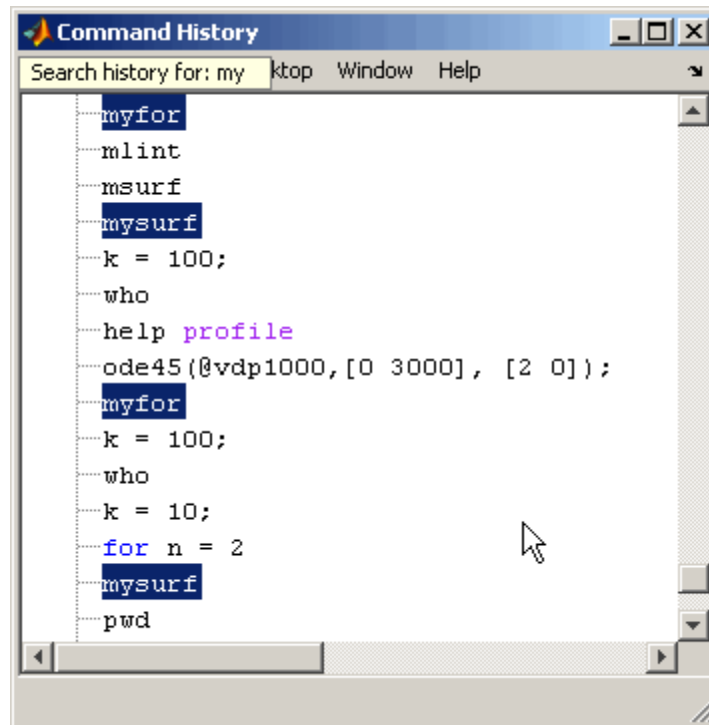
Finding Text

Select **Find** from the **Edit** menu to search for specified text using the Find dialog box. Complete the dialog box. The search begins at the current cursor position. MATLAB finds the text you specified and highlights it. Click **Find Next** or **Find Previous** to find another occurrence, or use the keyboard shortcuts **F3** and **Shift+F3**. Find looks for visible entries only, that is, it does not find entries in collapsed nodes.
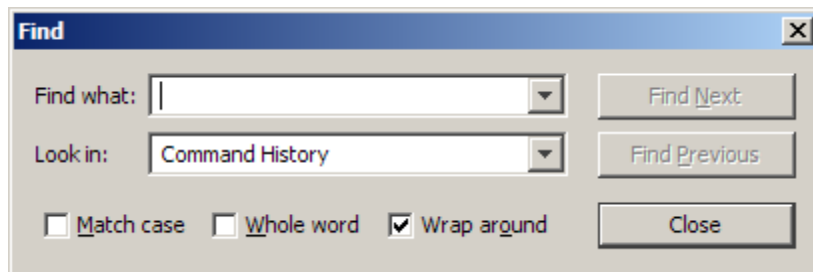
MATLAB beeps when a search for **Find Next** reaches the end of the Command History window, or when a search for **Find Previous** reaches the top of the Command History window. If you have **Wrap around** selected, it continues searching after beeping.

Change the selection in the **Look in** field to search for the specified text in other MATLAB desktop tools.

Printing the Command History Window

To print the contents of the Command History window, select **File > Print** or **Print Selection**. Specify options for printing by selecting **File > Page Setup**. For example, you can print the history with a header. For more information, see "Printing and Page Setup Options for Desktop Tools" on page 2-89.

The printed version is sized to fit the page. If there is a long statement in the Command History, the reduced page size might be difficult to read. As a workaround, either use **Print Selection**, where the long statement is not part of the selection, or remove any extremely long statements from the Command History before printing it.

Deleting Entries from the Command History Window

Delete entries from the Command History window when you think there are too many and it becomes inconvenient to find the ones you want. All entries remain until you delete them, or until the command history file exceeds its maximum size, at which point MATLAB automatically deletes the oldest entries. For more information, see "Viewing Statements in the Command History Window" on page 3-66.

To delete entries in the Command History window, first select the entries to delete, using one of these methods:

- Select a single entry.
- **Shift**+click or **Ctrl**+click to select multiple entries.
- Select the timestamp for a session to select all entries for that session. Then use **Shift**+click or **Ctrl**+click to select multiple timestamps with all their entries.

Then, right-click and select **Delete Selection** from the context menu, or press the **Delete** key. A confirmation dialog box might appear; for more information, see "Confirmation Dialogs Preferences" on page 2-112.

To delete all entries, select **Edit > Clear Command History**, or select **Clear Command History** from the context menu.

After deleting entries from the Command History window, you will not be able to recall those statements in the Command Window as described in "Recalling Previous Lines in the Command Window" on page 3-18.

Preferences for Command History

In this section...
"Overview of Command History Preferences" on page 3-76
"Settings" on page 3-76
"Saving" on page 3-77
"See Also" on page 3-78

Overview of Command History Preferences

Using Command History preferences, you can choose to exclude statements from the command history file, `history.m`, and specify how often to save it. The command history file is used for both the Command History window and statement recall in the Command Window.

To set preferences for the command history file, select **File > Preferences**, and then select **Command History** in the Preferences dialog box.

Settings

Specify the types of statements to exclude from the command history file. Note that when you exclude statements from the command history file, you cannot recall them in the Command Window as described in "Recalling Previous Lines in the Command Window" on page 3-18, nor can you view them in the Command History window.

Save Exit/Quit Commands

Select the check box to save `exit` and `quit` commands in the command history file.

Save Consecutive Duplicate Commands

Select the check box if you want consecutive executions of the same statement to be saved to the command history file.

For example, with this option selected, run `magic(5)`, and then run `magic(5)` again. The command history file saves two consecutive entries for `magic(5)`.

With this option cleared, for the same example, the command history file saves only one entry for `magic(5)`. If you then run `magic(10)`, the command history file saves both entries, `magic(5)` followed by `magic(10)`.

Saving

Use **Saving** preferences to specify how often to automatically save the command history file during a session of running the MATLAB software. By default, MATLAB saves the history after every statement. This allows you to more easily recover your state in the event of an abnormal termination, because you can reconstruct it using the history.

Save History File On Quit

Select this option to save the command history file when you end the session of MATLAB. If the session does not end via a normal termination, that is, via the `exit` or `quit` functions, **File > Exit MATLAB**, or the MATLAB desktop Close box, the history file is not saved for that session.

Save After n Commands

Select this option to save the command history file after `n` statements are added to the file. For example, when you select the option and set `n` to `10`, after every 10 statements are added, the history file is automatically saved. Use this option instead of **Save History File on Quit** if you don't want to risk losing entries to the saved history because of an abnormal termination, such as a power failure.

Don't Save History File

Select this option if you do not want to save the command history file. This feature is useful when multiple users share the same machine and do not want other users to view the statements they have run.

Note that any entries already in the `history.m` file remain. Prior to setting this preference, you might want to remove any existing entries. Follow the instructions in "Deleting Entries from the Command History Window" on page 3-74.

See Also

- "Command History Window" on page 3-65
- Additional preferences that relate to the Command History:
  - "Setting Fonts Preferences for Desktop Tools" on page 2-121
  - "Confirmation Dialogs Preferences" on page 2-112

4

Help, Demos, and Related Resources

Ways to Get Help for MathWorks Products

The Help browser provides documentation for all your installed products and is the primary way you get help—for more information, see "Help Browser Overview" on page 4-5. Other forms of help include getting help in the Command Window, running examples and demos, and searching Technical Support solutions on the Web site. If you have an active Internet connection, you can watch the Help and Documentation video demo for an overview of the major functionality. In addition to using resources provided by The MathWorks, you can also find files created by users and participate in a user newsgroup.

The following table summarizes some of the different ways to get help for MathWorks products.

If You Want To...	Use...
Get help about the syntax for a function	• "Highlighting Syntax to Help Ensure Correct Entries" on page 3-24 and "Matching Delimiters (Parentheses)" on page 3-25. • "Completing Statements in the Command Window — Tab Completion" on page 3-25. • "Viewing Function Syntax While Entering a Statement — Function Hints" on page 3-34. • "Viewing Information About the Selected File or Directory in the Current Directory Browser" on page 5-61.

If You Want To...	Use...
Learn more than just the syntax for a function	• "Getting Help in the Command Window — the help Function" on page 4-61. • "Viewing Function Reference Pages — the doc Function" on page 4-60. • "Finding Functions Using the Function Browser" on page 3-41. • "Getting Help for the Selected Function in the Command Window or Editor" on page 3-39 and "Getting Help for the Selected Function While Viewing a Help Page" on page 4-36.
Find a function to use	• "Finding Functions Using the Function Browser" on page 3-41. • Functions By Category or Alphabetical List of Functions, accessible from the top of each product roadmap page in the Help browser. • "Searching Documentation and Demos with the Help Browser" on page 4-18.
Learn how to improve your MATLAB code	"Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 6-123.
Learn the basics about a product or tool	The Help browser and refer to getting started material, accessible from the product roadmap page.
See all that you can do with a product or tool	The Help browser and scan the documentation Contents listing for the product.
Find information about a product, tool, or feature	• "Searching Documentation and Demos with the Help Browser" on page 4-18. • "Using the Help Browser Index to Find Keywords in the Documentation" on page 4-15.

If You Want To...	Use...
See and run examples and demonstrations for a product, tool, or feature	• "Viewing and Running Demos" on page 4-41. • The index of all Examples in the product documentation, accessible from the Help browser Contents listing for a product.
Special context-specific help	Additional ways to get help in some tools and products, such as entries in the **Help** menu, **Help** buttons in dialog boxes, selecting **Help** from a context menu, and pressing the **F1** key. These special ways are described in the documentation for the product or tool that uses it.
View documentation for all products	Documentation for all MathWorks products on the Web
View documentation in Japanese	Japanese versions of the MATLAB product, which include documentation that has been translated into Japanese.

Note You cannot read MATLAB documentation files from the installation media for MATLAB. You also cannot use a Web browser to read the documentation files installed with MATLAB because the files are compressed JAR files—you can access them only with the Help browser.

Help Browser Overview

In this section...
"About the Help Browser" on page 4-5
"Opening the Help Browser" on page 4-5
"Resizing the Help Browser" on page 4-7
"Types of Documentation" on page 4-9

About the Help Browser

The Help browser is an HTML browser integrated with the MATLAB desktop. Use the Help browser to search and view documentation and demonstrations for MATLAB and all other installed MathWorks products. MATLAB automatically installs the documentation and demos for a product when you install that product.

Opening the Help Browser

To open the Help browser, click the Help button 🕐 in the desktop toolbar, type helpbrowser in the Command Window, or use the **Help** menu in any tool. There are two panes:

- The Help Navigator, on the left, for finding information, includes a search field, and includes **Contents**, **Index**, **Search Results**, and **Demos** tabs. For more information, see "Finding Information with the Help Browser" on page 4-11.

- The display pane, on the right, for viewing documentation and demos.

Tabs in the **Help Navigator** pane provide
different ways to find information.

View documentation
in the display pane.

Use the close box
to hide the pane.

Drag the separator bar to
adjust the width of the panes.
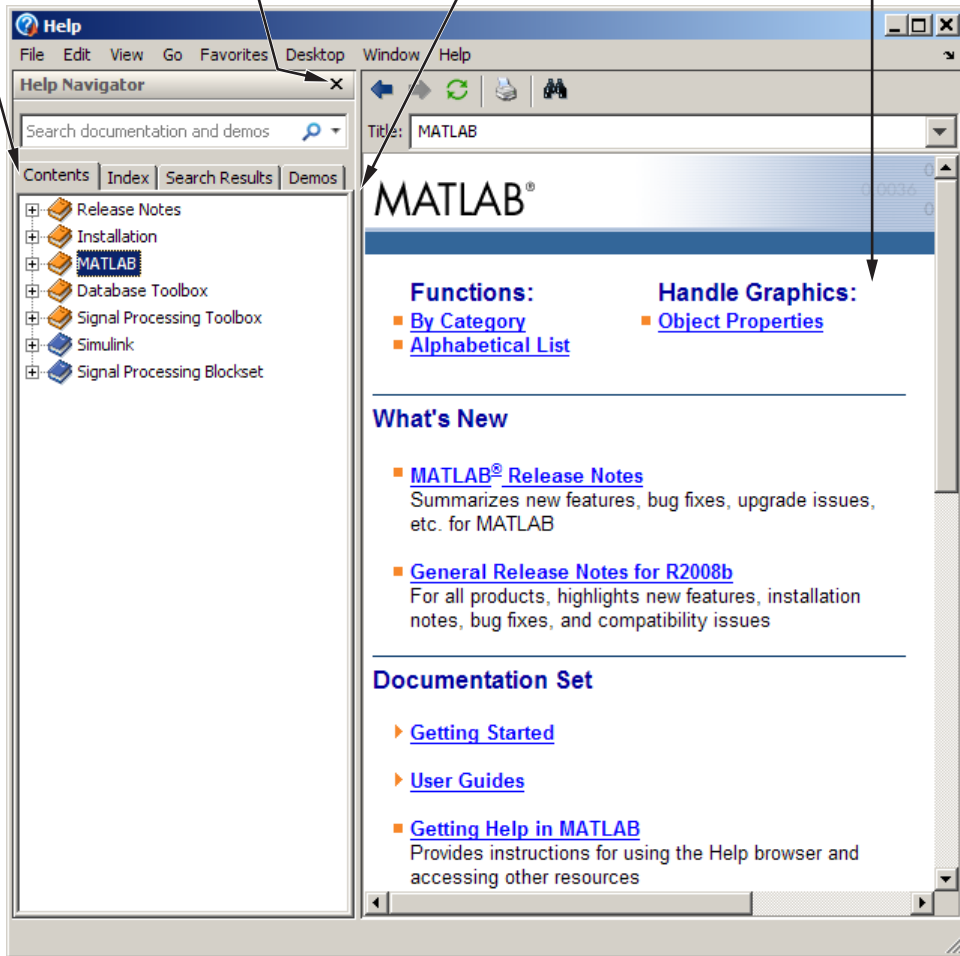
Resizing the Help Browser

To adjust the relative width of the two panes, drag the separator bar between them. You can also change the font in either of the panes — see "Font and Color Preferences for the Help Browser" on page 4-53.

Once you find the documentation you want, you can close the **Help Navigator** pane so there is more screen space to view the information itself. This is shown in the following figure. To close the **Help Navigator** pane, click the Close box **X** in the pane's upper right corner. To open the **Help Navigator** pane from the display pane, click the Help Navigator button ▣ on the toolbar. Alternatively, use the **View** menu.

Help — □ ×

File   Edit   View   Go   Favorites   Desktop   Window   Help

Title: Assistance While Entering Statements :: Running Functions - Command Window and History (MATLAB®)   ▼

Completing Statements in the Command Window — Tab Completion

MATLAB helps you complete the names of known items as you type them in the Command Window so that you can avoid spelling mistakes and do not have to spend time looking up the information in other tools. These are the items for which MATLAB can complete the names:

- Function or model on the search path or in the current directory
- Filename or directory
- Variable, including structures, in the current workspace
- Handle Graphics® property for figure in the current workspace

Type the first few characters of the item name and then press the **Tab** key. To use tab completion, you must have the tab completion preference for the Command Window selected. For details, see Keyboard Preferences.

Tab completion is also available in the Editor, but there are some slight differences in usage. See Tab Completion in the Editor.

These examples demonstrate how to use tab completion in the Command Window:

- Basic Example — Unique Completion
- Multiple Possible Completions
- Tab Completion for Directories and Filenames
- Tab Completion for Structures
- Tab Completion for Properties

Basic Example — Unique Completion

Types of Documentation

The Help browser and help functions provide access to the following types of information for all installed MathWorks products. The icons shown here appear in the Help browser contents listing to help you quickly identify documentation by type.

Icon	Type of Documentation	Description and When to Use
●	Getting Started	Review Getting Started documentation before you begin using a product or feature for the first time. Then, to learn more, go to the user guides, reference pages, demos, and examples.
◆ or ◆ or ◆	Product	MATLAB, toolboxes, and related products use orange book icons ◆. Simulink software, blocksets, and related products use blue book icons ◆. Link and Target products use green book icons ◆.
💡	Examples	Accessible via the Help browser **Contents** listing, this is an index of the major examples included in the Help browser documentation.
📄	Users Guide (blue)	User guide material contains overviews as well as detailed instructions. Consult it after reviewing Getting Started material.
📄	Reference Pages (orange)	Each function has a reference page that provides the syntax, description, examples, and other information for that function. Each reference page includes links to related functions and additional information. Reference pages are also provided for blocks and properties.
📝	Release Notes	An overview of new products and features in a release. Release Notes also include upgrade information, links to fixed and known problems, and compatibility considerations. Review the Release Notes for all your products when you first start using a new release. Release Notes for the current version include the release notes for multiple prior versions.

Icon	Type of Documentation	Description and When to Use
	Printable Documentation	Most products provide access to the online documentation in a printable format, PDF. Access PDF files via the Help browser and print them from your PDF reader, such as the Adobe® Acrobat® product. Most PDF files reside only on the MathWorks Web site, so you need an Internet connection to view them.
none	Demos	MathWorks products come with demonstrations that run key features of the product. Many of the demos run MATLAB code. Use the Help browser **Demos** pane or **Search Results** to access demos for the products you have installed.
none	M-File Help	Get M-file help in the Command Window to quickly access basic information for a function or model. It provides a brief description of a function and its syntax. It is called M-file help because the text of the help is a series of comments at the top of the M-file for a function.

Finding Information with the Help Browser

In this section...
"About the Help Navigator" on page 4-11
"Browsing Contents in the Help Browser" on page 4-11
"Using the Help Browser Index to Find Keywords in the Documentation" on page 4-15
"Searching Documentation and Demos with the Help Browser" on page 4-18
"Labeling Pages as Favorites in the Help Browser" on page 4-26

About the Help Navigator

The **Help Navigator** is in the left pane of the Help browser. It provides a table of contents, an index, a search feature, and a list of demos to help you find information.

Browsing Contents in the Help Browser

To list the documentation titles and tables of contents for products you installed, click the **Contents** tab in the **Help Navigator** pane. To show documentation for only some of the installed products, use the product filter.
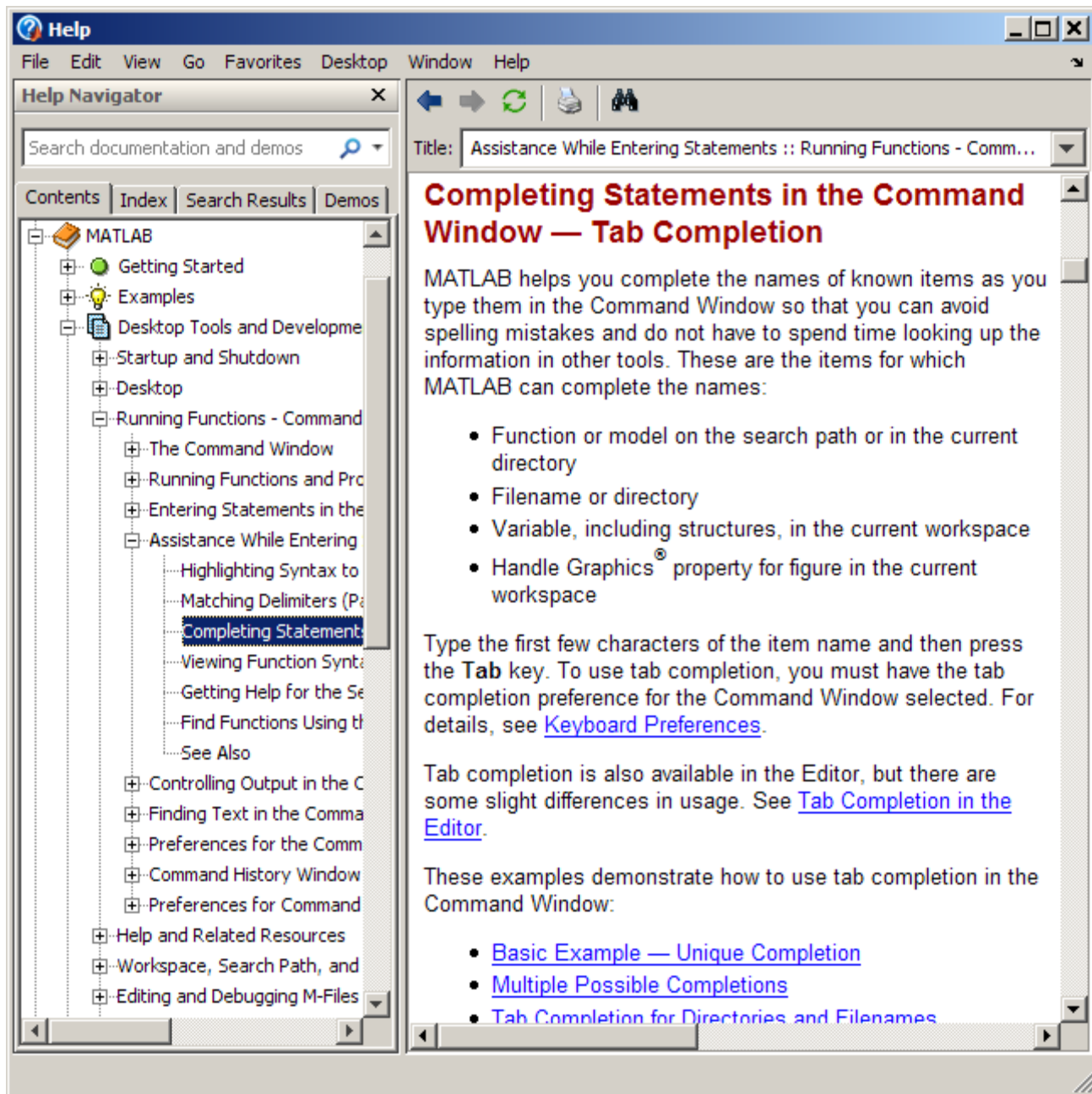
Using the Product Roadmap

When you select a product in the **Contents** pane (any entry with a book icon 📙), such as the MATLAB or Communications Toolbox products, a *roadmap* of the documentation for that product appears in the display pane. The roadmap includes links to commonly used documentation sections, including

- Function and block references pages

- An index of major examples in the documentation

- The PDF version of the documentation, which is suitable for printing (this is the only direct access from the MATLAB software to the printable documentation)

To access the roadmap page using a function, run `doc toolboxdirname`. For example, `doc matlab` displays the MATLAB roadmap page in the Help browser. For more information, see the reference page for the `doc` function.

Navigating the Contents Listing

In the **Contents** listing, you can

- Click the **+** to the left of an item to show the first page of that document or section in the display pane and expand the listing for that item in the **Help Navigator** pane. You can alternatively: double-click the item, press the right arrow key, or press **+** on the numeric keypad.

- Click the **-** to the left of an item to collapse the listings for that item. You can alternatively: double-click the item, press the left arrow key, or press **-** on the numeric keypad.

- Select an item to show the first page of that document or section in the display pane.

- Press ***** on the numeric keypad to show all subentries for the selection.

- Use the down and up arrow keys to move through the list of items.

About Icons in the Contents Listing

Icons for entries in the top levels of the **Contents** pane listing represent the type of documentation so you can quickly find the kind of information you

need for a product. For a description of the icons, see the table in "Types of Documentation" on page 4-9.

Synchronizing the Contents Listing and Demos Listing with the Display Pane

By default, the topic highlighted in the **Contents** pane matches the title of the page appearing in the display pane. The **Contents** pane listing is said to be synchronized with the displayed document. This feature is useful if you access documentation with a method other than the **Contents** pane, for example, using a link in a page in the display pane, or selecting a search result. With synchronization, you know what book and section the displayed page is part of, and where that section fits within the overall book. Note that synchronization only applies to the major headings in a document. For pages that begin with lower level headings, the **Contents** pane listing does not synchronize.

You can turn off synchronization. To do so, use preferences. See "General — Keeping Contents Synchronized" on page 4-52.

Synchronization applies to the **Contents** and **Demos** panes. The page shown in the display pane does not necessarily correspond to the selection in the **Index**, or **Search Results** panes. However, if you return to the **Contents** pane (or **Demos** pane), the displayed page synchronizes with the **Contents** (or **Demos**) pane.

This example illustrates synchronization for after performing a search. When you enter `"interactive plotting"` in the search field, the **Search Results** pane displays a list of results in the Help Navigator, with the first documentation result selected. The display pane shows the page corresponding to that first result, `Plotting Tools   Interactive Plotting`. When you click the **Contents** tab, the tree automatically expands to show `MATLAB > Graphics > Plots and Plotting Tools` and selects the `Plotting Tools   Interactive Plotting` entry.

This example illustrates synchronization for demos. When you run

```
demo('matlab', 'graphics')
```

the **Demos** pane appears, with the MATLAB Graphics entry selected.

Using the Help Browser Index to Find Keywords in the Documentation

To find specific index entries (selected keywords) in the MathWorks documentation for installed products, use the **Index** in the **Help Navigator** pane.

Help Navigator   ×

Search documentation and demos   🔍 ▾

| Contents | Index | Search Results | Demos |

A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

Enter index term: [   ]

Name	Product
abs	MATLAB
Abs block	Simulink
zero crossings	
abs function	Signal Proces...
absolute accuracy	
BVP	MATLAB
DDE	MATLAB
ODE	MATLAB
absolute tolerance	
definition	Simulink
simset parameter	Simulink
simulation accuracy	Simulink
specifying for a block state	Simulink
absolute value	
generating	Simulink
absolute value	MATLAB
ac2poly function	Signal Proces...
ac2rc function	Signal Proces...
accelbuild command	
building Accelerator MEX-file	Simulink
Acceleration	
code regeneration	Simulink
compilation overhead	Simulink
debugger advantages	Simulink
decision tree	Simulink
designing for	Simulink
how to run debugger with	Simulink
inhibiting	Simulink
numerical precision	Simulink

Follow these steps to use the Index.

1 Click the **Index** tab.

2 Type a word or words in the **Enter index term** field. As you type, the **Index** pane displays matching entries and their subentries (indented). It might take a moment for the display to appear. The index is not case sensitive. If there is not a matching entry, it displays the page for the letter that your entry begins with.

The product whose documentation includes the matching index entry is listed next to the index entry, which is useful when there are multiple matching index entries. You might have to make the **Help Navigator** pane wider to see the product.

3 Select a blue index entry from the list (where blue represents a hyperlink) to display the page to which the term refers. Multiple links per entry are denoted by numbers in brackets following the term. (Black index entries are only index headings and do not link to any page.)

When you select an entry, its color becomes red. The page whose entry you selected appears in the display pane, scrolled to the location that the entry references.

4 To see more matching entries, scroll through the list.

Tips for Using the Index

- To see entries only for products you specify, select **File > Preferences > Help**, and use the Filter by Product settings. For more information, see "Setting the Scope of Documentation — Product Filter" on page 4-47.

- To see entries for all installed products, select **File > Preferences > Help**, and select **All products** in the Filter by Product section.

- For more or different results, type a different term or reverse the order of the words you type. For example, if you are looking for an entry about tab completion under the `Editor` entry, and that subentry does not exist, instead try `tab completion` and look for the `Editor` subentry .

- After selecting an entry, search for specified text in the displayed page using the Find tool, which you access with the Find button on the display pane toolbar.

- When there are multiple matching entries, refer to the product associated with each entry, which appears in the second column of the **Index** results. You might need to make the pane wider to see the column.

- For different but related results, try using the search field—for instructions, see "Searching Documentation and Demos with the Help Browser" on page 4-18.

- See "Specifying Colors for the Help Browser" on page 4-56 for information about changing the color of hyperlinks in the **Index**.

Searching Documentation and Demos with the Help Browser

- "Searching in the Help Browser" on page 4-18

- "Wildcards in Search (Partial Word Searching)" on page 4-23

- "Exact Phrases in Search" on page 4-23

- "Boolean Operators in Search" on page 4-24

- "Refining Your Search" on page 4-24

- "To Get Fewer Search Results" on page 4-25

- "To Get More Search Results" on page 4-25

Searching in the Help Browser

To look for a specific word or phrase in the documentation or demos, use the search field in the **Help Navigator**.
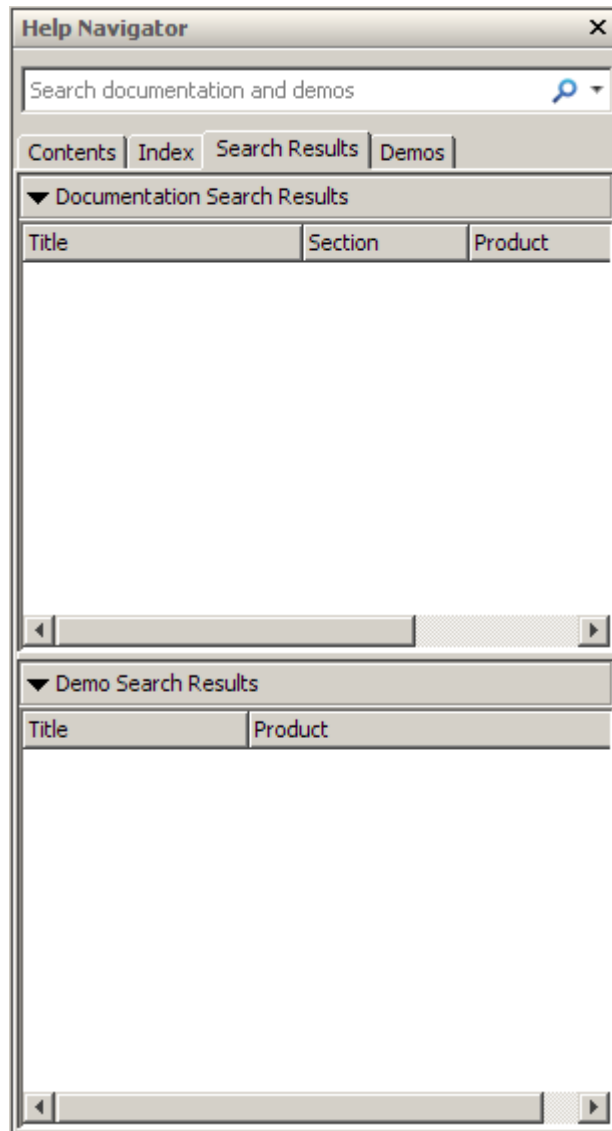
Follow these steps to use the Help browser search feature:

1 Specify which product documentation and demos to search by selecting the down arrow on the right side of the search field in the Help browser and select **Filter by Product**. For more information, see "Setting the Scope of Documentation — Product Filter" on page 4-47.

2 In the search field, type the word or words you want to find and press **Enter**. These are some ways to refine your search:

- Exact Phrase — Search for an exact phrase by enclosing words in quotation marks, for example, `"plot tools"`. For more information, see "Exact Phrases in Search" on page 4-23.

- Wildcard (Partial Word) — Search for variations of a word, also called partial word searching, by using the wildcard symbol (`*`) in place of letters in a word, for example, `plot* tool*`. For more information, see "Wildcards in Search (Partial Word Searching)" on page 4-23.

- Boolean Operators — Add the Boolean operators `AND`, `OR`, and `NOT` between search words to include or exclude words. By default, search assumes an `AND` between all search words and exact phrases. For more information, see "Boolean Operators in Search" on page 4-24.

- See also "Refining Your Search" on page 4-24.

You can reuse a search. To do so, select the down-arrow at the right side of the search field. From the resulting menu, select **Show Search History**. From the history, select an entry to rerun that search.

3 View the list of documents and demos containing all of the search words by clicking the **Search Results** tab. Two sets of results appear: **Documentation** and **Demo** results. The number of results for each appears in parentheses at the top of the listing. Both sets of results have a column that lists the product name. For documentation, another column lists the section where the search words appear. You might need to make the **Help Navigator** pane wider to see all of the columns.

4 Select an entry from the list of results. By default, the first documentation entry is automatically selected. If there are no documentation results, but there are demo results, the first demo entry is automatically selected.

The selected page appears in the display pane with all occurrences of the search words and exact phrases highlighted, using a different color for each

search word or phrase. Highlights remain until you browse to the next or previous page in the documentation, or until you click the Refresh button ⟳ in the toolbar.

For more results, you can look for the search words in bug reports, solutions, and notes on the MathWorks Web site. To do so, click the **Search Support Database on Web** link at the bottom of the **Search Results** pane.

5 To find a specified word within the page, use the **Find** tool, accessible from the Find button 🔍 on the display pane toolbar.

6 To better find the most relevant results, change the way the results are presented. By default, search results are ordered by relevance. For example, documentation, reference pages whose titles match the search words appear first, followed by any titles that contain all search words. Pages containing a single instance of each search word appear last. Here are the ways you can change the presentation of search results:

- Sort by column — Click a column heading to order the results by the information in that column. For example, click **Product** to group results by product name. Click **Title** to sort titles alphabetically. A triangular icon indicates the column on which you most recently sorted. Click the heading again to sort by the same column, but in reverse alphabetical order.

  After clicking a column heading, to return to seeing the results ordered by relevance, press **Enter** in the search field, which reruns the search.

- Reorder columns — To change the location of a column, drag its heading to a new position. For example, you can drag the **Product** column to the middle.

- Resize columns — To make columns wider or narrower, drag the separator bar between the column headings. Similarly, you can also make the Help Navigator (left pane) or the entire Help browser wider or narrower.
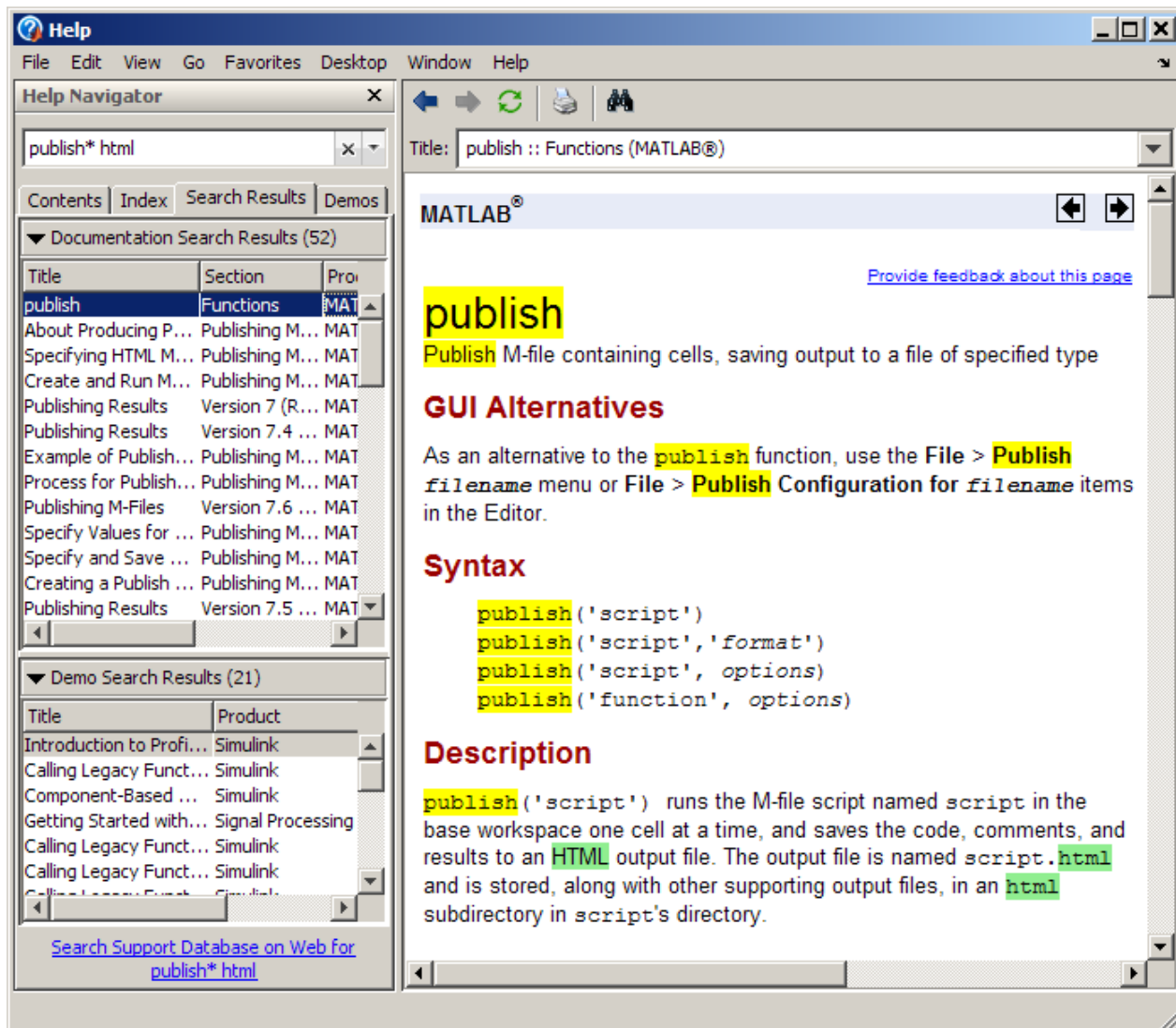
Function Alternative for Searching Documentation. From the Command Window, use docsearch to search for the specified words in the documentation and display results in the Help browser **Search Results** pane. For example, to find all pages that contain the word publish or its variations, such as publishing, published, and so on, and that contain the word html, run:

```
docsearch publish* html
```

To find all pages containing the exact phrase plot tools, run:

```
docsearch "plot tools"
```

For details, see the docsearch reference page.

Wildcards in Search (Partial Word Searching)

You can use the wildcard character (*) in place of letters or digits in your search words. For example, plot* finds various forms of the word *plot*, such as *plot*, *plots*, and *plotting*. When you enter p*t, it also finds those variations of *plot*, and it finds variations of other words that begin with *p* and end with *t*, such as *print* and *part*, among others.

You can use multiple wildcards in a word or words. For example, plot* tool* finds *plotting tools*, among other results. Adding more wildcards, as in p*t* tool*, also finds pages containing the words *path* and *toolbox*.

Wildcard Restrictions. You cannot use a wildcard with just one letter or digit, nor can you include wildcards within an exact phrase. You cannot begin a search word with a wildcard character. For example, these fail: p*, "plot* tools", plot *ool.

Exact Phrases in Search

To find an exact phrase, type quotation marks around the phrase. For example, "plot tools" finds only pages that include plot tools together, but does not find pages that include plot in one part of the page and tools in another part of the page. Specify an exact phrase to reduce the number of irrelevant results. For example, "plot tools" finds a dozen or so pages in MATLAB documentation, while plot tools finds an order of magnitude more, with nearly hundred results.

You can specify more than one exact phrase, such as `"plot tools"` `"figure palette"` to find pages that contain both `"plot tools"` and `"figure palette"`. You cannot use a wildcard within an exact phrase.

Boolean Operators in Search

The search automatically performs a Boolean `AND` for multiple words. In the example `publish* html`, it finds all pages that have the word `publish` or its variations, and the word `html`.

You can refine the search by including the Boolean operators `NOT`, `OR`, and `AND` between words. The operators must be in all capital letters and there must be a space before and after each operator. The search evaluates `NOT` operators first, `OR` operators second, and `AND` operators last.

Example Using Boolean Operators in Search. To find all pages that contain the words `plot` or its variations and the word `tools`, but not the words `time series`, type

```
plot* tools NOT time series
```

Refining Your Search

If you are not finding pages as you might expect, consider making changes to your search words based on how search works:

- Search ignores Insignificant words (a, an, the, of).

- Search is not case sensitive.

- Search finds letters and digits, but not symbols. To find a symbol, look for the word instead of the symbol, (for example, `plus` instead of +), use the **Index**, or see Operators and Special Characters in the MATLAB Functions — By Category listing. Another option is to search the PDF documentation, which supports searching for symbols. For details, see "Accessing the PDF Version of Documentation" on page 4-57.

- Search finds words in comments or code for M-file demos and Model demos. It finds comments in the M-file help for M-GUI demos. Search does not look in video demos.

- If you search for a function that is used in multiple products (called an overloaded function), the search results list includes reference pages for all those products. To identify the reference page you want, use the **Product** column in the search results.

To Get Fewer Search Results

If there are too many results for the search to be useful, try the following.

Problem	Try These Suggestions
Too many products	Specify which products to search—select the down arrow on the right side of the search field and select **Filter by Product**. For more information, see "Setting the Scope of Documentation — Product Filter" on page 4-47.
	Order results by product — click the **Product** column in **Search Results**. If you cannot see the column, make the pane wider.
Pages are not about search word, but just mention it	Try the **Index** pane to see only key entries for that word.
Too many irrelevant results	Type more than one word in the search field.
	Look for an exact phrase by enclosing words in quotations marks, such as `"plot tools"`.
	Use Boolean operators (in all capitals), for example, `printing figures NOT exporting`.
Topic is not relevant	Look at the **Section** column in **Search Results** before selecting a result in the list, which provides context for the result. If you cannot see the column, make the pane wider.
Want to look only within part of a product's documentation	For products like MATLAB, you might want to search only part of the documentation. There is no way to do this in the Help browser. However, you might be able to accomplish that via PDF search. For example, you can search the "MATLAB Getting Started Guide" PDF file, or the "MATLAB External Interfaces" PDF file. For details, see "Accessing the PDF Version of Documentation" on page 4-57.

To Get More Search Results

If you want more results, try the following.

Problem	Try These Suggestions
No results for the product	Be sure the product filter is set correctly. Select the down arrow on the right side of the search field and select **Filter by Product**. Ensure the products of interest are selected. For details, see "Setting the Scope of Documentation — Product Filter" on page 4-47.
No results appear, but you know the word should be there	Try variations of the search word by using a wildcard symbol (`*`). For example, search for `preference*` to find all pages that contain either the word `preference` or the word `preferences`.
Not enough information	Try looking in bug reports, solutions, and technical notes by clicking the **Search Support Database on Web** link at the bottom of the **Search Results** pane.
	If you are not running the most current version of MATLAB, try looking at the most current version of the documentation on the Web site. It might contain additional information. For more information, see "Accessing Documentation for All MathWorks Products" on page 4-64.

See Also.

- "Finding Files and Content Within Files in Any Directory" on page 5-82, which describes the Find Files tool you use to look for files and content within files, such as comments in M-files or code fragments.
- "Finding Functions Using the Function Browser" on page 3-41.

Labeling Pages as Favorites in the Help Browser

Favorites are bookmarks for pages in the Help browser documentation and M-file type demos.

Adding Favorites

To designate the displayed page as a favorite (that is, to bookmark it),

1 Select **Favorites > Add to Favorites**.

The Favorites Editor dialog box opens.

2 Accept the defaults and click **Save**, or make changes to the entries:

   a Use the **Label** provided, or change it to another term.

   b Maintain the **Category** as `Help Browser Favorites`.

   c For **Icon**, keep the default `Help icon`, or choose another.

   Do *not* change the entry for **Callback**.

A favorite is implemented as a MATLAB shortcut, so the dialog box is the same as for the Shortcut Editor.

Favorites from previous releases are not migrated to a new release.

Viewing Pages Labeled as Favorites

Select the **Favorites** menu to view the list of pages you previously designated as favorites (bookmarked pages). Select an entry and that page appears in the display pane.

Organizing Favorites

You can rename, remove, and reorder the list of favorites. Select **Favorites > Organize Favorites**. For more information, click **Help** in the Organize Favorites dialog box.

Viewing Documentation in the Help Browser

About the Display Pane

After finding documentation with the Help Navigator, view the documentation in the display pane. The following illustration shows the Help Navigator closed to provide a larger area for viewing the information.

Help

File   Edit   View   Go   Favorites   Desktop   Window   Help

Title: Assistance While Entering Statements :: Running Functions - Command Window and History (MATLAB®)

Completing Statements in the Command Window — Tab Completion

MATLAB helps you complete the names of known items as you type them in the Command Window so that you can avoid spelling mistakes and do not have to spend time looking up the information in other tools. These are the items for which MATLAB can complete the names:

- Function or model on the search path or in the current directory
- Filename or directory
- Variable, including structures, in the current workspace
- Handle Graphics® property for figure in the current workspace

Type the first few characters of the item name and then press the **Tab** key. To use tab completion, you must have the tab completion preference for the Command Window selected. For details, see Keyboard Preferences.

Tab completion is also available in the Editor, but there are some slight differences in usage. See Tab Completion in the Editor.

These examples demonstrate how to use tab completion in the Command Window:

- Basic Example — Unique Completion
- Multiple Possible Completions
- Tab Completion for Directories and Filenames
- Tab Completion for Structures
- Tab Completion for Properties

Basic Example — Unique Completion

Browsing to Other Pages in the Help Browser

Use the arrow buttons on a page and in the toolbar to go to other pages:

- View the next page in a document by clicking the Next page button ▶ at the top or bottom of the page. View the previous page in a document by clicking the Previous page button ◀ at the top or bottom of the page. These arrows allow you to move forward or backward within a single document. The arrows at the bottom of the page are labeled with the title of the page they go to.

- View previously shown pages by clicking the Back button ◀ and subsequently the Forward button ▶ in the display pane toolbar. These buttons work like the forward and back buttons of popular Web browsers. You also can go back or forward by right-clicking a page and selecting **Back** or **Forward** from the context menu.

Following Links in Help Pages

Click links on the displayed page to go to a related topic in the documentation that contains more information about the subject. Links are underlined and blue. Already visited links are purple. Links to Web addresses open in your system browser.

Finding Text in a Help Page

To find a string or exact phrase in the currently displayed page:

1 Click the Find button 🔍. In the resulting Find dialog box, type the string or phrase that you are looking for. For a string, you can type the partial word, such as `preference` to find all occurrences of `preference` *and* `preferences`.

2 To specify options, use the check boxes in the Find dialog box.

3 Click **Find Next**.

   The search begins at the current cursor position and the page scrolls to the first occurrence of the phrase in the page and highlights it.

4 To find more occurrences in the page, click **Find Next** or **Find Previous** in the Find dialog box, or use the keyboard shortcuts **F3** and **Shift+F3**.

MATLAB beeps when a search for **Find Next** reaches the end of the page, or when a search for **Find Previous** reaches the top of the page. If you have the **Wrap around** option selected, it continues searching after beeping.

You can change the selection in the **Look in** field of the Find dialog box to search for the specified text in other MATLAB desktop tools.

If you want to search through all the documentation instead of just one page, see "Searching Documentation and Demos with the Help Browser" on page 4-18.

Copying Information from a Help Page

To copy information from the display pane, such as code in an example, first select the information. Then right-click and select **Copy** from the context menu. You then can paste the information into another tool, such as the Command Window or Editor, or into another application, such as a word processing application.

Running Code Examples — Evaluating a Selection in a Help Page

To run code examples that appear in the documentation, select the code in the display pane. Then right-click and select **Evaluate Selection** from the context menu. The statements execute in the Command Window. The following illustrations show code in the `sort` reference page being evaluated.

Help _ □ ✕

File   Edit   View   Go   Favorites   Desktop   Window   Help

Title: sort :: Functions (MATLAB®)

Example 2

This example sorts each column of a matrix in descending order.

```
A = [ 3   7   5
      6   8   3
      0   4   2 ];

sort(A,1,'descend')
```

Evaluate Selection	F9
Open Selection	
Help on Selection	
Back	Alt+Left
Forward	Alt+Right
Refresh	F5
Copy	Ctrl+C
Page Source	
Print...	Ctrl+P

```
ans =
      6   8   5
      3   7   3
      0   4   2
```

This is equivalent to
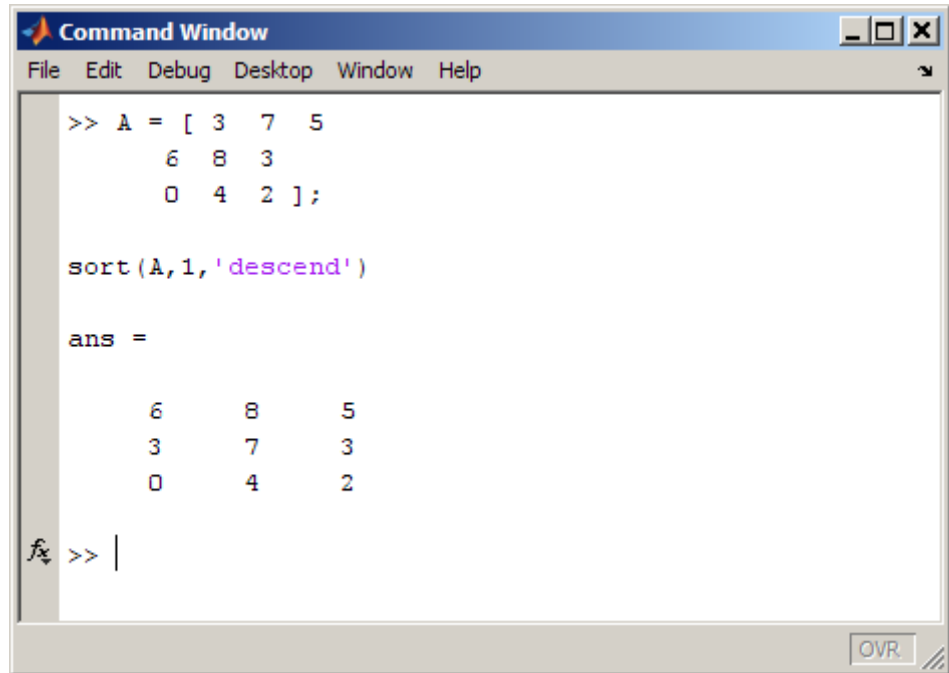
```
sort(A,'descend')

ans =
      6   8   5
      3   7   3
      0   4   2
```

See Also

issorted, max, mean, median, min, sortrows, unique

Opening a Selection in a Help Page

In a page in the display pane, select the name of a file that is provided with MATLAB, such as an M-file function. Then right-click and select **Open Selection** from the context menu. The file opens in MATLAB. For example, an M-file opens in the Editor.

The following illustrations show opening the `sortrows` M-file from the `issorted` reference page.

```
                                                            Help                    _ □ ×

File   Edit   View   Go   Favorites   Desktop   Window   Help

Title:  issorted :: Functions (MATLAB®)

Example 2 — Using issorted on a matrix

        A = magic(5)
        A =
            17    24     1     8    15
            23     5     7    14    16
             4     6    13    20    22
            10    12    19    21     3
            11    18    25     2     9

        issorted(A, 'rows')
        ans =
             0

        B = sortrows
        B =                    Evaluate Selection  F9

             4     6         Open Selection
            10    12
            11    18         Help on Selection
            17    24
            23     5         Back                Alt+Left
                             Forward             Alt+Right
        issorted(B)          Refresh             F5
        ans =
             1               Copy                Ctrl+C

Example 3 — Using                  Page Source

                             Print...            Ctrl+P
```

Getting Help for the Selected Function While Viewing a Help Page

While viewing a page in the Help browser display pane, select the name of a function. Then right-click and from the context menu, select **Help on Selection**. The reference page for that function opens in the Help browser.

The following illustrations show getting help for the `repmat` function from a page in the Data Analysis documentation.

Help

File   Edit   View   Go   Favorites   Desktop   Window   Help

Title: Descriptive Statistics :: Data Processing (MATLAB®)

To find the minimum value in the entire `count` matrix, you can reshape this 24-by-3 matrix into a 72-by-1 column vector by using the syntax `count(:)`. Then, to find the minimum value in the single column, you can use the following syntax:

```
min(count(:))

ans =
       7
```

Example 2 — Subtracting the Mean

You can subtract the mean from each column of the matrix by using the following syntax:

```
% Get the size of the count matrix
[n,p] = size(count)
% Compute the mean of each column
mu = mean(count)
% Create a matrix of mean values by
% replicating the mu vector for n rows
MeanMat = repmat(mu,n,1)
% Subtract the c                  lement
% in that column
x = count - Mean
```

Evaluate Selection	F9
Open Selection	
Help on Selection	
Back	Alt+Left
Forward	Alt+Right
Refresh	F5
Copy	Ctrl+C
Page Source	
Print...	Ctrl+P

Note Subtracting   called *detrending*. For more
information about re   t line from the data, see Detrending
Data.

▲ Back to Top

Example: Using M   ics

The Data Statistics dialog box helps you calculate and plot descriptive statistics with the data. This example shows how to use MATLAB Data Statistics to calculate and plot statistics for a

4-37

```
┌─────────────────────────────────────────────────────────────────────┐
│ ⦾ Help                                            _ □ ×  │
├─────────────────────────────────────────────────────────────────────┤
│ File   Edit   View   Go   Favorites   Desktop   Window   Help      ↘ │
├─────────────────────────────────────────────────────────────────────┤
│ ▣  │ ◀  ➡  ⟳  │ 🖨  │ 🔍                                           │
├─────────────────────────────────────────────────────────────────────┤
│ Title: │ repmat :: Functions (MATLAB®)                         │  ▼  │
├─────────────────────────────────────────────────────────────────────┤
```

MATLAB®

Provide feedback about this page

repmat
Replicate and tile array

Syntax

```
B = repmat(A,m,n)
B = repmat(A,[m n])
B = repmat(A,[m n p...])
```

Description

`B = repmat(A,m,n)` creates a large matrix `B` consisting of an m-by-n tiling of copies of `A`. The size of `B` is `[size(A,1)*m, (size(A,2)*n]`. The statement `repmat(A,n)` creates an n-by-n tiling.

`B = repmat(A,[m n])` accomplishes the same result as `repmat(A,m,n)`.

`B = repmat(A,[m n p...])` produces a multidimensional array `B` composed of copies of `A`. The size of `B` is `[size(A,1)*m, size(A,2)*n, size(A,3)*p, ...]`.

Remarks

`repmat(A,m,n)`, when `A` is a scalar, produces an m-by-n matrix filled with `A`'s value and having `A`'s `class`. For certain values, you can achieve the same results using other functions, as shown by the following examples:

- `repmat(NaN,m,n)` returns the same result as `NaN(m,n)`.

Viewing the Help Page Source (HTML)

To view the HTML source for the currently displayed page, select
View > Page Source. A read-only HTML version of the page appears in a
separate window. You can copy selections from the HTML source and paste
them into other tools like the Editor or Command Window, or into other
applications.

Viewing the Help Page Location

If you want to send someone a link to a page in the Help browser, you can get
the URL (link) information using the Help Page Location feature. With the
page displayed in the Help browser, select **View > Page Location**. The Help
Page Location dialog box appears, providing the full path to the documentation
file for access in the Help browser or on the MathWorks Web site.
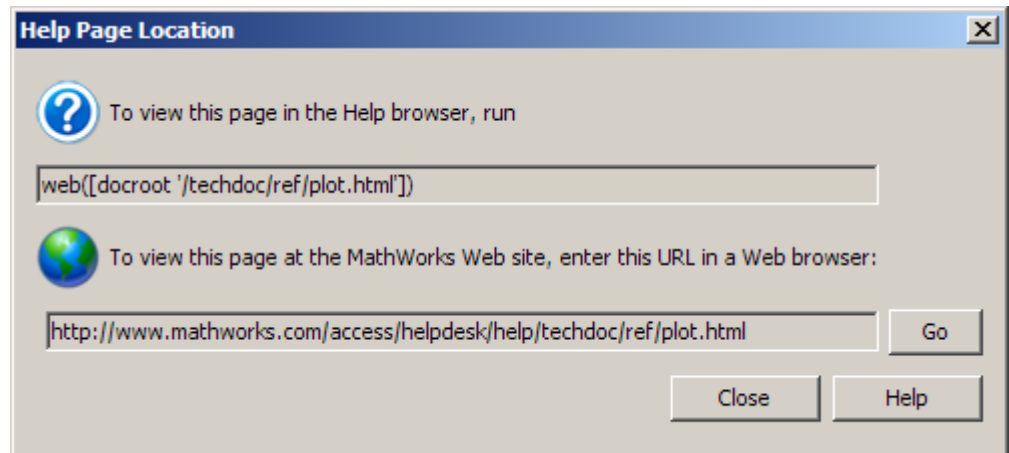
You can copy the information from this window into an e-mail message or
other tool to facilitate communication with other users or The MathWorks
about a page in the documentation. For example, if you find a page of
documentation that you know would be useful to a colleague running
MATLAB, send them the link so they can view the page in the Help browser.

To view the same documentation page on The MathWorks Web site, click the
Go button. This is useful if you do not see the information you are looking for
on the page in the Help browser and you are *not* running the most current

version of MATLAB. The documentation for the most current version is on the Web site and might include more information than the documentation for your version. Note, though, that the documentation on the Web site might refer to features that are not part of your earlier-version product. For more information, see "Accessing Documentation for All MathWorks Products" on page 4-64.

Viewing and Running Demos

In this section...
"About Demos" on page 4-41
"Using Demos" on page 4-42
"Adding Your Own Demos" on page 4-46

About Demos

MATLAB and related products include demos that you can access from the Help browser **Demos** pane.

There are four types of demos:

- M-file — Demos that tell a step-by-step story, including source code, commentary, and output. They are published from M-file scripts to HTML output using the Editor. The first comment line of the demo M-file begins with two comment symbols (%%), and similarly, two comment symbols (%%) create a cell for each step. The MATLAB Graphics Square Wave from Sine Waves demo is an M-file type demo.

- M-GUI — Standalone tools for exploring a feature. An example is the MATLAB Graphics Vibrating Logo demo.

- Model — Simulink block diagrams. An example is the Engine Timing Simulation demo.

- Video — Movies that highlight key features in a tool. They play in your system browser and require the Adobe® Flash Player plug-in. Some also require an Internet connection. An example is the MATLAB Desktop and Command Window demo.

The MATLAB code and Simulink blocks used in the demos (except videos) are available for you to view and copy for use in your own applications.

See also Examples for each product in the **Contents** pane. These examples are similar to demos but are integrated in the documentation.

Using Demos

To access demos for the products you have installed, follow these steps:

1 Click the **Demos** tab in the **Help Navigator**.

Other ways to access demos are from the:

- **Start** button
- demo function
- **Help** menu for some tools
- Search Results in the Help browser

2 In the Demos pane of the Help browser, click the expander (+) for a product area to list the products or categories that have demos. Then click the expander (+) for a product or product category to list its demos.

The display pane lists all demos for that product or product area, and for each demo, shows the type of demo along with a thumbnail image that represents output from the demo.

3 Select a specific demo. Information about the demo appears in the display pane.

Help

File   Edit   View   Go   Favorites   Desktop   Window   Help

Help Navigator   ✕

Search documentation and demos   🔍 ▼

Contents | Index | Search Results | **Demos**

- 🟢 Getting Started with Demos
- 🔺 MATLAB
  - 📁 Getting Started
  - 📁 Mathematics
  - 📁 Graphics
    - 2-D Plots
    - 3-D Plots
    - 3-D Surface Plots
    - Line Plotting
    - Axes Properties
    - Axes Aspect Ratio
    - Vibrating Logo
    - Lorenz Attractor Animation
    - Visualizing Sound
    - Earth's Topography
    - Images and Matrices
    - Examples of Images and
    - Viewing a Penny
    - Square Wave from Sine W
    - Functions of Complex Var
    - Interactive Plot Creation w
    - Linked Plots and Data Bru
  - 📁 3-D Visualization
  - 📁 Programming

Title: Square Wave from Sine Waves

Open xfourier.m in the Editor   **Run in the Command Window**

Square Wave from Sine Waves

The Fourier series expansion for a square-wave is made up of a sum of odd harmonics. We show this graphically using MATLAB®.

We start by forming a time vector running from 0 to 10 in steps of 0.1, and take the sine of all the points. Let's plot this fundamental frequency.

```
t = 0:.1:10;
y = sin(t);
plot(t,y);
```

4 You can then view and run the demo, with specific options depending on the type of demo:

- M-file demos — Click the **Open *filename* in the Editor** link at the top left. This opens the M-file in the Editor. From the Editor, run the demo step by step by selecting **Cell > Evaluate Current Cell and Advance**.

  You can also click **Run in the Command Window**, and then follow the instructions that appear in the Command Window. You might need to scroll up to see all of the instructions.

  See also "Running Demos and Base Workspace Variables" on page 4-45.

- M-GUI demos — Click the **Open filename in the Editor** link at the top left. This opens the M-file in the Editor.

  Click the **Run this demo** link at the top right to start the GUI. Then follow the instructions in the GUI to proceed through the demo.

- Model demos — Click **Open this model** to open the block diagram.

- Video demos — Click the **Run this demo** link in the top right to play the video. Video demos run in your system browser and require the Adobe Flash Player plug-in. Some video demos also require an Internet connection.

When you double-click a demo name in the **Help Navigator** pane, for M-file and Model demos, the demo file opens; for M-GUI and Video demos, the demo runs.

The following example shows the results of running the MATLAB Graphics Square Wave from Sine Waves demo (xfourier). In the demo, MATLAB generates a series of plots, culminating in the final one shown here.
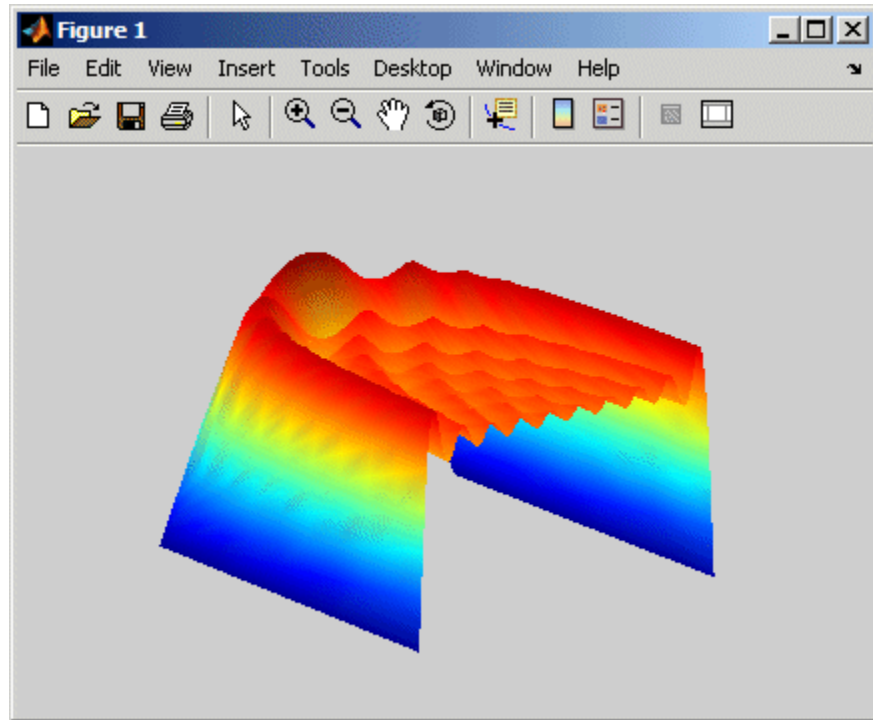
Running Demos and Base Workspace Variables

M-file demos run as scripts. Their variables are created in the base workspace for MATLAB. If you have variables in the base workspace when you run an M-file demo, and the demo uses an identical variable name, there could be problems due to variable name conflicts. For example, a variable of yours might be overwritten by the demo. The demo's variables remain in the base workspace until you clear them or quit MATLAB.

Function Alternative for Using Demos

To open the **Demos** pane in the Help browser, type demo in the Command Window. You can go directly to the demos for a specific product. For example

```
demo toolbox signal
```

opens the **Demos** listing for the Signal Processing Toolbox product.

To run an M-GUI demo, type the demo name in the Command Window. For example, type

```
vibes
```

to run the MATLAB Graphics demo showing an animated L-shaped membrane.

To run an M-file demo step by step from the Command Window, type echodemo followed by the demo name. For example, run

```
echodemo xfourier
```

Typing the demo name for an M-file demo runs the demo, but not step by step.

Typing the name of a model demo opens the block diagram.

Adding Your Own Demos

You can add your own demos so they appear in the **Demos** pane. For details, see "Adding Demos for Your Own Toolboxes" on page 4-82.

Setting Preferences for the Help Browser

In this section...

Setting the Scope of Documentation — Product Filter

If you have multiple MathWorks products, you can set the product filter to limit the product documentation and demos that the Help browser and Function Browser display and use for searching the documentation. To use the product filter, perform these steps:

1 Select **File > Preferences > Help**. Another way to access the product filter is by selecting **Filter by Product** from the drop-down list in the search field of the Help browser, as shown in the following illustration.
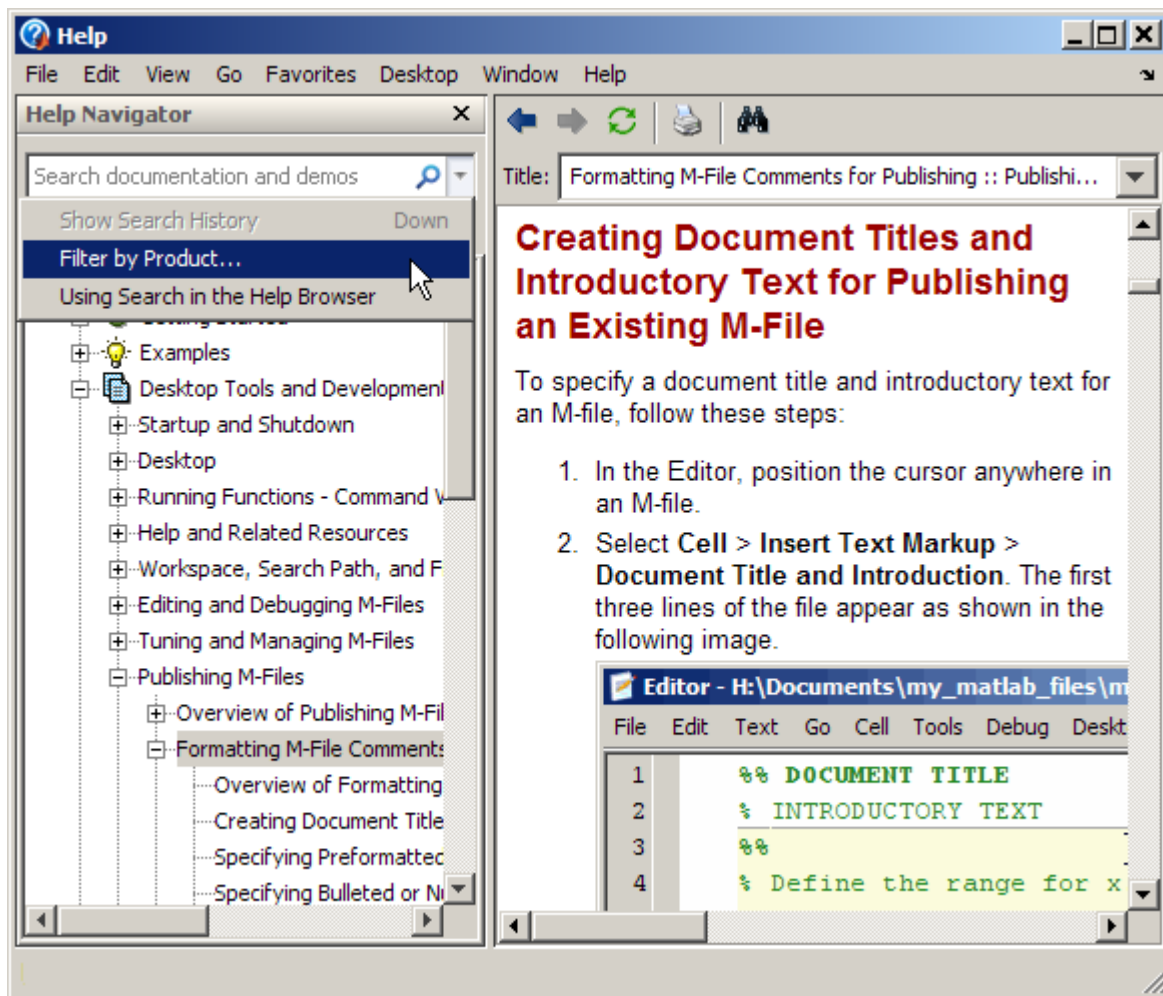
The Preferences dialog box opens to the Help Preferences pane.
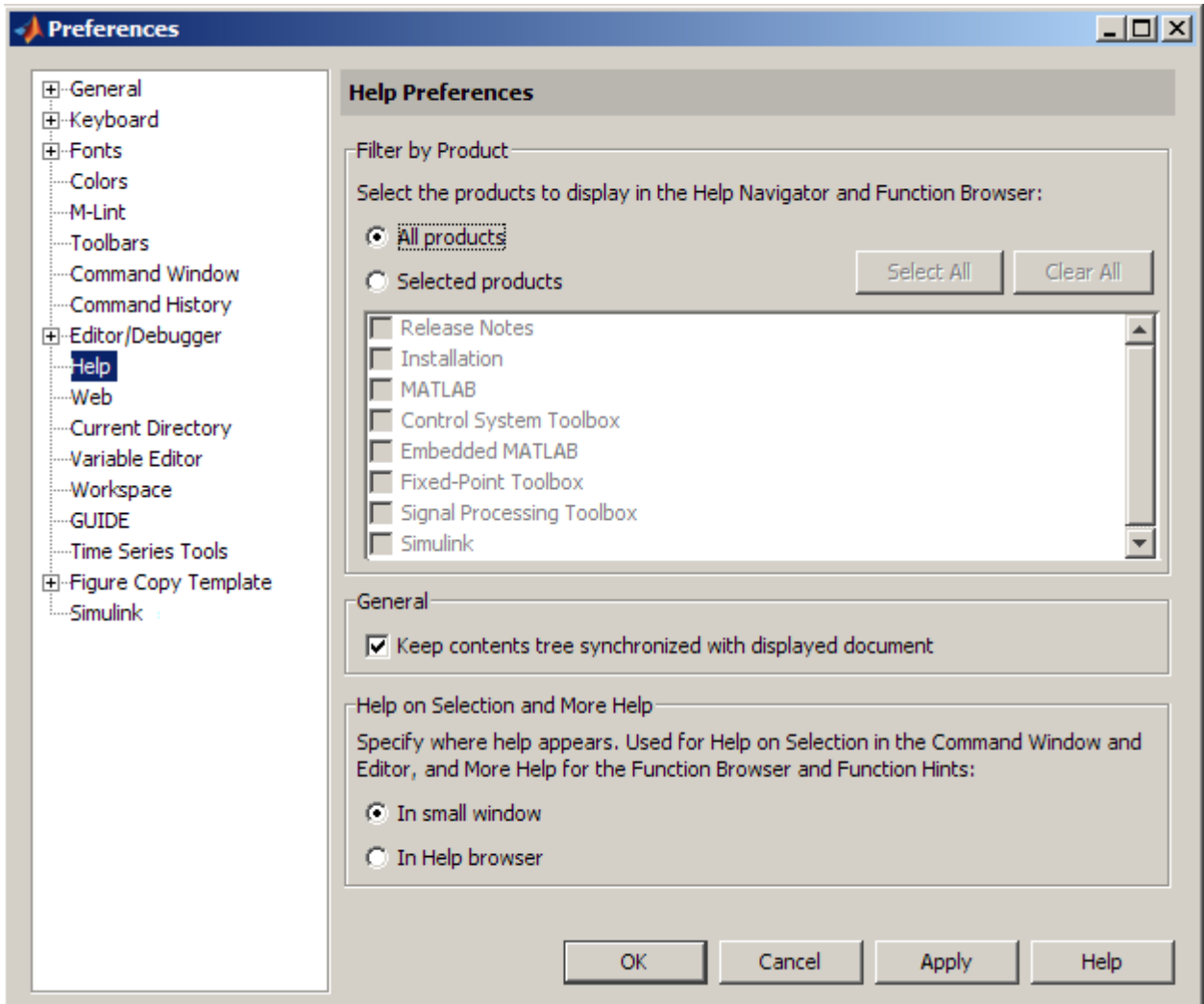
2 Narrow or expand the scope using **Filter by Product** in Help Preferences. Notes:

- The list of all products includes only those products you are licensed to use.

- The `Release Notes` entry refers to the general Release Notes overview document for all products in a release. It does not apply to the product-specific release notes, which are considered part of a product's documentation.

3 Click **OK** in the Preferences dialog box.

The Help browser now only uses the documentation for those products you specified. The Function Browser provides access to function reference pages for only the products you specified.

Example Using the Product Filter

If you have many products installed and want to perform a search, but know the information you are seeking is in MATLAB or the Communications Toolbox product, use the product filter to narrow your search. Perform these steps

1 Open Help Preferences by selecting **File > Preferences > Help**.

2 In the **Filter by Product** group box:

  a Click **Clear All**.

  b Select **Selected products**.

  c In the list of products, select `MATLAB` and `Control Systems Toolbox`.

PDF Reader — Specifying Its Location

If you want to view the PDF version of the documentation, the Help browser needs to know the location of your PDF reader (for example, the Adobe Acrobat product).

On UNIX[14] platforms, the default PDF reader is Acrobat® and MATLAB automatically determines its location. If a different command starts your PDF reader, you specify it using this preference. Select **File > Preferences > Help**, and enter the full path in the **PDF reader** field or use the Browse for Folder (**...**) button to navigate your file system to select it.

---

14. UNIX is a registered trademark of The Open Group in the United States and other countries.
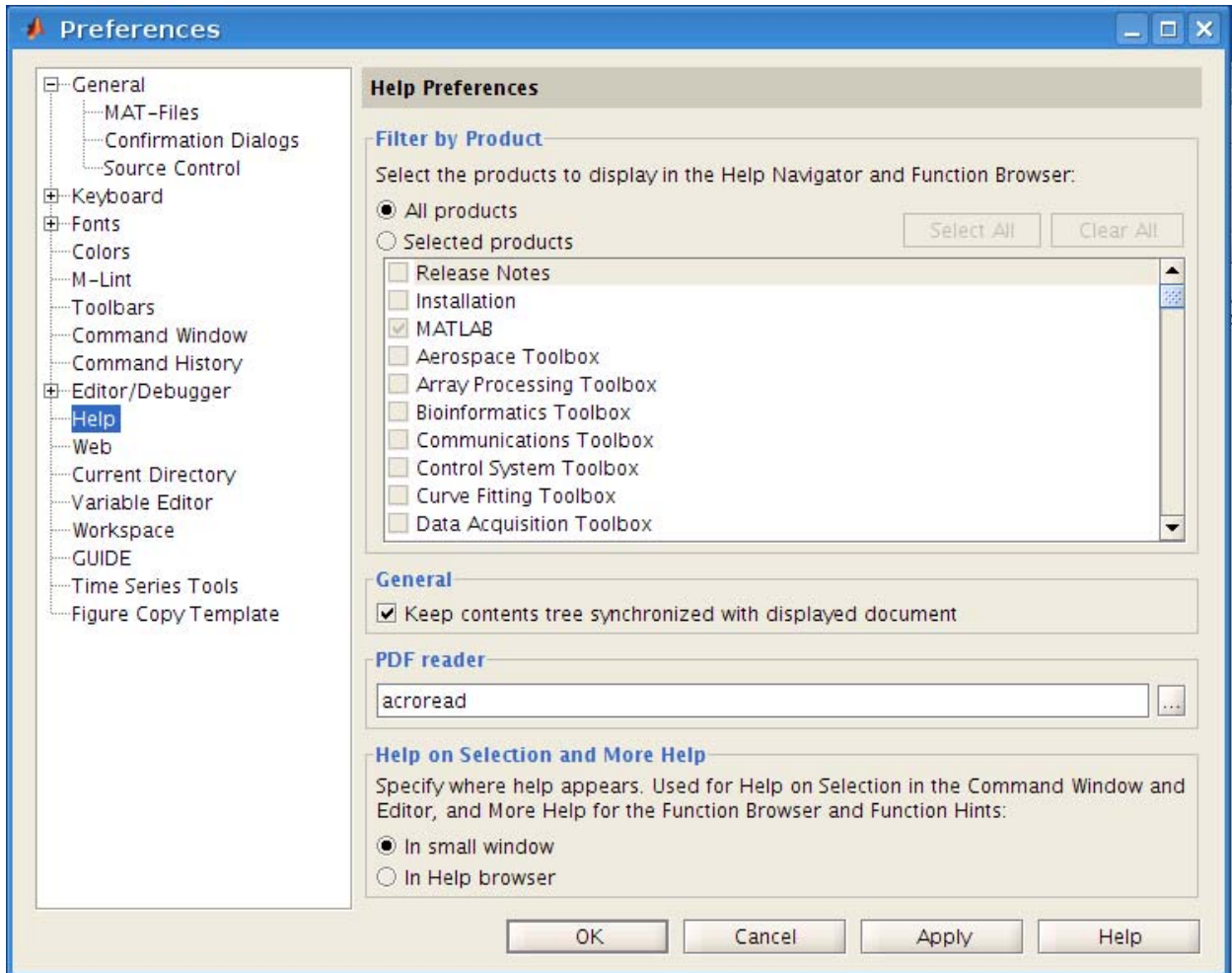
Note The **PDF reader** preference is only for UNIX platforms and does not appear in the preferences pane for other platforms. For example, on Microsoft Windows platforms, MATLAB reads the PDF reader location from the registry.

General — Keeping Contents Synchronized

By default, the displayed page in the Help browser is synchronized with the listings in the **Contents** or **Demos** panes. For more information about this feature, see "Synchronizing the Contents Listing and Demos Listing with the Display Pane" on page 4-14.

To turn synchronization off, select **File > Preferences > Help**. In the **General** group box, clear the check box for **Keep contents tree synchronized with displayed document**. Select the check box to turn synchronization back on.

Help on Selection Window — Specifying Where It Displays

When you work in the Editor or the Command Window, you can display the reference page for a function by using the Help on Selection feature. You can specify where the reference page displays by using Help preferences. Select **File > Preferences > Help**, and then for **Help on Selection and More Help**, select one of the following options:

- **In pop-up window** — The reference page displays in a small help window that appears with the Command Window or Editor.

- **In Help browser** — The reference page displays in the Help browser.

  For details about the feature, see "Getting Help for the Selected Function in the Command Window or Editor" on page 3-39.

The **Help on Selection** preference applies only to the help on selection feature in the Command Window and Editor. This preference also affects where the reference page appears when you click **More Help** in the Function Browser pop-up window, and in the function hints syntax listing.

This preference does not apply when you get help for a selected function in the Help browser or Current Directory browser; the reference page always opens in the Help browser.

Font and Color Preferences for the Help Browser

Set colors and fonts for the Help browser the same way you do for other desktop tools. Note that by default, the Help browser display pane uses a custom font. This section describes the process of making color and font changes for the Help browser:

- "Specifying the Font Name, Style, and Size for the Help Browser" on page 4-53

- "Specifying Colors for the Help Browser" on page 4-56

Specifying the Font Name, Style, and Size for the Help Browser

You specify fonts for the Help Navigator (left pane) and the Help display pane (right pane) separately:

- The Help Navigator uses the desktop text font by default. You can change the font family, style (bold or italic), and size that the Help Navigator uses.

- The Help browser display pane uses a custom font (Sans Serif, 10 pt.) by default. In MATLAB, tools that display HTML are referred to as HTML Proportional Text tools in the fonts preferences dialog box, and they all use the same custom font. However, when you change the font characteristics, not all of them impact the Help browser:

  - Changes you make to the font size affect all text and code in the display pane.

  - Changes you make to the font family affect text in the display pane, but do not affect code in the display pane.

  - Changes you make to the font style (bold or italic) have no impact in the display pane, but might impact other tools that use the HTML proportional text font preference.

For more information, see "Setting Fonts Preferences for Desktop Tools" on page 2-121.

4-53

Example — Specifying Fonts for the Help Browser. This example describes how to specify Microsoft Comic Sans® MS, italic, 14 point font for the HTML Proportional Text tools, which includes the Help display pane.

The Help Navigator uses the desktop text font by default. The Help display pane, because it is considered to be an HTML Proportional Text tool, uses SanSerif, Plain, 10 pt. font by default.

1 Select **File > Preferences > Fonts > Custom**.

2 From the **Desktop tools** list, select `HTML Proportional Text`.

3 For **Font to Use**, select **Custom**, and then specify the font characteristics:

- Family, for example, `Comic Sans`
- Size in points, for example, `14`
- Set the style as italic

After you make a selection, the **Sample** area shows how the font looks.

4 Click **OK**.

The font in the Help browser display pane uses the new settings. The font in the MATLAB Web Browser also uses the new settings because the MATLAB Web Browser is an HTML Proportional Text tool. The change to italic style has no effect. The code uses a monospace font, not the HTML Proportional Text font and is not affected by the change to Comic Sans MS.
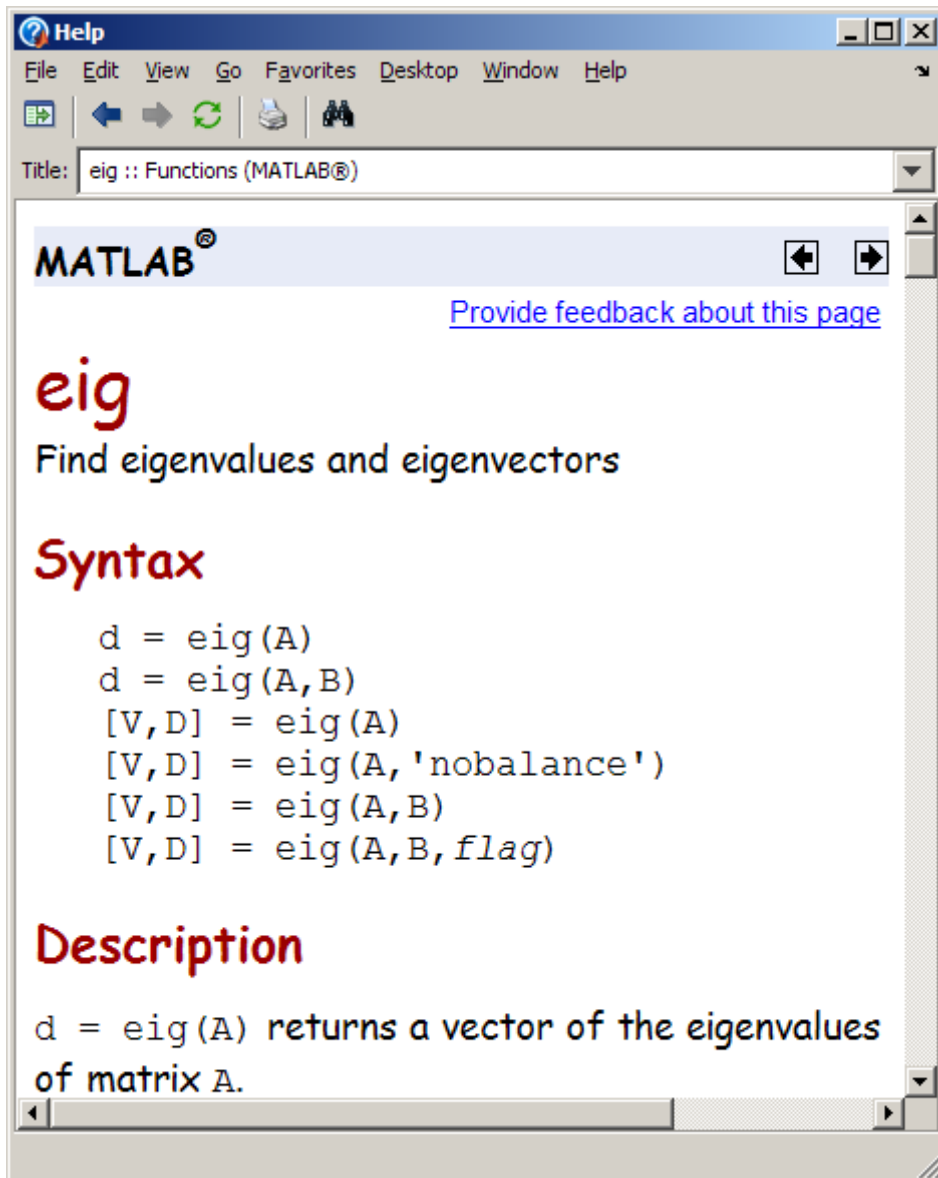
Specifying Colors for the Help Browser

You can specify the background and text color used in the **Help Navigator**. Use the same method as you would to specify the background color for any desktop tool—for more information, see "Setting Colors Preferences for Desktop Tools" on page 2-131.

If the background color preference for your desktop tools is a dark color, you might not be able to see index entries in the Help Navigator because they are links, for which the default color is blue. To see the links more easily, change the **Hyperlink** color preference to a light or other contrasting color—for more information, see "Other Colors" on page 2-137.

You cannot specify colors for the Help browser display pane.

Getting Printed Documentation

In this section...
"About Printed Manuals" on page 4-57
"Printing a Page from the Help Browser" on page 4-57
"Accessing the PDF Version of Documentation" on page 4-57

About Printed Manuals

Generally, printed manuals are not provided for most products and tools. The printed manuals typically contain less information, and are also sometimes less current than the online documentation. If you want to purchase printed documentation, see the online store at the MathWorks Web site at http://www.mathworks.com.

Printing a Page from the Help Browser

To print the page currently shown in the Help browser, select **File > Print**, or click the Print button 🖨 in the display pane toolbar. The Print dialog box appears.

Select **All** in the Print dialog box to print the entire page shown in the display pane. Specifying a range of pages, for example, 1 to 3, prints the first three pages of the page currently shown in the display pane.

Complete the dialog box and click **OK** to print.

Accessing the PDF Version of Documentation

If you need to print more than a few pages of documentation, or if you want the pages to appear as if they came from a printed book, print the PDF version of the documentation. The PDF documentation typically contains a subset of the online, HTML documentation that appears in the Help browser.

PDF documentation is shown and printed using your PDF reader, usually an Adobe Acrobat product. In the PDF document, use links from the table of contents, index, or within the document to go directly to the page of interest within that document.

> **Note** The Help browser accesses PDF documentation from the MathWorks Web site. Therefore, you need Internet access to view or print PDF documentation.

Follow these instructions to access PDF documentation:

1 In the Help browser, click the **Contents** tab and select a product, for example, the MATLAB product.

The roadmap page opens for that product, providing links to key documentation for that product.

2 Near the end of the roadmap page, listed under **Printing the Documentation Set**, are links for printing the documentation. Select the link for the item you want to print.

The selected document is accessed from the MathWorks Web site. Your PDF reader opens, displaying the PDF document.

If you are using a UNIX[15] platform and cannot open the PDF documentation, check the Help preferences. See "PDF Reader — Specifying Its Location" on page 4-50 for more information.

3 To print the documentation, select **Print** from the **File** menu in your PDF reader.

---

15. UNIX is a registered trademark of The Open Group in the United States and other countries.

Help Functions

In this section...

About Help Functions

There are several help functions that provide forms of help different than the Help browser documentation, or provide alternative ways to access the Help browser information.

Summary Table of Help Functions

Function	Description
dbtype	Displays specified M-file with line numbers. If you want to see only the input and output arguments for a function, use dbtype function 1, which displays the first line of the M-file.
demo	Displays the **Demos** pane in the Help browser, from which you can access demonstrations for the products you have installed. With an argument, runs the specified demo.
doc	Displays in the Help browser, the reference page for the specified function, block, or property. Usually more extensive than results for the help function, the reference page provides syntax, a description, examples, illustrations, and links to related functions.
docsearch	Run the Help browser search feature for the specified term.
help	Displays M-file help (a description and syntax) in the Command Window for the specified function. For MDL-files, displays a description of the model.
helpbrowser	Opens the Help browser, the MATLAB interface for accessing documentation.

Function	Description
helpdesk	Opens the Help browser. In previous releases, helpdesk displayed the Help Desk, which was the precursor to the Help browser. This function will be removed in a future release.
helpwin	Displays in the Help browser a list of all functions, and provides access to M-file help for the functions.
lookfor	Displays in the Command Window a list and brief description of all functions whose brief description includes the specified keyword.
web	Opens the specified URL in the specified browser. Use web in your own M-files to display HTML documentation you create for your work.
whatsnew	Displays the Release Notes in the Help browser.

Viewing Function Reference Pages — the doc Function

To view the reference page for a function, block, or property in the Help browser, use doc. For example, type

```
doc format
```

to view the reference page for the format function.

Overloaded Functions with the doc Function

When a function name is used in multiple products, it is said to be an overloaded function. The doc function displays the reference page for the first function on the search path having that name, and displays a hyperlinked list of the overloaded functions in the Command Window.

For example, using the default search path

```
doc set
```

displays the reference page for the MATLAB set function in the Help browser. The Command Window displays a hyperlinked list of the set functions located in other directories, such as

```
database/set
```

which is the set function for the Database Toolbox™ product. Click a link to go to that set reference page.

To directly get the reference page for an overloaded function, specify the name of the directory containing the function you want the reference page for, followed by the function name. For example, to display the reference page for the set function in theDatabase Toolbox product, type

```
doc database/set
```

Some products have more than one function with the same name. For example, MATLAB includes a built-in get function in the graphics directory and a get function in the MATLAB serial directory (for serial port functions). Type

```
doc get
```

The reference page for the MATLAB graphics built-in get function appears, and the Command Window lists overloaded functions in other products. But the list does not include any overloaded functions in the same product. Therefore, get in the MATLAB serial directory is not listed as an overloaded function. Type

```
doc ('get (serial)')
```

to display the reference page for the get function located in the MATLAB serial directory.

Getting Help in the Command Window — the help Function

To quickly view a brief description and syntax for a function in the Command Window, use the help function. For example, typing

```
help bar
```

displays a description and syntax for the bar function in the Command Window. This is called the M-file help. For other arguments you can supply, see the reference page for help.

Note M-file help displayed in the Command Window uses all uppercase characters for the function and variable names to distinguish them from the rest of the text. When typing function names, however, use lowercase characters. Some functions, especially those for interfacing to the Sun Microsystems Java language, use mixed case; the M-file help accurately reflects that, and when you type them, use mixed case.

If you need more information than the `help` function provides, use the `doc` function, which displays the reference page in the Help browser. It can include color, images, links, and more extensive examples than the M-file help. For example, typing

```
doc bar
```

displays the reference page for the `bar` function in the Help browser.

Overloaded Functions with the help Function

When a function name is used in multiple products, it is said to be an overloaded function. The `help` function displays M-file help for the first function on the search path having that name, and then displays a hyperlinked list of the overloaded functions.

For example, using the default search path

```
help set
```

displays M-file help for the MATLAB `set` function, and displays a hyperlinked list of the `set` functions residing in other directories, such as

```
database/set
```

which is the `set` function for the Database Toolbox product. Click a link to display the M-file help for that `set` function.

To directly get help for an overloaded function, specify the name of the directory containing the function you want help for, followed by the function name. For example, to get help for the `set` function in the Database Toolbox product, type

```
help database/set
```

Creating M-File Help for Your Own M-Files

You can create M-file help for your own M-files and access it using the `help` command. For more information, see "Help for the Files You and Other Users Create" on page 4-65.

Getting Help for Model Files

Use the `help` function with an MDL filename to display the complete description for the model file. For example, run

```
help 4_dap.mdl
```

and MATLAB displays the description of the Simulink F-14 Digital Autopilot High Angle of Attack Model, as defined in the **Model > Properties > Description**.

```
Multirate digital pitch loop control for F-14 control design
demonstration.
```

If the Simulink product is installed, you do not need to include the `.mdl` extension.

You can see the same description in the Current Directory browser **Details** pane.

Accessing Documentation for All MathWorks Products

The Help browser provides access to documentation for MathWorks products installed on your system. To view the online documentation for all MathWorks products, use the MathWorks Web site at `http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml`.

These are some uses for the Web site version of the documentation:

- Access documentation for products you do *not* have installed.

- Access documentation for the most current version. If you know you are not running the most current version of MATLAB, the documentation on the Web site might include more information than is provided in your version. However, the documentation on the Web site might refer to features that are not part of your earlier-version product.

- Access documentation for a prior version of some products (Release 13 with Service Pack 2). Note that the release notes on this page include release notes for multiple prior versions. For example, you can find information about the MATLAB Version 7.0 (Release 13) new features and changes.

- Access documentation via your system Web browser, such as when you are not running MATLAB, or if you prefer your system Web browser instead of the Help browser.

- PDF documentation is available only on the Web site. All of the information available in the PDF is available in the Help browser, but PDF is sometimes preferable, especially for printing documentation.

For any page you view in the Help browser, you can determine the URL for that page in the documentation on the Web site. See "Viewing the Help Page Location" on page 4-39.

Help for the Files You and Other Users Create

In this section...
"About Help for User-Created Files" on page 4-65
"Help for M-Files Created by Users" on page 4-65
"Contents Files for Your Own M-File Directories" on page 4-66
"Help for User-Created Classes" on page 4-67
"Adding Your Own Help Files and Demos in the Help Browser" on page 4-72
"Adding HTML Help Files for Your Own Toolbox" on page 4-72
"Adding Demos for Your Own Toolboxes" on page 4-82
"Validation Errors for info.xml Files" on page 4-88

About Help for User-Created Files

You can provide help for the files you create using MathWorks products. The help reminds you about a function, but also aids other users who might access your files. The type of help to provide depends primarily on what you are providing and the number of users you are providing the files to.

- For a few M-files that only you or a few others use, providing help in the M-file is probably sufficient.

- If you maintain many M-files, you can use Contents files to help you and other users utilize the help.

- For MATLAB classes you create, you and other users can access the M-file help in the Help browser.

- If you provide your files to many other users, you might want to provide HTML documentation and demos that users can access in the MATLAB Help browser.

Help for M-Files Created by Users

The help system in MATLAB, like MATLAB itself, is highly extensible. You can write help descriptions for your own M-files and toolboxes using the same self-documenting method that MATLAB M-files and toolboxes use. This is

helpful if you share your files with other users. Similarly, if you use M-files created by other users, you will benefit if they provide M-file help in the files.

Create self-documenting online help for your own M-files by entering text on one or more contiguous comment lines, beginning with the second line of the file (first line if it is a script). For example, the function `soundspeed.m` begins with

```
function c=soundspeed(s,t,p)
% soundspeed computes the speed of sound in water
% where c is the speed of sound in water in m/s

t = 0:.1:35;
```

When you execute `help soundspeed`, MATLAB displays

```
soundspeed computes the speed of sound in water
where c is the speed of sound in water in m/s
```

These lines are the first block of contiguous comment lines. After the first contiguous comment lines, enter an executable statement or blank line, which effectively ends the help section. Any later comments in the M-file do not appear when you type `help` for the function.

The first comment line in any M-file (the H1 line) is special because the `Contents.m` file uses it to provide a brief summary of the function's purpose. The H1 line should contain the function name and a brief description of the function. For the `soundspeed` example, the H1 line is

```
% soundspeed computes speed of sound in water
```

Use the "Generating a Summary View of the Help Components in M-Files" on page 7-8 to help you create and manage M-file help for your own files.

Contents Files for Your Own M-File Directories

A `Contents.m` file is provided for each M-file directory included with the MATLAB product. With an up-to-date `Contents.m` file, running `help toolboxname` displays the functions in the M-file directory, along with brief help (the H1 line) for each function. If you create directories in which to store your own M-files, it is a good practice to create and maintain `Contents.m` files

for them. Use the "Displaying and Updating a Report on the Contents of a Directory" on page 7-12 to help you create and maintain your own `Contents.m` files.

Help for User-Created Classes

If you create your own MATLAB classes and provide help (comments) in the class definition M-file, you can view help for the class in the Help browser by using the `doc` function. To use the feature, you must provide help in a comment line directly following the classdef statement, or in the comment line directly following the constructor method for the class. Optionally, provide additional help comments in the classdef file for other methods, and for properties and events.

When you run

```
doc classname
```

the resulting HTML page provides a convenient summary of the details, properties, methods, and events for the class, and includes links to descriptions as well as to help for any super classes. You can open the classdef M-file in the Editor from the help page by clicking the **View code for *classname*** link at the top of the page in the Help browser.

To go directly to help for a method, run

```
doc classname.methodname
```

Similarly, to go directly to the help for properties and events, use `doc` with the dot notation, providing the property name or event name after the dot.

When you create an instance of a class and open the object in the Variable Editor, you can access the HTML help for the class by clicking the class name link below the toolbar. This provides the same information as when you run `doc classname`.

Example of Help for a User-Created Class

This example shows help for a user-created class M-file, `sads.m`, which is provided with MATLAB documentation. Run the following statements to use the example.

```
% Change the current directory to where the example file is located.
cd(fullfile(matlabroot,'help','techdoc','matlab_env', 'examples'))
% View help for the file in the Help browser.
doc sads
% For more information, follow links. Or go directly to help, e.g. for the steer method.
doc sads.steer
% To see the help comments in the class M-file, sads.m, click the link in help, or run:
open(fullfile(matlabroot,'help','techdoc','matlab_env', 'examples','sads.m'))
% Create an instance of the sads class.
loadparameters
sensorArray=sads(Data, Wavelength,SampleRate,Spacing,Name);
% Another way to see help for sads is by clicking the classname link in the Variable Editor.
% Open sensorArray in the Variable Editor.
openvar sensorArray
```

The following illustration shows the help for sads, displayed in the Help
browser.

Help

File   Edit   View   Go   Favorites   Desktop   Window   Help

Help Navigator   ✕

Search documentation and demos   🔍 ▾

Contents | Index | Search Results | Demos

⊞ 📙 Release Notes
⊞ 📙 Installation
⊞ 📙 MATLAB
⊞ 📙 Signal Processing Toolbox
⊞ 📘 Simulink
⊞ 📘 Signal Processing Blockset

Title: M-File Help: sads   ▾

M-File Help: sads   View code for sads   Default Topics

sads

    Sensor Array Data Set Class

Class Details

Sealed   false
Construct on load   false

Constructor Summary

sads   SADS Create sensor array data set

Property Summary

Data	Data - Sampled sensor data
SampleRate	SampleRate - Sample rate (Hz)
Spacing	Spacing - Spacing of array (m)
Name	Name - Sensor array test run name
Wavelength	
c	c - Speed of wave in medium (m/s)
NumSensors	NumSensors - Number of sensors
NumSamples	NumSamples - Number of samples

Method Summary

steer   STEER Steer array electronically by angle theta (in degrees),

The following illustration shows more information for the `steer` method, which you can view by clicking the `steer` link under "Method Summary" in the `sads` help page, or by running `doc sads.steer`.
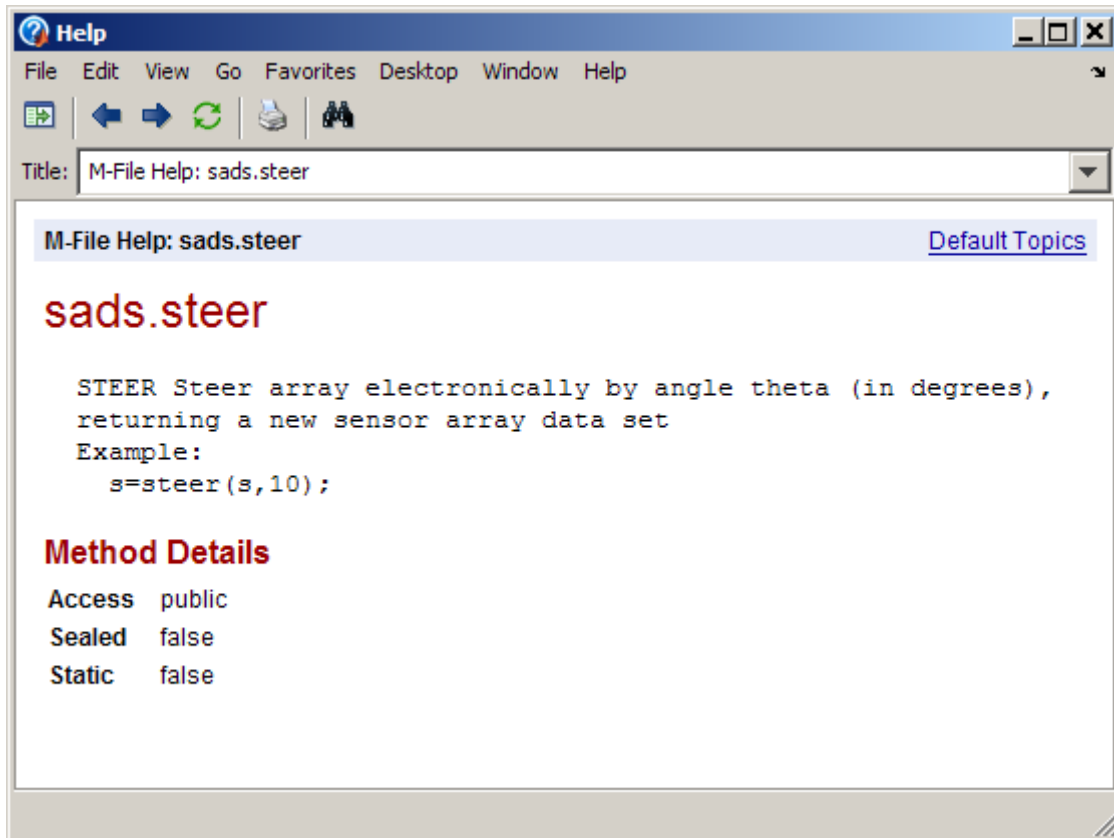
The following illustration shows the `sads` M-file opened in the Editor, which you can view by clicking the **View code for sads** link at the top of the `sads` help page.

The following illustration shows an instance of the `sads` object, `sensorArray`, in the Variable Editor. You can click the **sads** link in the Variable Editor to view the `sads` help in the Help browser.
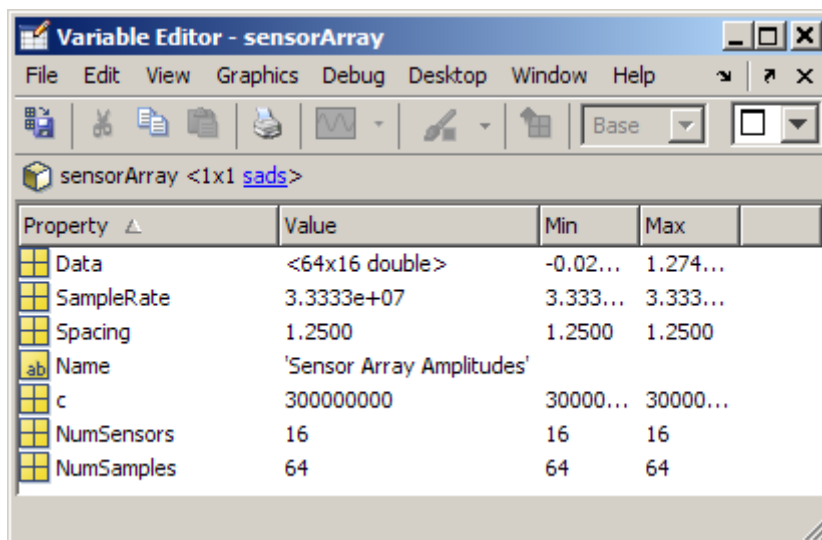
Adding Your Own Help Files and Demos in the Help Browser

You can add your own HTML help files and demos so that they appear in the Help browser, alongside MathWorks help files and demos. Similarly, you can include your own toolboxes on the MATLAB **Start** button. For more information, see "Adding Your Own Toolboxes to the Start Button" on page 2-72.

For tools that are useful when preparing a collection of M-file, see the "Using M-File Reports" on page 7-2.

Adding HTML Help Files for Your Own Toolbox

- "Overview of Providing HTML Help Files for Your Own Toolbox" on page 4-73

- "Adding HTML Help Files for Your Own Toolbox to the Help Browser" on page 4-74

- "Description of info.xml File for Adding HTML Help Files for Your Toolbox" on page 4-77

- "Creating a Searchable Database" on page 4-81

Overview of Providing HTML Help Files for Your Own Toolbox

If you offer a collection of files for use with MATLAB and related products (referred to here as a toolbox) to other users, you can provide HTML help files for your toolbox and provide access to them from the MATLAB Help browser. With this form of help, you can provide content that is richer than M-file help for functions that you create, allowing you to include figures, illustrations, screen captures and equations. For background information about the Help browser and other forms of help supported by MATLAB, see Chapter 4, "Help, Demos, and Related Resources" .

You can use the MATLAB Editor or any tool to create your HTML help files. To use syntax highlighting and indenting features for HTML and related files in the Editor, select **File > Preferences > Editor/Debugger > Language**, and choose HTML.

To view an HTML help file that you created in a browser in MATLAB, use the web function. For example, to display my_help_file in the MATLAB Web browser, run:

```
web('I:\my_matlab_files\my_help_file.html')
```

Using the web function with the -helpbrowser options displays the HTML page in the Help browser:

```
web('fullpath/filename.html', '-helpbrowser')
```

You can include a link to the HTML help for a function within the M-file help for the function. M-files provided with MATLAB include such a link to the HTML reference page for the function. To create this link in the M-file help, use a matlabcolon (matlab:) statement in conjunction with the web statement. The doc function does *not* support user-created HTML files.

You can add the HTML help files for your own toolbox, as well as provide a search database for the help files so that you and others can access the HTML help from the Help browser using the contents listing and search features. For details, see "Adding HTML Help Files for Your Own Toolbox to the Help Browser" on page 4-74 and "Creating a Searchable Database" on page 4-81.

Adding HTML Help Files for Your Own Toolbox to the Help Browser

MATLAB determines what information to display in the Help browser using `info.xml` files that reside on the search path or in the current directory. You create an `info.xml` file for your toolbox, which includes a pointer to your HTML help files.

You provide the `info.xml` to users, along with the other help files for your toolbox. With the `info.xml` file on their search path or in their current directory, users can access your HTML help files from the Help browser **Contents** pane.

If you also want your toolbox to appear on the **Start** button, put the Help browser and **Start** button information into a single `info.xml` file (see "Adding Your Own Toolboxes to the Start Button" on page 2-72).

Example files accompany the following instructions, which add HTML help files for the SpecSim (Example) Toolbox to the Help browser. This example assumes that you want to add HTML help files to the Help browser, but do not want to add any entries for the help files on the **Start** button. Use the example `helpinfo.xml` file as a basis for creating an `info.xml` file to add HTML help files for your own toolbox.

1 Follow these steps to create the `info.xml` file:

   a Choose a directory for which you have write access and add it to the search path. (The directory cannot be the current directory when you add it to the path). For this example, create `specsim_example`.

   b Copy *matlabroot*/help/techdoc/matlab_env/examples/helpinfo.xml to the directory you just created (for the example, copy to `specsim_example`).

   c Rename `specsim_example/helpinfo.xml` to `info.xml`.

2 Follow these steps to establish the Help contents listing and HTML files in a valid location:

   a In the directory containing your newly-created `info.xml` file, create a subdirectory for your help files. For the example, in `specsim_example`, create a directory called `specsim_example_helpfiles`.

    b Copy *matlabroot*/help/techdoc/matlab_env/examples/helptoc.xml
        to the directory you just created. For the example, copy helptoc.xml
        to specsim_example_helpfiles.

    c Put your HTML help files in the same directory. For the
        example, copy the set of specsim_..._example.html files
        from *matlabroot*/help/techdoc/matlab_env/examples to
        specsim_example_helpfiles.

3 To see the example HTML help files in the Help browser:

    a Select **Start > Desktop Tools > View Start Button Configuration
        Files**.

    b In the resulting Start Button Configuration Files dialog box, click
        Refresh Start Button. This refreshes info.xml files for MATLAB.

    c Open the Help browser to see the entries for the SpecSim (Example)
        Toolbox. Be sure to disable the product filter, or select the SpecSim
        (Example) Toolbox in the product filter, if the product filter is enabled.
        Use **Preferences > Help** to access the product filter.
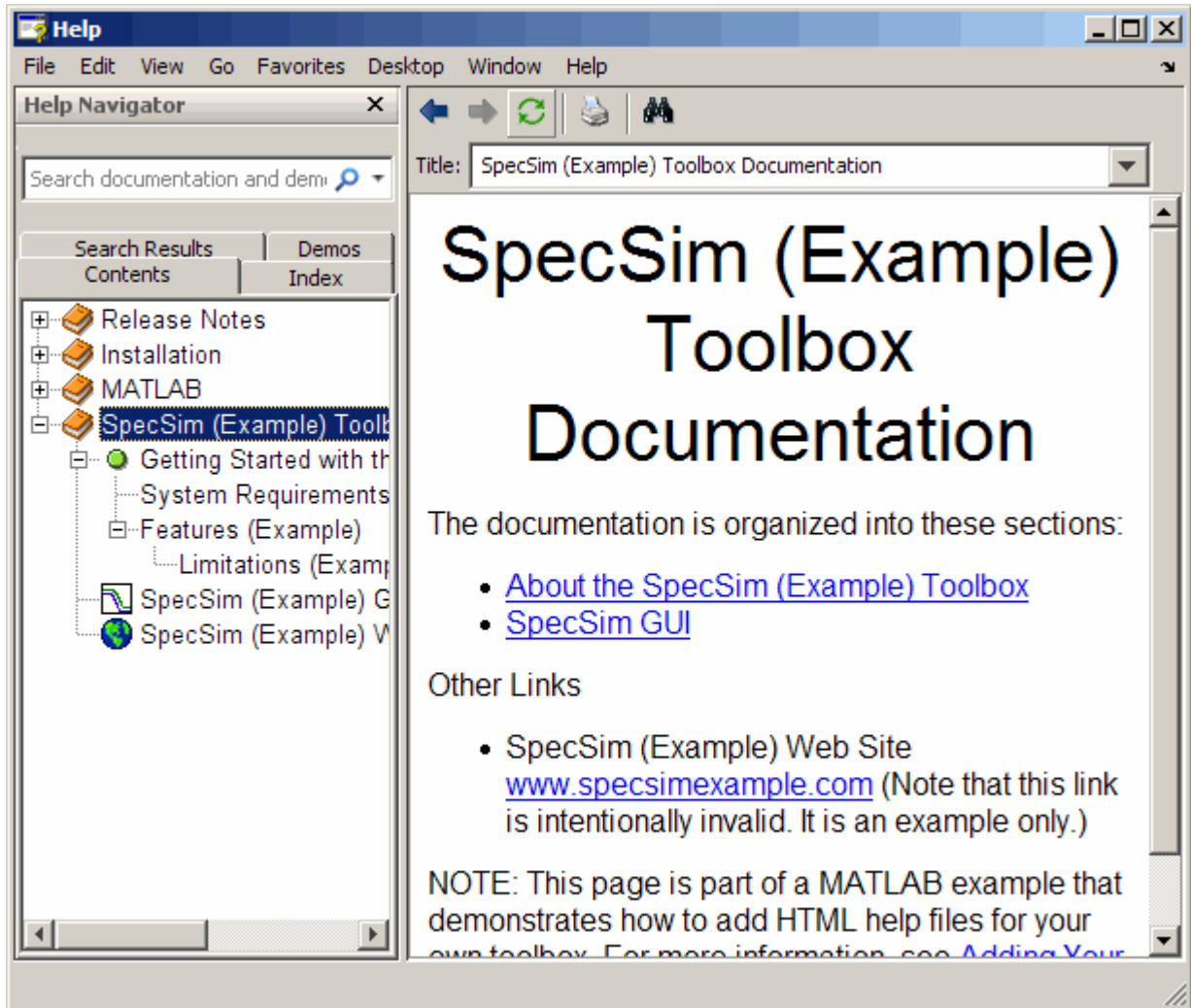
4 In the Editor, open and modify the `info.xml` and
`helptoc.xml` files for your toolbox. Save the files. For
the example, the files are `specsim_example/info.xml` and
`specsim_example/specsim_example_helpfiles/helptoc.xml`. You can
also include:

- A `helpindex.xml` file to provide entries for your toolbox in the Help browser **Index** pane.

- A `helpsearch` directory and search database files, allowing users to include the content of your toolbox HTML help files when they perform searches using the Help browser.

For details, see "Description of info.xml File for Adding HTML Help Files for Your Toolbox" on page 4-77.

5 Refresh the Help browser by selecting **Start > Desktop Tools > View Start Button Configuration Files**, and then click **Refresh Start** Button in the dialog box.

Description of info.xml File for Adding HTML Help Files for Your Toolbox

- "Content of the info.xml File" on page 4-77
- "Help Files for <help_location> in the info.xml File" on page 4-80
- "More Examples of info.xml Files" on page 4-80

Content of the info.xml File. The content of the `info.xml` file adds the HTML help files to the Help browser. This is the content of the `info.xml` file for the SpecSim (Example) Toolbox, with an explanation of each entry in the table that follows:

```
<productinfo
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="optional">
<?xml-stylesheet type="text/xsl" href="optional"?>

<matlabrelease>R2008b</matlabrelease>
<name>SpecSim (Example)</name>
<type>toolbox</type>
<icon></icon>
<help_location>specsim_example_helpfiles</help_location>

</productinfo>
```

XML Tag	Description	Value in Example	Notes
`<matlabrelease>`	Release of MATLAB.	`R2008b`	Not currently used.
`<name>`	Title of toolbox.	`SpecSim (Example)`	The name of your toolbox as it will appear in the Help browser **Contents** pane.
`<type>`	Determines where the toolbox is located in the Help browser **Contents** pane, appends a type to the entry, and assigns a colored book icon.	`toolbox`	Types are presented in the order shown here: `matlab`, `toolbox`, `simulink`, `blockset`, `links_targets`, `other`. Within a type, items are presented alphabetically. For the example, SpecSim (Example) Toolbox will appear with other toolboxes and with an orange book icon, which is the standard for toolboxes.
`<icon>`	Icon for help for your toolbox in the **Start** button.	None	Use an icon for your help files. In this example, the **Start** button is not used. See also the `<list>` entry, in this table.

XML Tag	Description	Value in Example	Notes
`<help_location>`	Location of help files	`specsim_example_helpfiles`	Location of `helptoc.xml` and HTML help files you provide for your toolbox. Specify a relative path, that is, relative to the location of the `info.xml` file. If you provide HTML help files for multiple toolboxes, each directory name must be unique.
`<list>` `<listitem>` ...	Entries for **Start** button	None	If you also want your toolbox to appear in the **Start** button, add at least one `listitem`. For details, see "Adding Your Own Toolboxes to the Start Button" on page 2-72.

Help Files for <help_location> in the info.xml File. Put these help files into the directory you specified for help_location in the info.xml file.

File Name	Required or Optional	Description
helptoc.xml	Required	The file that the Help browser displays in the **Contents** pane, with hierarchical entries for your toolbox. It references your HTML help files. A helptoc.xml file is provided for the example, which illustrates the structure. If you include icons, provide them in a directory within matlabroot/help and specify their path using $help/..., as shown in the example. You can also use $toolbox /... to reference icons in a *matlabroot*/toolbox directory; see *matlabroot*/toolbox/matlab/icons for icons provided with MATLAB. You can also specify a full path.
helpindex.xml	Optional	Provides entries for the **Index** pane in the Help browser. View the helpindex.xml file for any MathWorks product that you installed to see how you can create a similar file for your own toolbox. The top-level entries in the helpindex.xml file must be in case-sensitive alphabetical order.
helpsearch directory containing search database files	Optional	Allows the content of your HTML help files to be included in Help browser searches, providing entries for the **Search Results** in the Help browser. For details on creating this directory and its contents, see "Creating a Searchable Database" on page 4-81.
HTML help files	Required	HTML files containing the help for your toolbox. At least one of these is referenced by helptoc.xml.

More Examples of info.xml Files. For additional examples, see the info.xml files for MathWorks products:

1 Select **Start > Desktop Tools > View Start Button Configuration Files** files.

2 From the resulting Start Button Configuration Files dialog box, select a product.

3 Click **Open** to view its info.xml file.

Creating a Searchable Database

The builddocsearchdb function creates a searchable database that allows you to search HTML help files that you created, and to display the search results in the **Search Results** tab of the Help browser. This example assumes that you have created an info.xml file for the SpecSim (Example) files, and that the help_location for the SpecSim (Example) files is I:\my_matlab_files\specsim_example_helpfiles.

1 Create an info.xml that has a valid entry for <help_location>, which contains HTML files. Add to the search path the directory containing the info.xml file. (Do not add it when the directory is the current directory.)

2 Create a searchable database by running builddocsearchdb('*full_path to_help_location*'). For the example, run:

```
builddocsearchdb ('I:\my_matlab_files\specsim_example\specsim_example_helpfiles')
```

This creates a directory named helpsearch in *help_location_dir_name* (for the example, I:\my_matlab_files\specsim_example_helpfiles), and creates three files in helpsearch that are used when searching the documentation. MATLAB creates the deletable and segments file names each time you run builddocsearchdb. The name of the third file varies, but always uses the cfs extension.

3 From **File > Preferences > Help**, set the product filter to search only your toolbox. Use the **Search for:** field in the Help browser to search for any words in the HTML help files that you provided in *help_location_dir_name*. This example shows a search of SpecSim HTML help files for the term specsim gui.
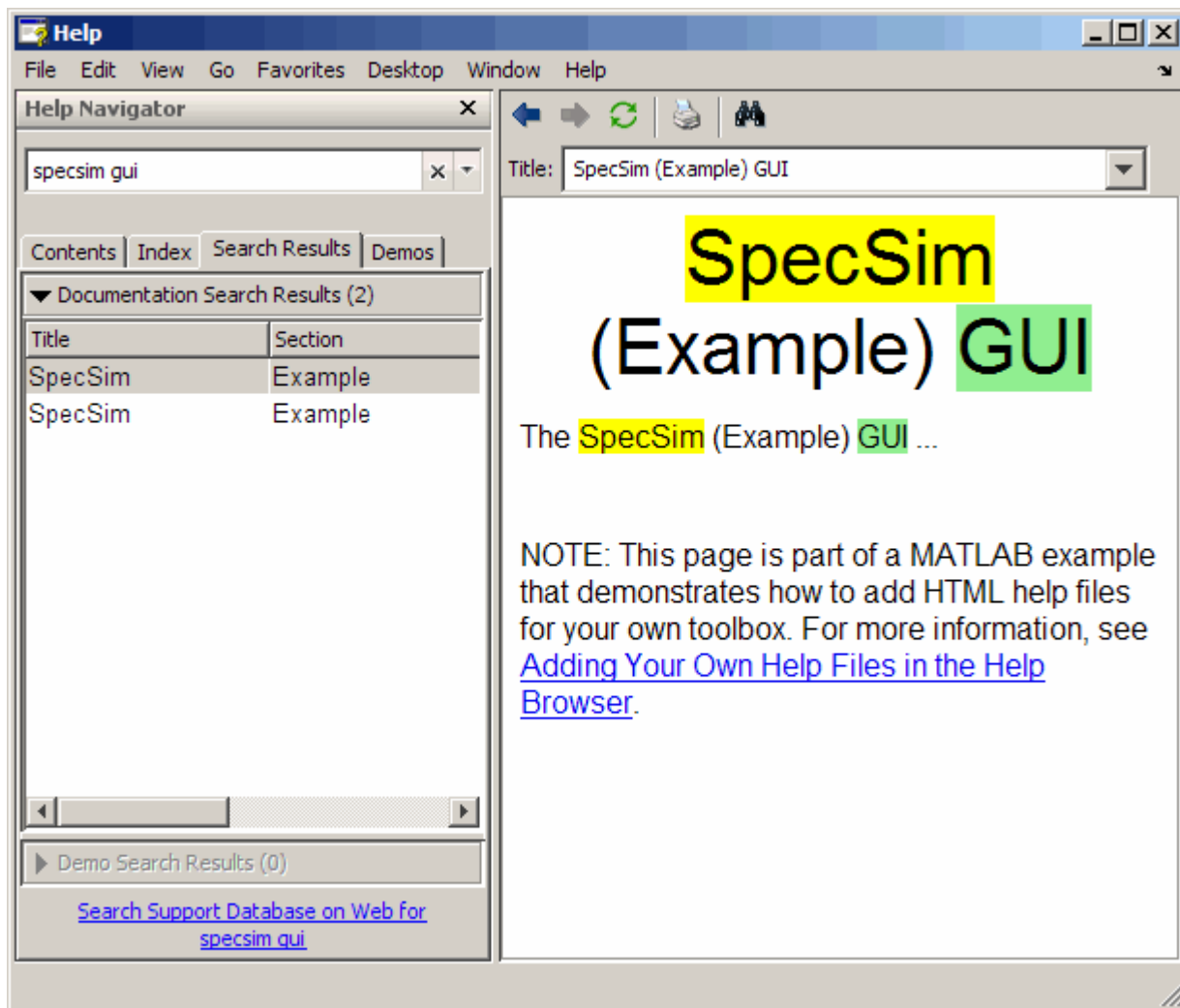
Adding Demos for Your Own Toolboxes

- "About Adding Demos for Your Own Toolboxes" on page 4-83
- "Procedure for Adding Demos for Your Own Toolboxes" on page 4-83

About Adding Demos for Your Own Toolboxes

If you offer a collection of files for use with MATLAB and related products (referred to here as a toolbox) to other users, you can include demo files for your toolbox and provide access to them from the MATLAB Help browser **Demos** pane. (For background information about demos, see "Viewing and Running Demos" on page 4-41.)

You can include demos and related files of any file type. The cell-publishing feature, available in the Editor, provides a convenient way to create demos as files that run in MATLAB and also display as HTML files. For more information, see "Overview of Publishing M-Files" on page 8-2.

MATLAB determines what demos to display in the Help browser using `demos.xml` files that reside on the search path or in the current directory. To add demos for your toolbox, you create a `demos.xml` file that includes pointers to your demo files. You provide the `demos.xml` file to users, along with the demos files for your toolbox. With the files on their search path, users can access your demos from the Help browser **Demos** pane or by using the `demo` function with the full path to a demo type.
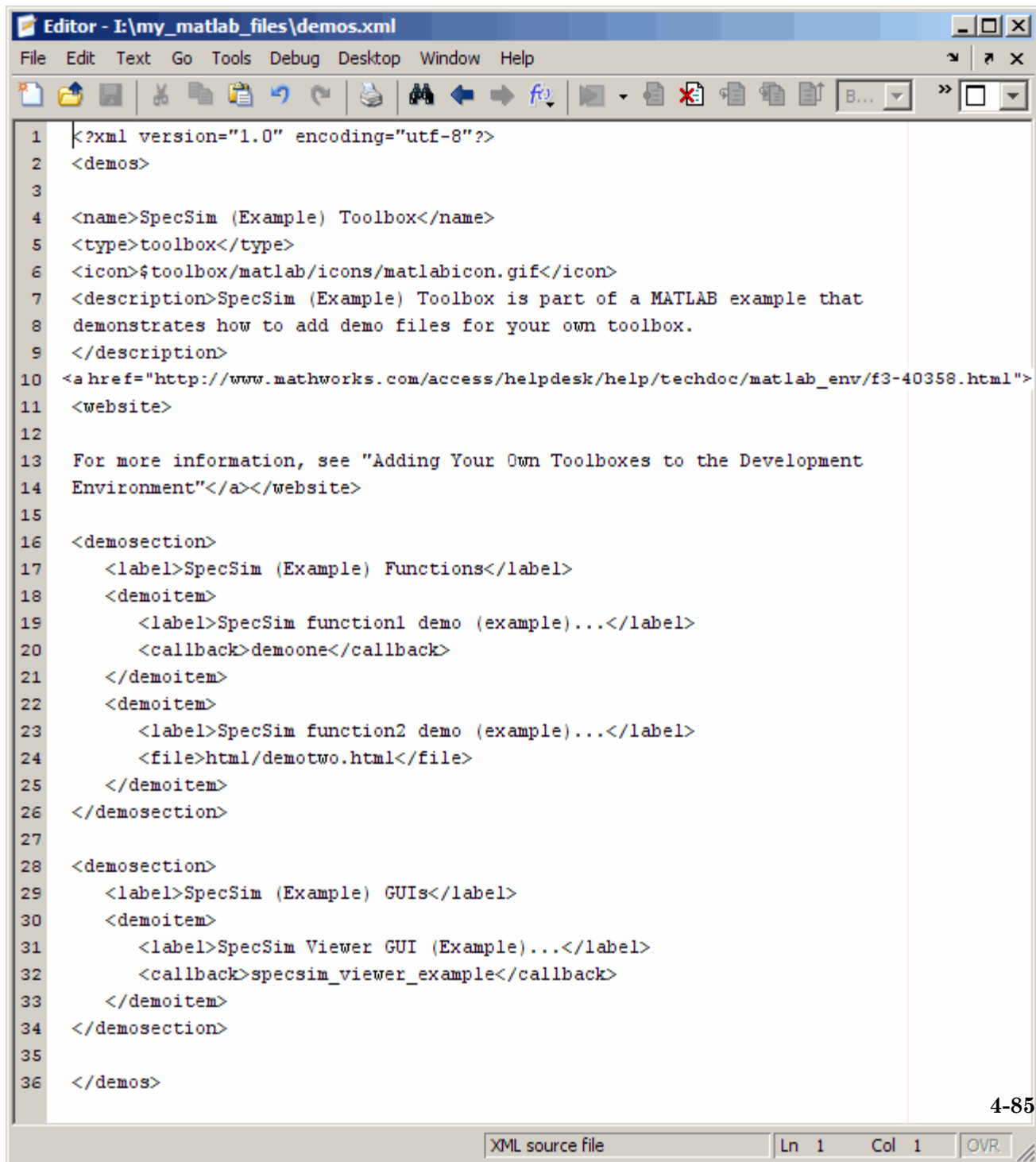
You can also put your toolbox on the **Start** button. For more information, see "Adding Your Own Toolboxes to the Start Button" on page 2-72.

Procedure for Adding Demos for Your Own Toolboxes

An example `demos.xml` file accompanies the following instructions,which adds demos for the SpecSim (Example) Toolbox to the Help browser. Use the example file as a basis for creating a `demos.xml` file to add demos for your own toolbox.

1 Copy *matlabroot*`/help/techdoc/matlab_env/examples/demos.xml` to a directory for which you have write access and add it to the search path. This example uses `I:/my_matlab_files`.

2 To see how the `demos.xml` file looks in the Help browser, open the Help browser and click the **Demos** tab to see the entries for the SpecSim (Example) Toolbox under `Toolboxes`. Be sure to disable the product filter, or select the `SpecSim (Example) Toolbox` in the product filter, if the product filter is enabled. Use **Preferences > Help** to access the product filter.

3 In the Editor, open `I:/my_matlab_files/demos.xml`

```
Editor - I:\my_matlab_files\demos.xml                                    _ □ ×

File   Edit   Text   Go   Tools   Debug   Desktop   Window   Help             ↘ | ⱶ ✕

1    <?xml version="1.0" encoding="utf-8"?>
2    <demos>
3
4    <name>SpecSim (Example) Toolbox</name>
5    <type>toolbox</type>
6    <icon>$toolbox/matlab/icons/matlabicon.gif</icon>
7    <description>SpecSim (Example) Toolbox is part of a MATLAB example that
8    demonstrates how to add demo files for your own toolbox.
9    </description>
10   <a href="http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_env/f3-40358.html">
11   <website>
12
13   For more information, see "Adding Your Own Toolboxes to the Development
14   Environment"</a></website>
15
16   <demosection>
17      <label>SpecSim (Example) Functions</label>
18      <demoitem>
19         <label>SpecSim function1 demo (example)...</label>
20         <callback>demoone</callback>
21      </demoitem>
22      <demoitem>
23         <label>SpecSim function2 demo (example)...</label>
24         <file>html/demotwo.html</file>
25      </demoitem>
26   </demosection>
27
28   <demosection>
29      <label>SpecSim (Example) GUIs</label>
30      <demoitem>
31         <label>SpecSim Viewer GUI (Example)...</label>
32         <callback>specsim_viewer_example</callback>
33      </demoitem>
34   </demosection>
35
36   </demos>

                                           XML source file        Ln  1    Col  1    OVR
```

The root tag is <demos> and it includes <name>, <type>, <icon>and <description> for your toolbox demos.

Add a <demoitem> or <demosection> containing <demoitem> s, where each <demoitem> is for a demo.

Within one demos.xml file, every <demoitem> must either be a direct child of <demos>, or all must be within <demosection> tags. You cannot have a<demoitem> as child of <demos> and have another <demoitem> in demosection.

Save the demos.xml file.

Line	XML Tag	Value for Example	Notes
2	<demos>	No value	The root element for a demos.xml file.
4	<name>	SpecSim (Example) Toolbox	Name of your toolbox.
5	<type>	toolbox	The product type. Allowable values are matlab, simulink, toolbox, blockset, links_targets, or other. This determines where your entry appears in the **Demos** pane. Setting this to toolbox makes SpecSim (Example) Toolbox appear in **Demos > Toolboxes**. Within a type, entries are displayed in alphabetical order.
6	<icon>	$toolbox/matlab/icons/ matlabicon.gif	Use an icon for your demo. You can specify a relative path, that is, relative to the location of the info.xml file. This example uses $toolbox /... to reference icons in a *matlabroot*/toolbox directory; see *matlabroot*/toolbox/matlab/icons for icons provided with MATLAB.

Line	XML Tag	Value for Example	Notes
7	`<description>`	SpecSim (Example) Toolbox is ...	The description that appears in the Help browser display pane, on the main page for your demo.
11 to 14	`<website>` `<a href= "url"</a> </website>`	... http://www.mathworks.com/ access/helpdesk/help/ techdoc/matlab_env/ f3-40358.html"> For more information, see ...	Links to a Web site. For example, MATLAB and related product demos include this link near the top, on the right: `Product page at mathworks.com`. This tag can appear anywhere before the `</demos>` tag. For the example, displays this help page at The MathWorks Web site.
16	`<demosection>`	No value required	Begins a section of individual demos: `<label>`, `<description>`, and `<demoitem>`. Use for each demo you want in the section.
17	`<label>`	SpecSim (Example) Functions	Title shown for a `<demosection>` or `<demoitem>`. For `<demoitem>`, `<label>` is required.
18	`<demoitem>`	No value required	Contains `<label>` and `<callback>`, and/or `<file>`. One `<demoitem>` per demo.
19	`<label>`	SpecSim function1 demo (example)...	Title shown for `demoitem`.
None	`<type>`	Not shown	The type of demo. This causes the icon for that type to appear with the demo in the list of demos for your toolbox. Allowable values are: `M-file`, `model`, `M-GUI`, `video`, or `other`. If you include `<type>`, MATLAB does not automatically create headers in the demo pages.

Line	XML Tag	Value for Example	Notes
20	`<callback>`	demoone (file not actually provided; this is only an example name)	The callback runs when you double-click the demo icon or click the title text for the demo item in the display pane.
24	`<file>`	html/demotwo.html	The HTML file displays in the Help browser when you click the demo in the list. Specify a relative path from the location of `demos.xml`.
None	`<dependency>`	Not shown	Optional. Specifies other products required to run the demo, such as another toolbox. The text must match a product name specified in an `info.xml` file on the search path or in the current directory.

5 In the Help browser, right-click in the list of demos and from the context menu, select **Refresh** to be sure that the Help browser is including the changes to your `demos.xml` file. Your demos should now appear in the **Demos** pane.

For additional examples, look at other `demos.xml` files for MathWorks products.

Validation Errors for info.xml Files

When MATLAB finds an `info.xml` file on the search path or in the current directory, it assumes that the file is intended to add information to the Help browser or to the **Start** button, and automatically validates the file against the supported schema. If there is an invalid construct in the `info.xml` file, MATLAB displays an error in the Command Window. The error is typically of the form:

```
XML-file failed validation against schema located in
...
XML-file name: full path to...\info.xml
```

It can appear when you start MATLAB, press the **Start** button, or in other situations.

These are some reasons why you might see the XML file validation error and what to do:

- "Outdated info.xml File for MATLAB Product" on page 4-89
- "info.xml File Unrelated to MATLAB Product" on page 4-89
- "info.xml File Intended for Use with MATLAB Software Contains Invalid Constructs" on page 4-89

Outdated info.xml File for MATLAB Product

If you have an `info.xml` file from an old version of MATLAB, it might contain constructs that are no longer valid. In these cases, the full path to the `info.xml` file usually indicates an old version of MATLAB, for example, `...\MATLAB\R14\...` . The error is not actually causing any problems and you can safely ignore it. To ensure that the error does not occur, move or remove the old `info.xml` file, assuming you no longer use that version of MATLAB. If you do use it, remove the directory from the search path and do not make it the current directory in the new version of MATLAB.

info.xml File Unrelated to MATLAB Product

If the `info.xml` exists for reasons other than adding information to the **Start** button or Help browser, the validation MATLAB attempts to perform is irrelevant. In this case, the error is not actually causing any problems and you can safely ignore it. To ensure that the error does not occur, one option is to rename the offending `info.xml` file. Another option is to move the offending `info.xml` file to a directory that is not on the search path and is unlikely to be used as the current directory in MATLAB.

info.xml File Intended for Use with MATLAB Software Contains Invalid Constructs

If the `info.xml` file is intended for adding information to the **Start** button or Help browser, you need to correct the reported problem. Use the message in the error to isolate the problem or use any validator. One validator you can use is from the W3C at `http://www.w3.org/2001/03/webdata/xsv`. For

more information about the structure of the info.xml file, consult its schema, located at *matlabroot*/sys/namespace/info/v1/info.xsd.

Related Resources

In this section...
"Getting Technical Support" on page 4-91
"Getting More Information About MathWorks Products" on page 4-91
"Version and License Information" on page 4-92
"Terms of Use and Patents" on page 4-93
"Newsgroup for MathWorks Products" on page 4-93
"Accessing User-Contributed Files — File Exchange" on page 4-93

Getting Technical Support

Technical Support provides help for problems you have with MathWorks products:

- Find specific Technical Support information using the Help browser search feature. Run a search for a specified term. The end of the results list includes a link that runs the same search on the support database. This database, on the MathWorks Web site, provides the most up-to-date solutions, bug reports, and technical notes for questions posed by users.

- Select **Help > Web Resources > Support** to go to the Support Web page (`http://www.mathworks.com/support`). You can find out about other types of information such as third-party books, ask questions, make suggestions, view known bugs and workarounds, and report possible bugs.

  You will be prompted to provide information about your account or license, as well as your product version numbers. For more about how to obtain the information, see "Version and License Information" on page 4-92.

  If you cannot access the Web site, send an e-mail Technical Support using the address `support@mathworks.com`.

Getting More Information About MathWorks Products

Following are some additional resources for help with MATLAB and related products:

- Newsletters — The MathWorks publishes News and Notes twice a year, containing feature articles, technical notes, and product information for users of MATLAB. More frequently, The MathWorks issues MATLAB Digest, an electronic bulletin consisting of technical notes, solutions, and timely announcements to the user community. Select **Help > Web Resources > MATLAB Newsletters** or see `http://www.mathworks.com/company/newsletters/`.

- Books — There are many books about MATLAB. For a list with descriptions, see `http://www.mathworks.com/support/books/`.

- Seminars and Training — The MathWorks regularly presents free seminars on special topics conducted in various locations. Webinars on special topics are presented via the Web. For details, see `http://www.mathworks.com/company/events/`.

  The MathWorks also offers training classes for MATLAB and other products. For details, select **Help > Web Resources > Training**, or see `http://www.mathworks.com/services/training`.

Version and License Information

If you need version or license information for a product, in the product, select **Help > About**. The About dialog box displays the version for the product.

If you want to find out if you are running a 32-bit or 64-bit version of MATLAB, select **Help > About MATLAB** from any desktop tool, and view the value in the resulting About MATLAB dialog box. Use this feature if you are not sure which version you are running and you want to take advantage of the benefits of 64-bit MATLAB. Or use this feature if you want to use files that depend on a specific platform, such as MEX-files. The About MATLAB dialog box also provides the value for `arch`, for example `win32` or `win64`, that you use with the `mex` function.

You can access information about your passcodes and licenses, as well as get trial versions of products using **Help > Web Resources > MathWorks Account**.

You can also use functions to view the version and license data. This is how you can get the information when a product does not have a **Help** menu. For more information, see the reference pages for `computer`, `license`, `ver`, and `version`.

Terms of Use and Patents

To access the terms of use and patent information for MathWorks products, select either item from the **Help** menu, or from the links at the bottom of any page in the Help browser.

Newsgroup for MathWorks Products

The Usenet newsgroup for MATLAB and related products, `comp.soft-sys.matlab`, (also known as `cssm`) is read by thousands of users worldwide. Access the newsgroup to ask for or provide help or advice. You can read and submit postings as well as view and search through a sizable archive of postings using the MATLAB Central Newsgroup Access Web page on the MathWorks Web site, `http://www.mathworks.com/matlabcentral`. You can access this via **Help > Web Resources > MATLAB Newsgroup Access** from any desktop component.

First-time users to the newsgroup should read the newsgroup FAQ, linked to from the MATLAB Central page. It is a good practice to try to solve your own problem using the documentation and Technical Support database before posting a question to the newsgroup. Be sure to post with a meaningful subject that briefly describes the nature of the issue.

Accessing User-Contributed Files — File Exchange

You can download M-files and other related files contributed by users and developers of MATLAB, Simulink, and related products from MATLAB Central on the MathWorks Web site. Before you write an M-file yourself, check the list of contributed files to see if someone has already written it. If you create an M-file that others might benefit from, you can submit it to File Exchange for others to use.

To use File Exchange, you need an active Internet connection. Then, select **Help > Web Resources > MATLAB File Exchange**. The MATLAB Central File Exchange page on the MathWorks Web site opens in your system browser. You can also access the repository directly at `http://www.mathworks.com/matlabcentral/fileexchange`.

Providing Feedback About Documentation

To report problems or provide comments or suggestions to The MathWorks about the documentation or help features, use the **Provide feedback about this page** link that appears at the top and bottom of each page in the Help browser.

5
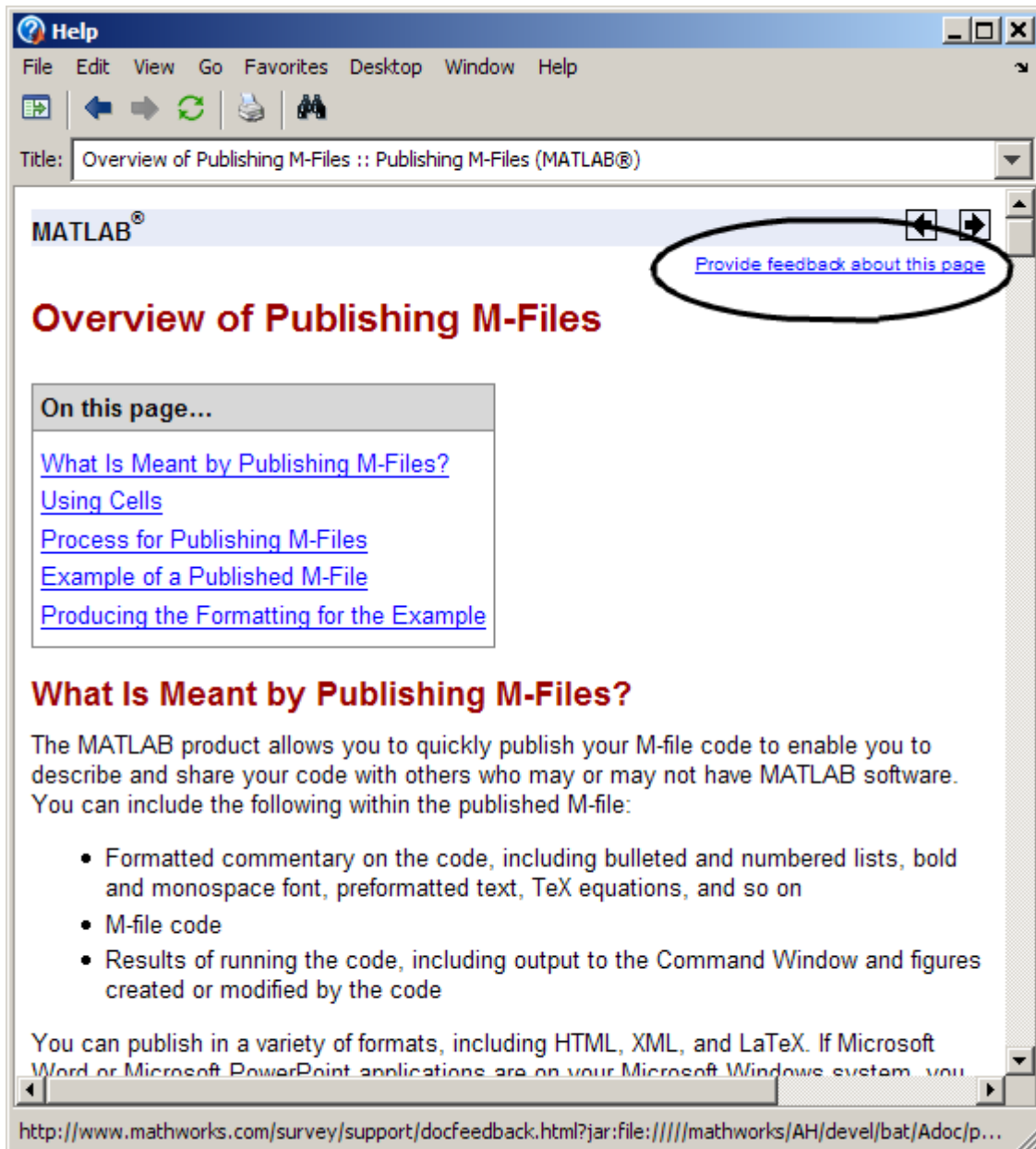
Workspace, Search Path, and File Operations

MATLAB Workspace

In this section...

About the Workspace

The workspace consists of the set of variables built up during a session of using the MATLAB software and stored in memory. You add variables to the workspace by using functions, running M-files, and loading saved workspaces. For example, if you run these statements,

```
A = magic(4)
R = randn(3,4,5)
```
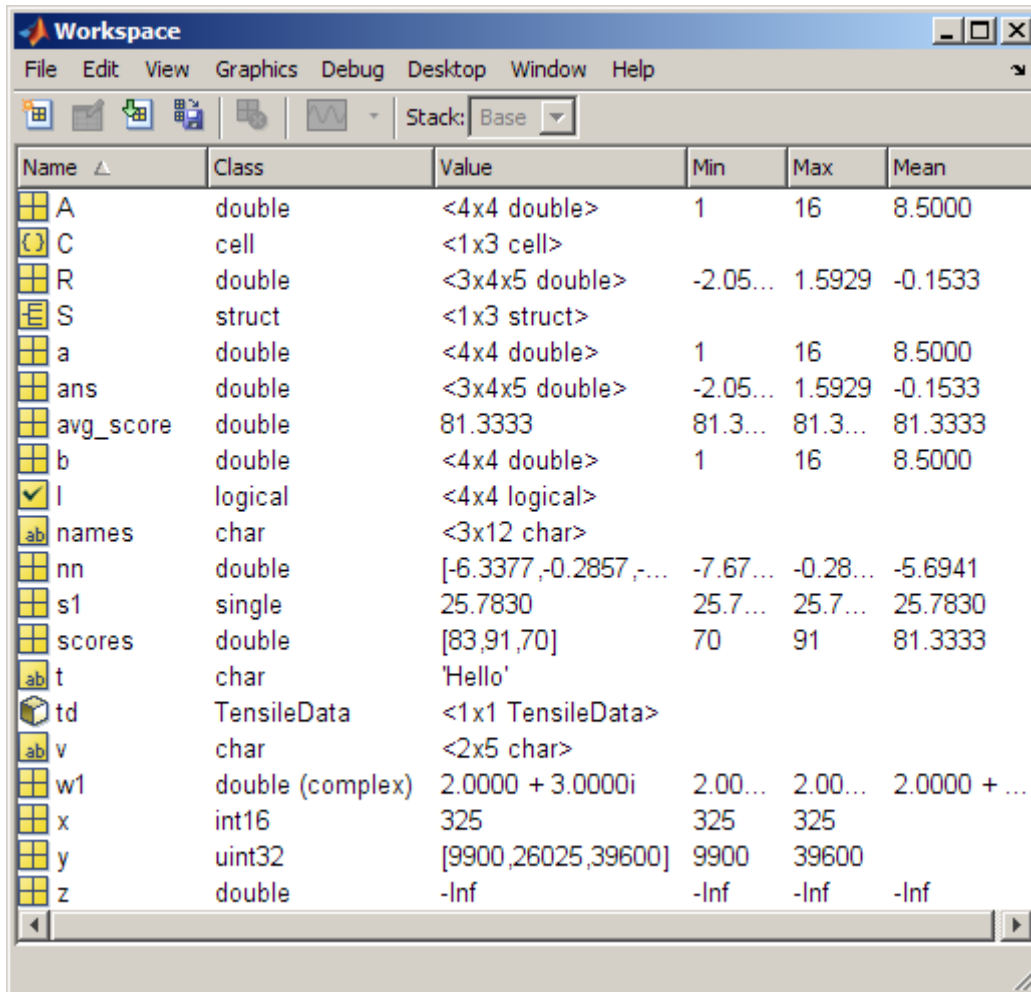
the workspace includes two variables, A and R.

You can perform workspace and related operations using the Workspace browser. When available, equivalent functions are documented with each feature of the Workspace browser. If you have an active Internet connection, you can watch the Workspace browser video demo for an overview of the major functionality:

Opening the Workspace Browser

To open the Workspace browser, select **Desktop > Workspace** in the MATLAB desktop, or type `workspace` at the Command Window prompt.

The Workspace browser opens.

Name △	Class	Value	Min	Max	Mean
A	double	<4x4 double>	1	16	8.5000
C	cell	<1x3 cell>			
R	double	<3x4x5 double>	-2.05...	1.5929	-0.1533
S	struct	<1x3 struct>			
a	double	<4x4 double>	1	16	8.5000
ans	double	<3x4x5 double>	-2.05...	1.5929	-0.1533
avg_score	double	81.3333	81.3...	81.3...	81.3333
b	double	<4x4 double>	1	16	8.5000
l	logical	<4x4 logical>			
names	char	<3x12 char>			
nn	double	[-6.3377,-0.2857,-...	-7.67...	-0.28...	-5.6941
s1	single	25.7830	25.7...	25.7...	25.7830
scores	double	[83,91,70]	70	91	81.3333
t	char	'Hello'			
td	TensileData	<1x1 TensileData>			
v	char	<2x5 char>			
w1	double (complex)	2.0000 + 3.0000i	2.00...	2.00...	2.0000 + ...
x	int16	325	325	325	
y	uint32	[9900,26025,39600]	9900	39600	
z	double	-Inf	-Inf	-Inf	-Inf

You can specify which buttons you want to appear on the toolbar using preferences. Select **File > Preferences > Toolbars** to open the dialog box. Click **Help** for information using the dialog box.

Viewing and Editing Values in the Current Workspace

The Workspace browser shows the name of each variable or object, the class (also represented by the icon), its value, and where relevant, the **Min**, **Max**, and **Mean** calculations. MATLAB performs these calculations using the min, max, and mean functions, and updates the results automatically. These are other features of the Workspace browser:

- You can display additional columns, including size (dimensions), size in bytes, and other common statistical calculations such as mode and standard deviation. To show or hide columns, select **View > Choose Columns** or right-click any column header. To specify the size limit for calculations and how NaNs are considered, use "Preferences for the Workspace Browser" on page 5-11.

- To resize a column of information, drag the column header border. To reorder columns, drag a column header to a new position.

- You can select the column on which to sort as well as reverse the sort order of any column. Click a column header to sort on that column. Click the column header again to reverse the sort order in that column. For example, to sort on **Name**, click the column header once. To change from ascending to descending, click the header again. You cannot sort by the **Value** column in the Workspace browser.

- You can directly edit variable values in the Workspace browser **Value** column. To edit a value, position the pointer in the **Value** column at the row you want to edit, click, and type the new value.

- To view more of the data for a variable, as well as to more easily edit it, double-click a variable name and it opens in the Variable Editor. For more information, see "Viewing and Editing Workspace Variables with the Variable Editor" on page 5-14.

Function Alternative

Use who to list the current workspace variables. Use whos to list the variables and information about size and class. For example:

```
>> who

Your variables are:

A          S          avg_score  names      scores     v          y
C          a          b          nn         t          w1         z
R          ans        l          s1         td         x

>> whos
  Name            Size            Bytes  Class         Attributes

  A               4x4               128  double
  C               1x3               348  cell
  R               3x4x5             480  double
  S               1x3               826  struct
  a               4x4               128  double
  ans             3x4x5             480  double
  avg_score       1x1                 8  double
  b               4x4               128  double
  l               4x4                16  logical
  names           3x12               72  char
  nn              3x3                72  double
  s1              1x1                 4  single
  scores          1x3                24  double
  t               1x5                10  char
  td              1x1               152  TensileData
  v               2x5                20  char
  w1              1x1                16  double        complex
  x               1x1                 2  int16
  y               1x3                12  uint32
  z               1x1                 8  double
```

Use exist to see if the specified variable is in the workspace.

Saving the Current Workspace

The workspace is not maintained across sessions of MATLAB. When you quit
MATLAB, the workspace is cleared. You can save any or all of the variables
in the current workspace to a MAT-file, which is a binary file specifically
for use in MATLAB. You can then load the MAT-file at a later time during

the current or another session to reuse the workspace variables. MAT-files use a .mat extension.

Note The .mat extension is also used by Microsoft Access application. You can change the default file association in the Microsoft Windows operating system to associate MAT-files with either MATLAB or the Access application.

Saving All Variables

To save all of the workspace variables using the Workspace browser:

1 Select **File > Save Workspace As** from the Workspace browser, or click the Save button in the Workspace browser toolbar.

The Save to MAT-File dialog box opens.

2 Specify the location and **File name**. MATLAB automatically supplies the .mat extension.

3 Click **Save**.

The workspace variables are saved under the MAT-file name you specified.

Saving Selected Variables

To save some but not all of the current workspace variables:

1 Select the variable in the Workspace browser. To select multiple variables, **Shift**+click or **Ctrl**+click.

2 Right-click, and from the context menu, select **Save As**.

The Save to MAT-File dialog box opens.

3 Specify the location and **File name**. MATLAB automatically supplies the .mat extension.

4 Click **Save**.

The workspace variables are saved under the MAT-file name you specified.

Another way to save selected variables from the Workspace browser to a MAT-file is by dragging the selected variables from the Workspace browser to the Current Directory browser. For more information, see "Creating and Updating MAT-Files in the Current Directory Browser" on page 5-70.

Specifying the Format When Saving MAT-Files

To specify preferences for saving MAT-files that pertain to compression, and compatibility between different versions of MATLAB, see "MAT-Files Preferences" on page 2-109.

Function Alternative

To save workspace variables, use the `save` function followed by the filename you want to save to. For example,

```
save('june10')
```

saves all current workspace variables to the file `june10.mat`.

If you don't specify a filename, the workspace is saved to `matlab.mat` in the current directory. You can specify which variables to save, as well as control the format in which the data is stored, such as ASCII. For these and other forms of the function, see the reference page for `save`. MATLAB provides additional functions for saving information — see "Data Import and Export" in the MATLAB Programming Fundamentals documentation.

Viewing and Loading a Saved Workspace and Importing Data

You can view saved variables and load the variables and other data into the workspace using the Current Directory browser, the Import Wizard, or the `load` function.

Viewing Variables in MAT-Files and Loading Them into the Workspace

Use the Current Directory browser to view the contents of a MAT-file without loading the file into MATLAB. For details, see "Viewing Information About the Selected File or Directory in the Current Directory Browser" on page 5-61

You can also load selected variables by dragging them from a MAT-file in the Current Directory browser to the Workspace browser. For details, see "Opening Files In MATLAB from the Current Directory Browser" on page 5-67.

Function Alternative. Use `whos` with the `-file` option.

Importing Data

MATLAB provides an Import Wizard to load MAT-files and other forms of data into the workspace. This procedure briefly describes how to use the Import Wizard from the Workspace browser to import data from MAT-files.

1 Click the Import Data button on the toolbar in the Workspace browser.

   The Import Data dialog box opens.

2 Select the MAT-file you want to load and click **Open**.

   The variables and their values, as stored in the MAT-file, are loaded into the current workspace. Any variables in the MAT-file that are not in the workspace are added to the workspace. If any variables being loaded have the same names as variables in the current workspace, MATLAB asks if you want to replace the values in the current workspace with the values in the MAT-file, or cancel.

You can also use the Workspace browser to import data you previously copied to the clipboard by selecting **Edit > Paste to workspace**, or use **Ctrl+V**. This imports the clipboard data using the Import Wizard.

For more information about the Import Wizard, see the "Using the Import Wizard" in the Programming Fundamentals documentation.

Function Alternative for Loading Variables

Use `load` to open a saved workspace. For example,

```
load('june10')
```

loads all workspace variables from the file `june10.mat`.

Changing and Copying Variable Names

To rename a variable in the workspace, right-click the variable in the Workspace browser and select **Rename** from the context menu. Type the new variable name over the existing name and press **Enter**.

To copy variable names to the clipboard, select the workspace variables and select **Edit > Copy**. You can then paste the names, for example, into the Command Window. Multiple variables are comma separated.

Deleting Workspace Variables

You can delete a variable, which removes it from the workspace:

1 In the Workspace browser, select the variable, or **Shift**+click or **Ctrl**+click to select multiple variables. To select all variables, choose **Select All** from the **Edit** or context menus.

2 Press the **Delete** key on your keyboard or click the Delete button 🖳 on the Workspace browser toolbar.

3 A confirmation dialog box might appear. If it does, click **OK** to clear the variables.

The confirmation dialog box appears if you selected that preference. For more information, see "Confirmation Dialogs Preferences" on page 2-112.

To delete all variables, select **Edit > Clear Workspace** from any desktop tool.

Function Alternative

Use the `clear` function to clear variables from the workspace. For example,

```
clear A M
```

clears the variables A and M from the workspace.

Use the `clearvars` function with the `-except` option to keep the specified variables and clear all other variables.

Viewing Base and Function Workspaces Using the Stack

When you run M-files, MATLAB assigns each function its own workspace, called the function workspace, which is separate from the base workspace in MATLAB. To access the base and function workspaces when running or debugging M-files, use the **Stack** field in the Workspace browser. The **Stack** field is only available in debug mode and otherwise is grayed out. The **Stack** field is also accessible from the Variable Editor and the Editor/Debugger. For more information, see "Finding Errors, Debugging, and Correcting M-Files" on page 6-120. See also the `dbstack` and `evalin` functions.

Creating Plots from the Workspace Browser

From the Workspace browser, you can generate a plot of a variable. To create a plot, click the Plot button on the Workspace browser toolbar and select the plot type. The plot appears in a figure window. The button itself changes to reflect the currently selected style of plot, for example bar or stem.

This feature is only available for variables whose classes can be plotted, such as numeric. Open the variable in the Variable Editor for additional plotting options.

In addition, you can right-click the variable you want to plot. From the context menu, choose the type of plot you want to create.

You can also **Shift**+click or **Ctrl**+click to select multiple variables to plot together. When one of the variables is named `time`, `t`, or `T`, MATLAB assumes it is the independent variable.

For more information about creating graphs in MATLAB, see the *MATLAB Graphics* documentation.

Opening Variables and Objects for Viewing and Editing

In the Workspace browser, double-click a variable and it opens in the Variable Editor where you can view and edit the contents of the variable. See "Viewing and Editing Workspace Variables with the Variable Editor" on page 5-14 for more information.

Some toolboxes allow you to double-click an object in the Workspace browser to open a viewer or other tool appropriate for that object. For details, see the toolbox documentation for that object type.

Preferences for the Workspace Browser

The Workspace browser displays statistical calculations for variables. Use preferences to restrict the size of arrays on which calculations are performed and to specify if NaNs are included or ignored in calculations. Select **File > Preferences > Workspace** to open the dialog box. Make changes and click **OK**.
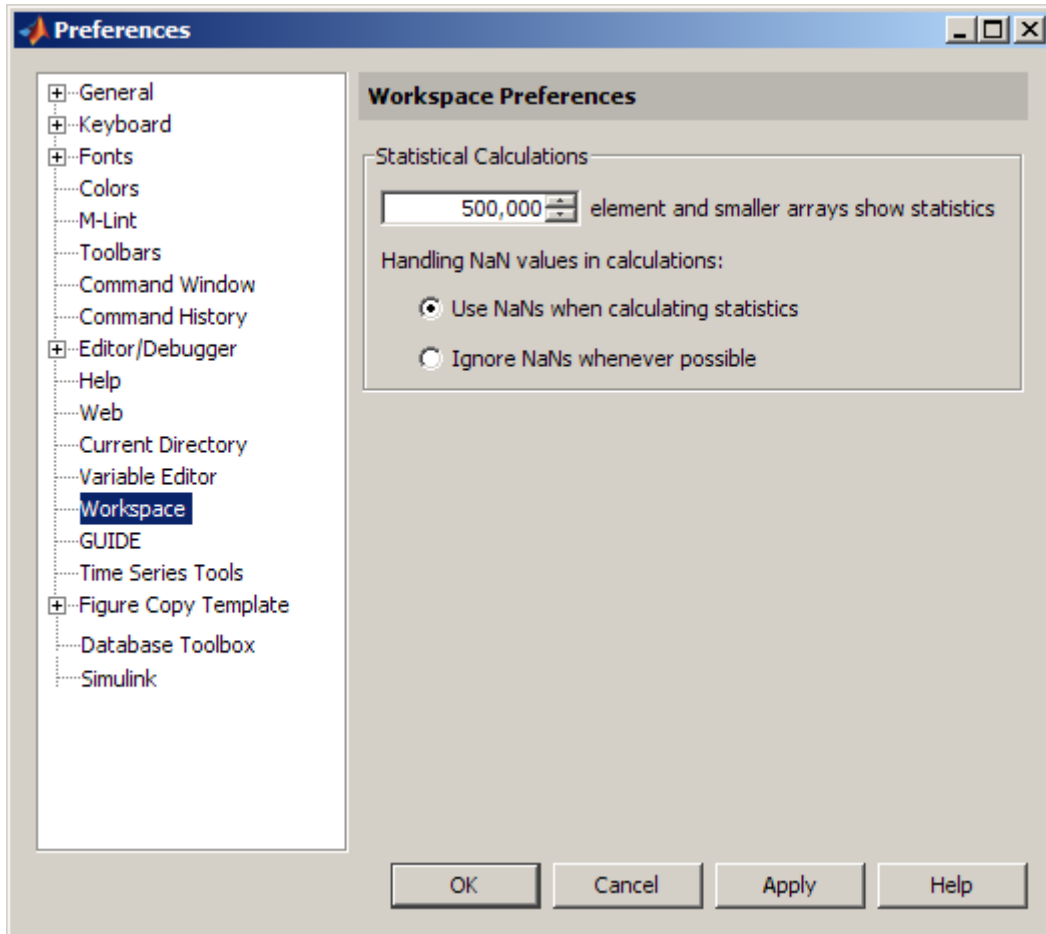
Specify Maximum Array Size on Which to Compute Statistics

If you show statistical columns in the Workspace browser, and if you work with very large arrays, you might experience performance issues when the data changes as MATLAB updates the statistical results. In that event, show only the columns of interest to you and hide those you do not need.

Another step you can take is specify via a preference that the Workspace browser *not* perform statistical calculations on the largest arrays. Use the arrows to change the value of the maximum array size for which you want

the Workspace browser to perform statistical calculations. The default value is `500,000` elements. Any variable exceeding that size reports `<Too many elements>` instead of statistical results.

Handling NaN Values in Calculations

If your data includes NaNs, you can specify that the statistical calculations consider the NaNs or ignore the NaNs. For example, if a variable includes a `NaN` and the preference is set to **Use NaNs when calculating statistics**, the values for **Min**, **Max**, **Var** and some others will appear as `NaN`, although **Mode**, for example, shows a numeric result. With the preference set to **Ignore NaNs whenever possible**, numeric results appear for most of the statistical columns including **Min** and **Max**; **Var**, however, is still reported as `NaN`.

For more information about statistical values in the Workspace browser, see "Viewing and Editing Values in the Current Workspace" on page 5-4.

Viewing and Editing Workspace Variables with the Variable Editor

About the Variable Editor

The Variable Editor is a desktop component that lets you display variables in the current workspace. Use it to view and edit values of one or two-dimensional arrays, character strings, cell arrays, structures, and objects and their properties. You can also view the contents of multidimensional arrays. Edits you make in the Variable Editor are immediately reflected in the workspace. It also supports copying and pasting of data values.

Opening the Variable Editor

To open the Variable Editor from the Workspace browser, perform these steps:

1 In the Workspace browser, select the variable you want to open. Use **Shift**+click or **Ctrl**+click to select multiple variables, or use **Ctrl+A** to select all variables to open.

2 Click the Open Selection button ![icon] on the toolbar. For one variable, you can also open it by double-clicking it.

The Variable Editor opens, displaying the values for the selected variable. The class and size of the value appear below the toolbar, and for some classes, include a link to the help for that class.

Size and class of the variable.

Click an element and then edit its value.
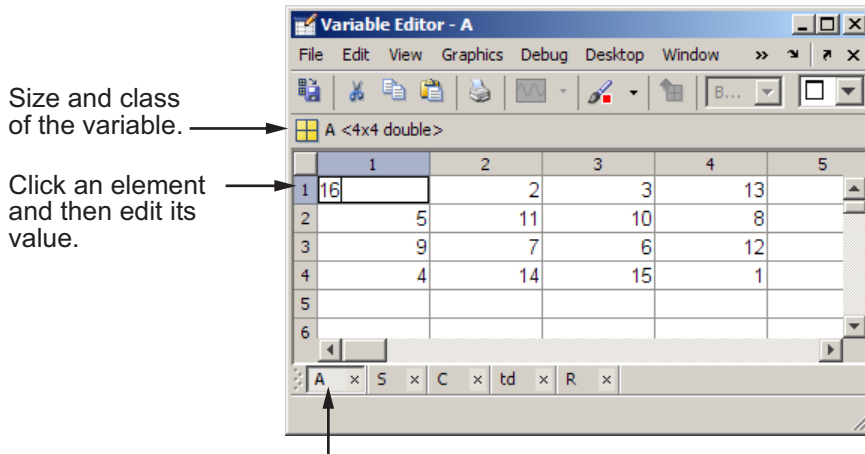
Select a tab to view a variable that you have open in the Variable Editor.

Repeat the steps to open additional variables in the Variable Editor. Access each variable via its tab at the bottom of the window, or use the **Window** menu.

Changes you make to variables via the Command Window or other operations automatically update the information for those variables in the Variable Editor.

Note The maximum number of elements in a variable that you can open in the Variable Editor is not limited by the MATLAB software, but is based on your operating system or the amount of physical memory installed on your system.

Keyboard Alternatives

To open a variable in the Variable Editor, use `openvar` with the name of the variable you want to open as the argument. For example, type

```
openvar('A')
```

You need to enclose the variable's name in single quotes, because the Variable Editor needs to know the name of the variable in order to be notified if the variable changes value, is deleted, or goes out of scope. If you were to type `openvar(A)` instead of `openvar('A')`, the Variable Editor would receive the value of `A` instead of its name. However, `openvar varname` and `openvar 'varname'` both work, as string arguments are assumed when using *command syntax*. See "Command vs. Function Syntax" in the MATLAB Programming Fundamentals documentation for more information.

MATLAB opens `A` in the Variable Editor.

To see the contents of a variable in the workspace, just type the variable name at the Command Window prompt. For example, type

```
A
```

and MATLAB returns

```
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Viewing and Editing Cell Arrays, Structures, Objects, and Multidimensional Arrays in the Variable Editor

- "Cell Arrays — Viewing and Editing in the Variable Editor" on page 5-17

- "Structures — Viewing and Editing in the Variable Editor" on page 5-18

- "Objects and Their Properties — Viewing and Editing in the Variable Editor" on page 5-20

- "Multidimensional Arrays — Viewing in the Variable Editor" on page 5-24

Cell Arrays — Viewing and Editing in the Variable Editor

You can view and edit the content of cell arrays in the Variable Editor. In the Variable Editor, double-click an element of a cell array to open it as its own Variable Editor document. You can then view and edit the contents of that element. The following illustrations show an 1-by-3 cell array, C, and the contents of C{1,1}. When viewing an element in a cell array, for example, C{1,1}, use the Up button 🔲 to go to its cell array, for this example, C.
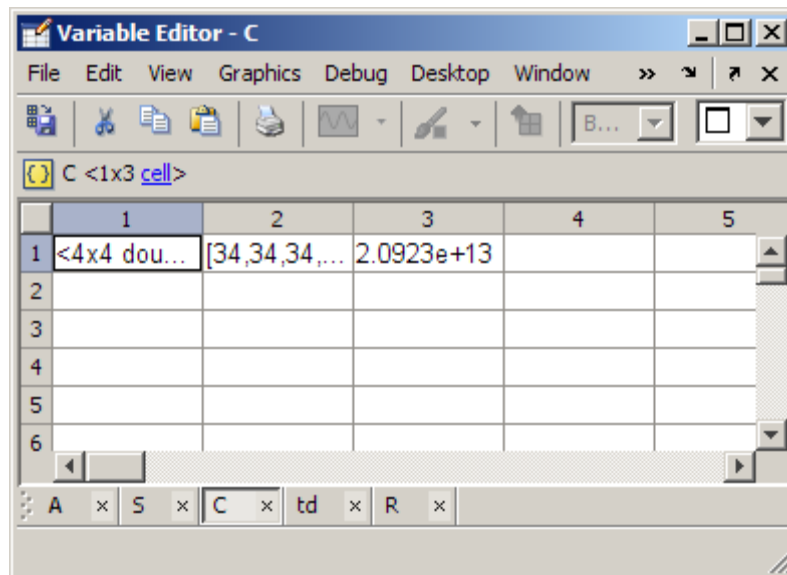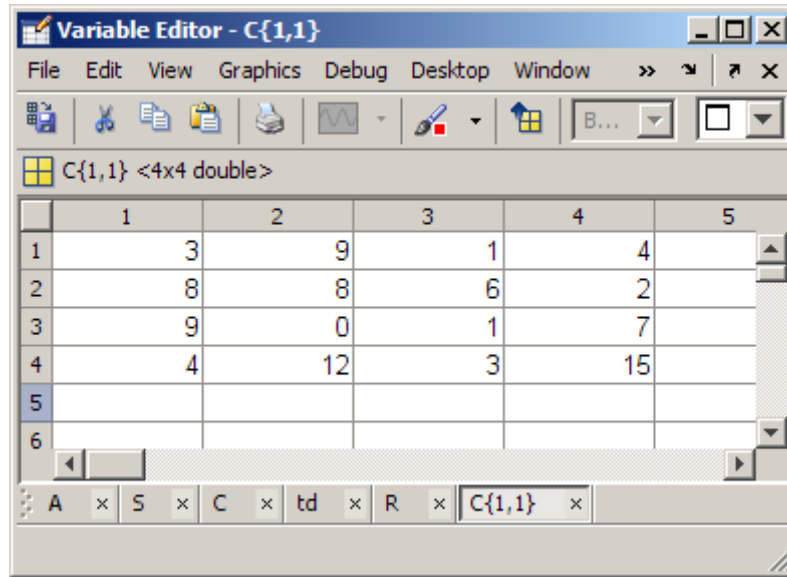
Structures — Viewing and Editing in the Variable Editor

You can view and edit the content of structures in the Variable Editor. In the Variable Editor, double-click an element of a structure to open it as its own Variable Editor document. The following illustrations show a 1-by-3 structure, S and the result of double-clicking S(1,2), which displays the contents in its own new document.
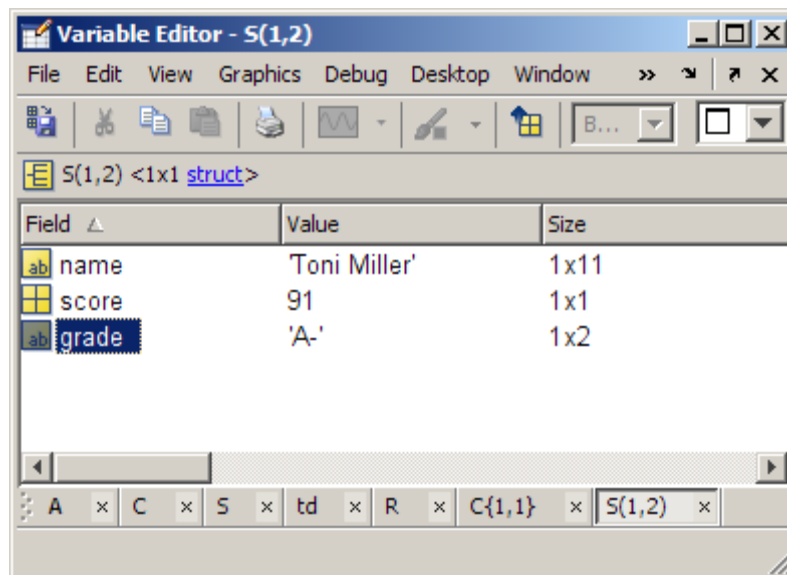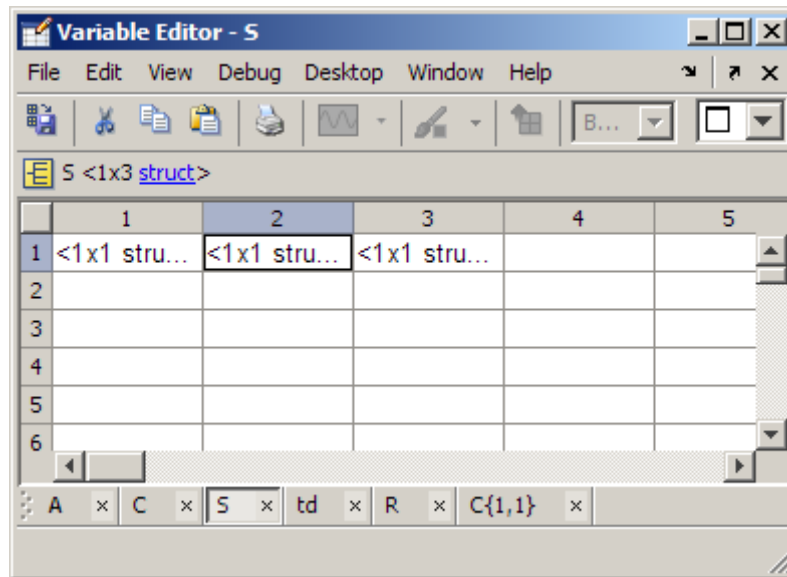
The information shown for the element of the structure is similar to that shown in the Workspace browser: **Field**, **Value**, **Size** and other information.

Right-click a column header to show or hide columns. Click a column header to sort by that column, and click again to reverse the sort order. When viewing an element in a structure, for example, S(1,2), use the Up button 🏠 to view the structure, for this example, S. This can help you navigate in the Variable Editor when there are many variables open.

To edit the value for an element, you can click the value and make changes. Or double click the value; a new Variable Editor document opens where you can click and then make the changes. The following illustration shows the result of double-clicking the grade field for S(1,2), where you can change its value. You can use the Up button go up from the field to view the element. For example, when viewing S(1,2).grade, click the Up button to view S(1,2).
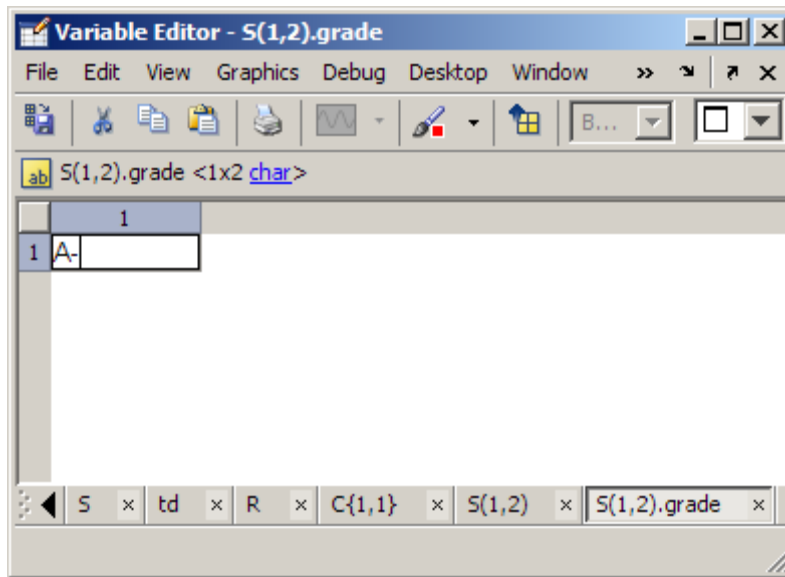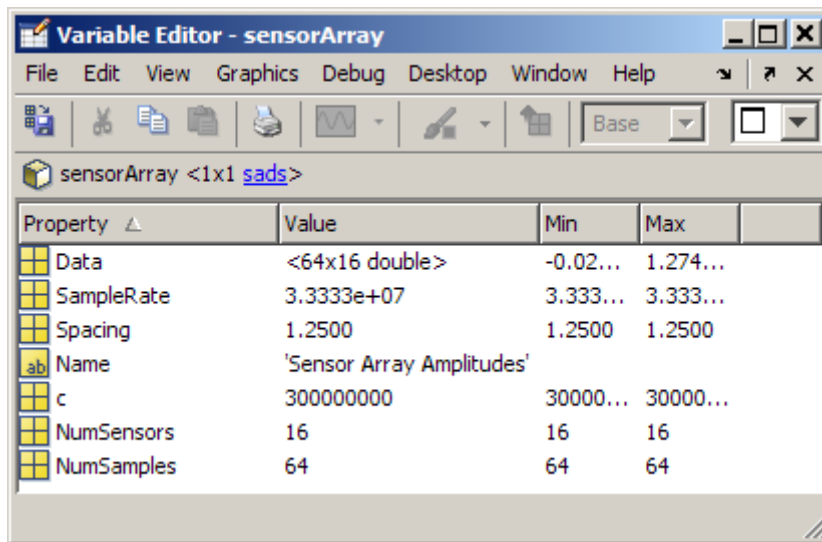
Objects and Their Properties — Viewing and Editing in the Variable Editor

- "Viewing Object Properties in the Variable Editor" on page 5-21
- "Editing Property Values in the Variable Editor" on page 5-23

- "Getting Help for Objects and Properties from the Variable Editor" on page 5-24

Viewing Object Properties in the Variable Editor. In the Variable Editor, you can view and edit properties of many MATLAB objects you create. When you open an object in the Variable Editor, it displays **Property**, **Value**, **Size**, and other information. To show or hide a column, right-click the column header. To sort by a column, click that column header; to reverse the sort order, click the column header again. You can view help for the class by clicking the class name link. Note that for `timeseries` objects, the Variable Editor has special attributes—for more information, see "Viewing Time Series Objects" in the MATLAB Data Analysis documentation.

The following illustration shows the `sensorArray` object of the `sads` class, in the Variable Editor. For more information about this example, see "Example of Help for a User-Created Class" on page 4-67.

Alternatively, double-click the value, which displays the value in its own document where you can more easily view and edit it. The following illustration shows the `sensorArray.Spacing` property opened in its own document. When viewing a property, use the Up button to view the object,

for this example, `sensorArray`. This can help you navigate in the Variable Editor when there are many variables open.
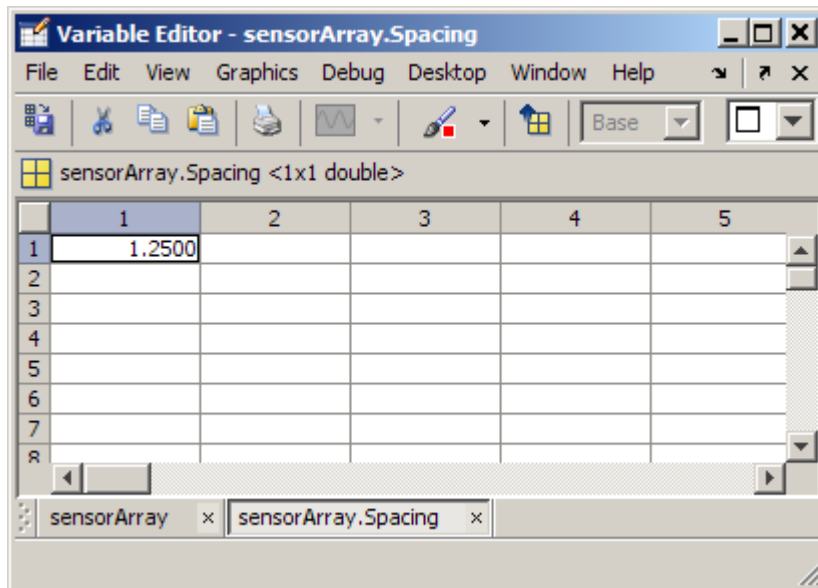
Additional icons, images of locks, denote protected and private properties of an object, indicating you cannot edit the values. The following illustration shows an `MException` object, `ME`, with the private properties `identifier` and `message`.

Icon denotes that
`identifier` and `message`
properties are private, and
therefore you cannot
edit their values.
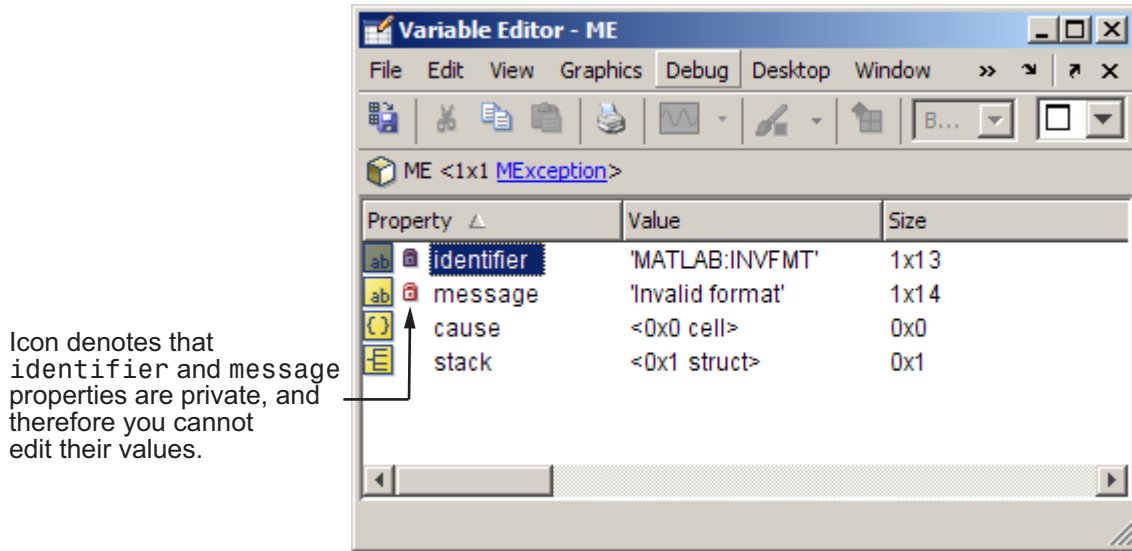
Editing Property Values in the Variable Editor. To edit a property's value
while viewing the object, click the value field and make changes, as shown
in the following illustration.
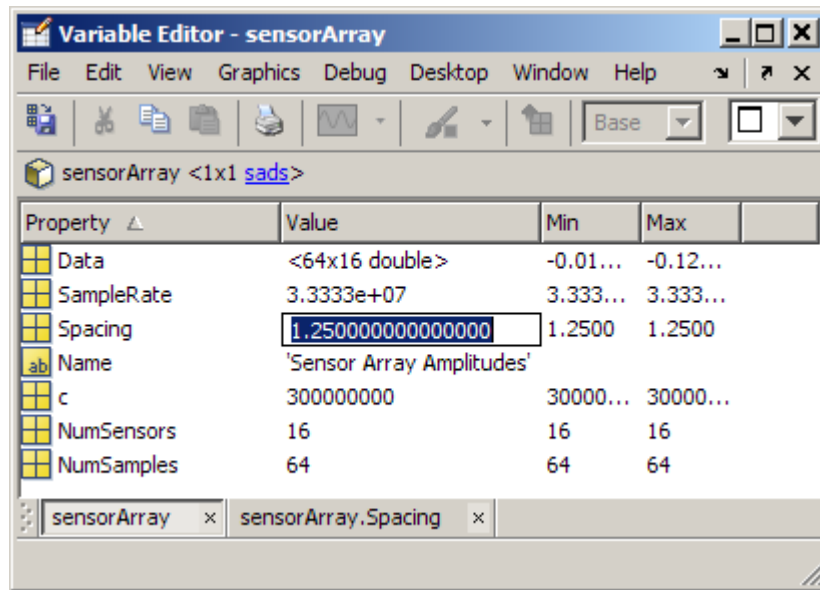
You can also double-click a property to open it in its own page in the Variable Editor, where you can edit it.

Getting Help for Objects and Properties from the Variable Editor. For most classes supplied by The MathWorks, when you click the link to the class name, for example, char, the reference page help appears in the Help browser. For user-created classes, help comments supplied in the class definition file display in HTML format in the Help browser. For more information, see "Help for User-Created Classes" on page 4-67.

Multidimensional Arrays — Viewing in the Variable Editor

You can view the contents of multidimensional arrays in the Variable Editor. When you open a multidimensional array in the Variable Editor, it does not have usual grid structure, because multidimensional arrays do not fit that format. You cannot double-click an element in a multidimensional array to edit it. The following illustration shows R = rand(3,4,5) opened in the Variable Editor.
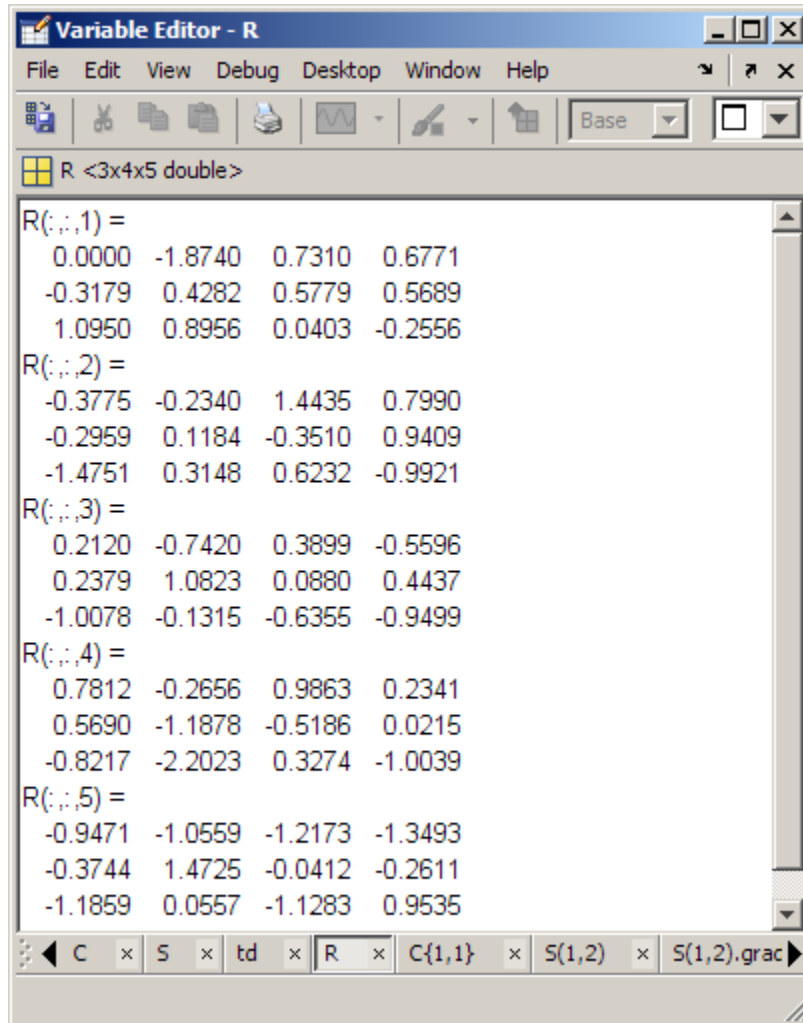
Navigating and Editing Shortcut Keys for the Variable Editor

Use the following shortcut keys (sometimes called hot keys) to move among elements in the Variable Editor. Navigating in the Variable Editor is much like navigating in the Microsoft Excel application.

Key	Result
Enter	Commit any changes to the element and move to next element, where next element is specified using "Preferences for the Variable Editor" on page 5-33 (default is down)
Tab	Move right

Within a selection, also moves from the last column to the first column in the next row |
Shift+Enter or **Shift+Tab**	Move in opposite direction of **Enter** or **Tab**
Page Up	Move up m rows, where m is the number of visible rows
Page Down	Move down m rows, where m is the number of visible rows
Home	Move to column 1
Ctrl+Home	Move to row 1, column 1
Shift+Home	Select to column 1
End	Move to last column in current row
F2 (**Ctrl+U** on Apple Macintosh platforms)	Edit current element, positioning cursor at the end of the element

Changing Size, Content, and Format of Variables in the Variable Editor

To increase the size of an array, scroll to the desired element in the variable and enter a value. The array automatically expands to accommodate the new value. Empty elements fill with zeros if numeric, or empty arrays if a cell array. To decrease the size of an array, select the rows or columns that you want to remove by clicking in the row or column header, which selects the entire row, and then right-click, and select **Delete** from the context menu. Similarly, you can make changes to arrays in structure and objects.

To change the value of an element in the Variable Editor, click the element and type a new value. Press **Enter**, or click another element to make the

change take effect. You can specify where the cursor moves to after you press **Enter** — see "Preferences for the Variable Editor" on page 5-33.

If you want to change the display format for the Variable Editor, select the **View** menu and choose a format. To change the default format for future use, use the Preferences dialog. For more information, see "Preferences for the Variable Editor" on page 5-33.

If you opened an existing MAT-file and made changes to it using the Variable Editor, save that MAT-file if you want the changes to be saved. For instructions, see "Saving the Current Workspace" on page 5-5.

Cut, Copy, Paste, and Clear Contents in the Variable Editor

You can cut or copy selected elements, rows, and columns in an array and paste them to another position in that or another open array. To select a column or row, click the row or column header (the element that shows the row or column number). Use **Shift**+click to choose contiguous elements, rows, or columns in the array, or **Ctrl+A** to select all elements. For the cut, copy, and paste operations, use the **Edit** menu, the context menu, or the toolbar buttons. You can undo the last operation you performed in the Variable Editor.

When you cut elements, the value of each element you cut becomes 0 if numeric or [ ] if a cell array. After cutting, select the elements whose value you want to replace with the cut elements and then choose **Edit > Paste**. If the shape of the elements you cut differs from the shape of the elements into which you are pasting, the Variable Editor pastes all the elements, either by expanding the selection to be pasted into, or by expanding the array size to allow all the elements to be pasted. Pasting copied elements is the same as pasting cut elements, but the elements copied maintain their value rather than becoming 0 or [ ].

To make the value of elements 0, select elements, rows, or columns and then select **Edit > Clear Contents**. This differs from a performing a **Cut** because the data from the selected elements does not move to the clipboard; any clipboard content is unaffected by **Clear Contents**.

Example Copying and Pasting Array Elements

In this example, two elements are copied. The selected area for pasting is only one element, but two elements are replaced.
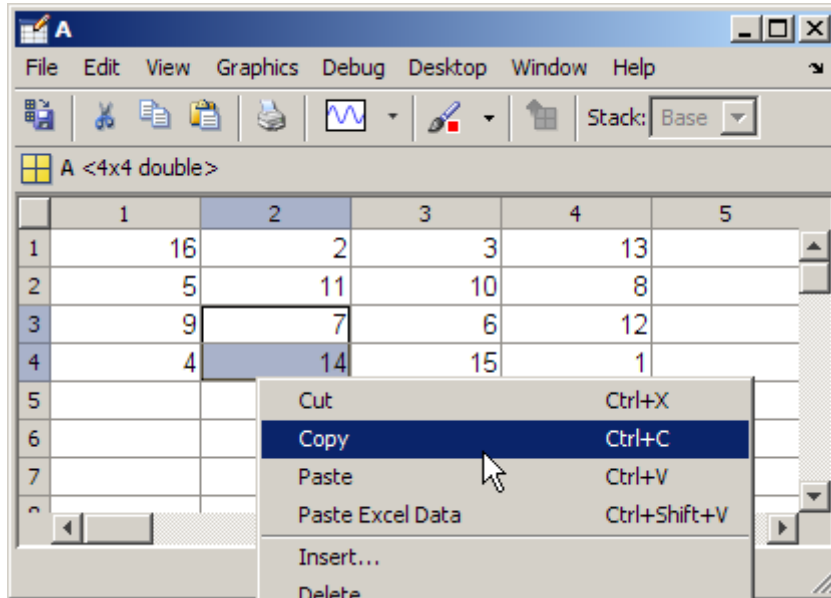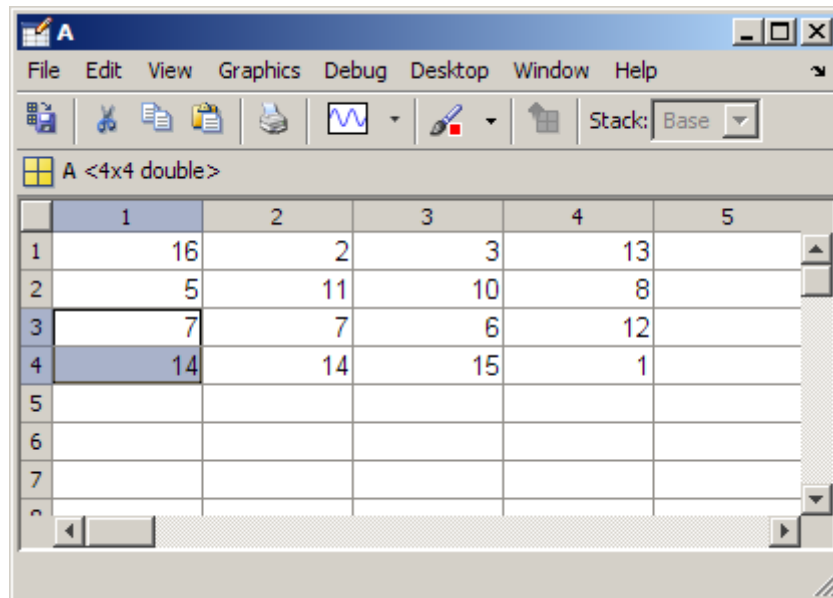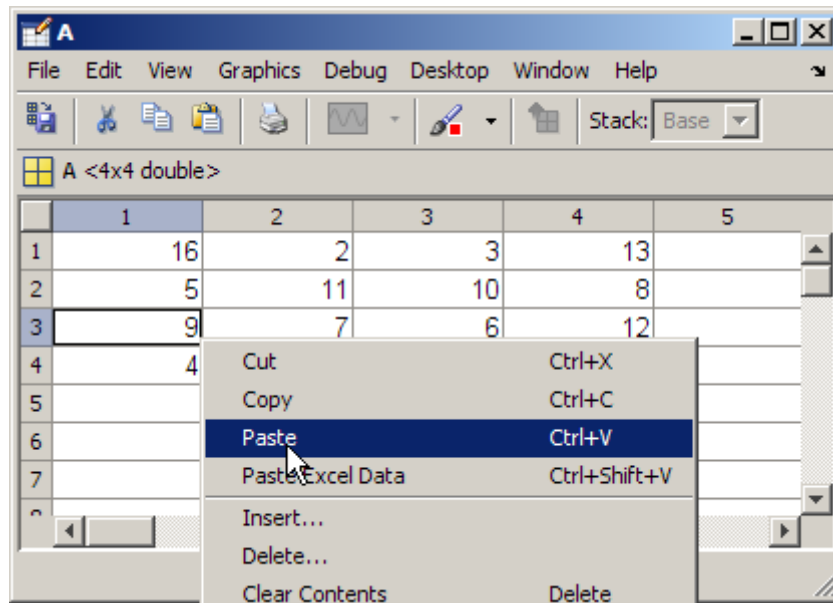
Example Cutting and Pasting Array Elements

In this example, two rows are selected for cutting. One row is selected for pasting. The Variable Editor expands the array size, adding a row, so all cut elements can be pasted. The value of the cut elements cut becomes 0.
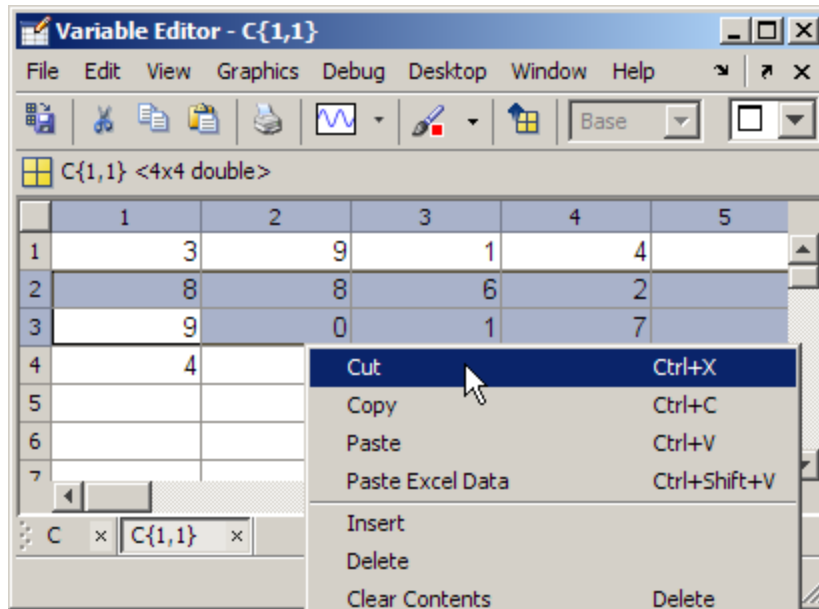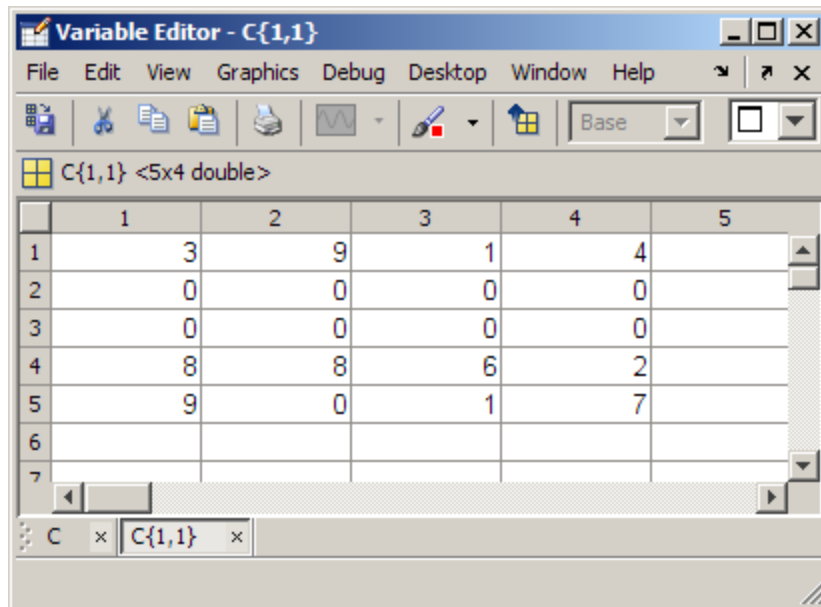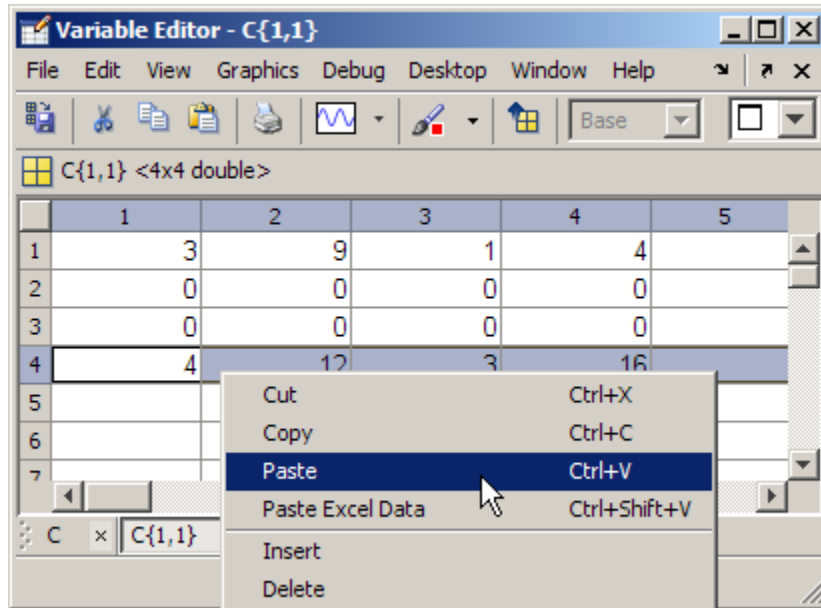
Insert and Delete in the Variable Editor

You can insert and delete elements, rows, and columns in arrays in the Variable Editor. When you select **Edit > Insert**, or **Edit > Delete**, a dialog box appears in which you specify rows, columns, or elements; for elements, the Variable Editor prompts you to provide, the direction for shifting existing elements.

Undo and Redo in the Variable Editor

You can undo the last action you performed in the Variable Editor, or redo a change after choosing undo. Select **Edit > Undo** or **Edit > Redo**. The actions supported are a change to a value you make by editing it in the Variable Editor, cutting, pasting, inserting, deleting, clearing contents, and pasting data from the Microsoft Excel application.

Exchanging Data with the Command Window

You can copy data from an array in the Variable Editor and paste it into the Command Window. You can also copy a value from the Command Window and paste it into an element in the Variable Editor. Be sure the data types are compatible. For example, you cannot paste text from the Command Window into a numeric array in the Variable Editor.

Exchanging Data with the Microsoft Excel Application

You can cut or copy cells from the Microsoft Excel application and paste them into the Variable Editor—use **Edit > Paste from Excel**. You can also cut or copy elements from an array in the Variable Editor and paste them into the Excel® application.

Be sure the data types are compatible. For example, you cannot paste text from the Excel application into a numeric array in the Variable Editor.

Creating Graphs and Variables, and Data Brushing in the Variable Editor

You can create graphs from selected variables in the Variable Editor. To create a graph, select an element, row, or column in an array, and in the right-click context menu, choose the graph type. MATLAB presents allowable

options for the selected data. In some cases, MATLAB makes assumptions, such as using `cell2mat` to convert selected cell array data, which cannot be plotted directly. For more information, see "Plotting Process" in the MATLAB Getting Started Guide.

To create a new variable, select an element, row, or column in an array in the Variable Editor, right-click, and from the context menu, select **Create Variable from Selection**.

Use the Data Brushing Tool, accessible via its toolbar button , to mark specific observations (or ranges of them) on graphs, allowing you to remove or save them to new variables. If a variable you brush in the Variable Editor is plotted on a graph, and that figure has Linked Plot mode enabled (which connects a graph's `XData`, `YData` and `ZData` to its data sources in the current workspace), any brushing you do in the Variable Editor also appears in the graph(s) displaying that variable. Likewise, brushing a linked plot highlights the same observations in the Variable Editor. For more information, see "Data Brushing with the Variable Editor" in the MATLAB Data Analysis documentation and the reference pages for `brush` and `linkdata`.

Preferences for the Variable Editor

To set preferences for the Variable Editor, select **File > Preferences > Variable Editor**. The Preferences dialog box opens showing **Variable Editor Preferences**.

Format

Specify the default array output format of numeric values displayed in the Variable Editor. This affects only how numbers are displayed, not how MATLAB computes or saves them. For more information, see the reference page for `format`.

Editing

You can specify where the cursor moves to after you type an element and press **Enter**:

- If you want the cursor to remain at the element where you just typed, clear the **Move selection after Enter** check box.

• If you want the cursor to move to another element, select the **Move selection after Enter** check box, and then use **Direction** to specify how you want the cursor to move. For example, if you want the cursor to move right one element after you press **Enter**, select **Right**.

International Number Handling

You can specify how you want decimal numbers to be formatted when you cut or copy elements from the Variable Editor and paste them into text files or other applications. The **Decimal separator to use when copying** edit field is by default " . " (period). If you are working in or providing data to a locale that uses a different character to delimit decimals, type that character in this preference and click **OK** or **Apply**.

Search Path

In this section...

Overview of the Search Path

MATLAB software uses a *search path* to find M-files and other files related to MATLAB and MathWorks products. The search path is a subset of all the directories in your computer's file system, which allows MATLAB to efficiently access the files it needs to run. The search path is also referred to as the *path* and directories in the search path are *on the path*.

By default, files supplied with MATLAB and other MathWorks products are in directories that are on the search path. This default search path includes all relevant directories under *matlabroot*/toolbox, where *matlabroot* is the directory where MATLAB is installed.

Files you use with MATLAB that were *not* provided by The MathWorks, such as M-files you create, must be on the search path or in the current directory so MATLAB can find and run them. All files called by the files you run must also be in the current directory or in directories on the search path.

When you include a directory on the search path, you *add it to the path*. You must explicitly add subdirectories to the search path; a subdirectory is not on the search path just because its parent directory is on the search path. Adding directories to the search path is similar to performing an include or import in some other applications. The following illustration shows an example of how the MATLAB search path works.
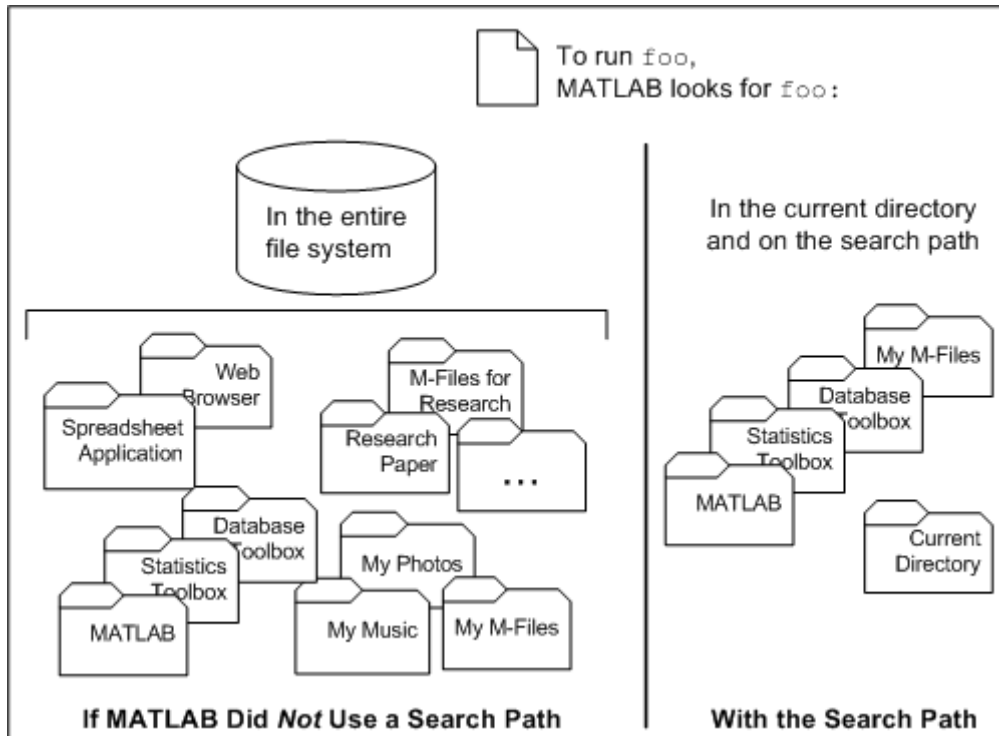
By default, the top of the search path includes a directory known as the userpath, intended as the place where you can store your files. The specific default userpath directory varies by platform.

MATLAB maintains search path information in the file pathdef.m. By default, pathdef.m is located in *matlabroot*/toolbox/local. You can save changes you make to the path for use in later sessions.

The order of directories on the path can be important. When there are two files with the same name, one in the current directory and the other in a directory on the search path, or both in directories on the search path, MATLAB follows rules to determine which file to run. If you get unexpected results, MATLAB might not be running the file you think it is, because another file of the same name is located in a directory that is higher up on the search path, or MATLAB is otherwise not interpreting the name the way you expect it to. For more information, see "Shadowed Functions and Name Clashes" on page 5-48.

How To Ensure MATLAB Software Can Access Files You Want To Use

Files you want to use in MATLAB must be in the current directory or in directories that are on the search path so MATLAB can access them. By default, files supplied with MathWorks products are in directories on the search path. Here are some ways of ensuring that MATLAB can access your files.

- "Determine If MATLAB Software Can Access a File" on page 5-37
- "Making Your Files Accessible to MATLAB Software" on page 5-39

Determine If MATLAB Software Can Access a File

- If you don't know if MATLAB can access a file, just try to use it. If it works successfully you do not have to do anything.
- If you use a file and experience problems, see "Running or Setting Breakpoints in a File From the Editor — File Not on Path or in Current Directory" on page 5-37 or "Running a File From the Command Window — File Not on Path or in Current Directory" on page 5-38.
- If you want to avoid problems, see "Before Using a File, Determine If MATLAB Software Can Access It" on page 5-38.

Running or Setting Breakpoints in a File From the Editor — File Not on Path or in Current Directory. When you try to run a file from the Editor/Debugger and the file is not in the current directory or in a directory on the search path, MATLAB cannot access the file. A dialog box notifies you about the problem and helps you to automatically correct it by prompting you to take one of the following actions:

- **Change Directory** — Makes the directory containing the file you want to run become the current directory

- **Add to Path** — Adds the directory containing the file you want to run to the top of the search path

To avoid this problem in the future, see "Before Using a File, Determine If MATLAB Software Can Access It" on page 5-38. See also the various ways of "Making Your Files Accessible to MATLAB Software" on page 5-39.

The dialog box will also notify you if there could be a *shadowing* problem, meaning the file you want to run or set breakpoints in has the same name as another file that is in the current directory or in a directory on the search path. When there are two files with the same name, be sure MATLAB accesses the file you want it to, and not the other file. The dialog box prompts you to run the file you want by changing the current directory. When the other file is not in the current directory, the dialog box also allows you to add the file's directory to the search path. For more information, see "Shadowed Functions and Name Clashes" on page 5-48.

Running a File From the Command Window — File Not on Path or in Current Directory. When you try to run a file from the Command Window and the file is not in the current directory or in a directory on the search path, MATLAB cannot access the file. Errors result, typically reporting a message like one of the following:

```
??? Undefined function or method 'some_name'
```

or

```
Cannot find function 'some_name'
```

To avoid this problem in the future, see "Before Using a File, Determine If MATLAB Software Can Access It" on page 5-38. See also the various ways of "Making Your Files Accessible to MATLAB Software" on page 5-39.

Before Using a File, Determine If MATLAB Software Can Access It.
Before you use a file, you can determine if MATLAB will be able to access it by using the which function. Run

```
which anyfilename
```

If MATLAB can access the file, it returns the full path name for it. You can use the file without a problem.

If MATLAB cannot find the file, it will not be able to use it. Instead of returning a full path name for the file, MATLAB returns this message:

```
'anyfilename' not found.
```

There are various ways to make the files accessible to MATLAB, as described in "Making Your Files Accessible to MATLAB Software" on page 5-39.

Making Your Files Accessible to MATLAB Software

There are different ways to ensure MATLAB can access files it needs when you do your work. This table advises about the most appropriate method for various situations. For background information, see "Overview of the Search Path" on page 5-35.

When...	You Should...
Your files are all in *one* directory.	Keep your files in the `userpath` directory. MATLAB automatically adds the `userpath` directory to the search path upon startup. The `userpath` directory can be the current directory upon startup. See "The userpath Directory" on page 5-48.
You *rarely* run the file, and it is not in a directory on the search path.	Move the file to a directory on the search path—see "Overview of Viewing and Changing the Search Path" on page 5-40. Or change the current directory to be the file's directory—see "Viewing and Changing the Current Directory" on page 5-57.
You rarely run the M-file *script* (M-file takes no input or output arguments)	Use the `run` function, specifying the relative pathname to the M-file.
The files you want to use are in *multiple* directories.	Add the directories to the search path—see "Adding Directories to the Search Path" on page 5-42

When...	You Should...
You *regularly* use files that are in multiple directories.	Add the directories to the search path —see "Adding Directories to the Search Path" on page 5-42.
	Then save the path in the MATLAB startup directory so you can use it again in future sessions or modify the search path upon startup. See "Saving Changes to the Search Path" on page 5-46 and "Automatically Modifying the Search Path at Startup" on page 5-44.
A file you want to use calls many files that are in multiple directories.	Determine where all of the called files are located—see "Displaying Dependencies Among M-Files" on page 7-15. Then add the directories to the search path—see "Adding Directories to the Search Path" on page 5-42.
You have files with the same name in multiple directories that are on the search path, or a file you run has the same name as a file supplied with MathWorks products.	Change the order of directories on the search path, or remove directories from the path. See "Moving Directories Within the Search Path" on page 5-46 and "Removing Directories from the Search Path" on page 5-44.
You want to work with the search path content in your files.	See "Programmatically Working with the Search Path" on page 5-52.

If you have problems related to the search path, troubleshoot the problem—see "Recovering from Problems with the Search Path" on page 5-51

Overview of Viewing and Changing the Search Path

Use the Set Path dialog box to view and change the MATLAB search path. As an alternative, you can view and change the search using functions.

To access the dialog box, select **File > Set Path**, or type `pathtool` in the Command Window and press **Enter**. The Set Path dialog box opens.

The Set Path dialog box lists all of the directories on the search path. The top of the list is the start of the search path, while the bottom of the list is the end. As an alternative, use the `path` function to list all directories on the search path.

Make changes to the search path.

User portion of search path, specified by `userpath` function.

List of directories on the search path.

Set Path

All changes take effect immediately.

MATLAB search path:

Add Folder...

Add with Subfolders...

Move to Top

Move Up

Move Down

Move to Bottom

Remove

- H:\Documents\MATLAB
- C:\Program Files\MATLAB\R2009a\toolbox\matlab\general
- C:\Program Files\MATLAB\R2009a\toolbox\matlab\ops
- C:\Program Files\MATLAB\R2009a\toolbox\matlab\lang
- C:\Program Files\MATLAB\R2009a\toolbox\matlab\elmat
- C:\Program Files\MATLAB\R2009a\toolbox\matlab\elfun
- C:\Program Files\MATLAB\R2009a\toolbox\matlab\specfun
- C:\Program Files\MATLAB\R2009a\toolbox\matlab\matfun
- C:\Program Files\MATLAB\R2009a\toolbox\matlab\datafun
- C:\Program Files\MATLAB\R2009a\toolbox\matlab\polyfun
- C:\Program Files\MATLAB\R2009a\toolbox\matlab\funfun
- C:\Program Files\MATLAB\R2009a\toolbox\matlab\sparfun
- C:\Program Files\MATLAB\R2009a\toolbox\matlab\scribe
- C:\Program Files\MATLAB\R2009a\toolbox\matlab\graph2d

Save   Close   Revert   Default   Help

Save changes for use in a future session.

Use changes in current session, but do not save for use in a future session.

Change search path to include only directories installed with MATLAB and related products.

See these topics about performing tasks using the Set Path dialog box or equivalent functions:

- "Adding Directories to the Search Path" on page 5-42

- "Removing Directories from the Search Path" on page 5-44
- "Moving Directories Within the Search Path" on page 5-46
- "Saving Changes to the Search Path" on page 5-46
- "Recovering from Problems with the Search Path" on page 5-51

See these topics for background information about the search path:

- "Overview of the Search Path" on page 5-35
- "How To Ensure MATLAB Software Can Access Files You Want To Use" on page 5-37

Adding Directories to the Search Path

Add directories to the search path when you want to run M-files in those directories using any of these methods:

- "Using the Set Path Dialog Box to Add Directories to the Path" on page 5-42
- "Using the Current Directory Browser to Add Directories to the Path" on page 5-43
- "Function Alternative for Adding Directories to the Path" on page 5-43
- "Automatically Modifying the Search Path at Startup" on page 5-44
- "When Not to Add Directories to the Path" on page 5-44

Using the Set Path Dialog Box to Add Directories to the Path

1 Select **File** > **Set Path** to open the dialog box.

2 Click one of these options:

- **Add Folder** to add only the selected directory but do not want to add all of its subdirectories.
- **Add with Subfolders** to add the selected directory and all of its subdirectories.

The Browse for Folder dialog box opens.

3 In the Browse for Folder dialog box, use the view of your file system to select the directory to add, and then click **OK**.

MATLAB adds the selected directory, and subdirectories if specified in the previous step, to the top of the search path.

4 If you want to position the newly added directory somewhere other than the top of the search path, see "Moving Directories Within the Search Path" on page 5-46.

Note The `userpath` directory automatically moves to the top of search path the next time you start MATLAB. For more information, see "The userpath Directory" on page 5-48.

---

5 If you want to use the newly modified search path in future sessions, click **Save**. For more information about saving the path, see "Saving Changes to the Search Path" on page 5-46.

If you do not save the changes, the directories you added remain on the search path until you end the current session of MATLAB.

6 Click **Close**.

Using the Current Directory Browser to Add Directories to the Path

In the Current Directory browser, select a directory, right-click, and select **Add to Path** from the context menu. Then select one of the submenus, for example, **Selected Folder and Subfolders**.

Function Alternative for Adding Directories to the Path

To add directories to the top of the search path, use `addpath`. Then, if you want to use the newly modified search path in future sessions, run `savepath`—see "Saving Changes to the Search Path" on page 5-46for more information.

You can include `addpath` statements in your `startup.m` file to automatically modify the path when MATLAB starts. For details, see "Automatically Modifying the Search Path at Startup" on page 5-44.

Automatically Modifying the Search Path at Startup

By default, upon startup, MATLAB automatically adds the userpath
directory to the top of the search path. On UNIX[16] platforms, including Apple
Macintosh platforms, you can instruct MATLAB to also add subdirectories
of the userpath to the search path upon startup by using the MATLABPATH
system environment variable. For more information, see the userpath
reference page.

On all platforms, you can instruct MATLAB to automatically modify the
search path upon startup by including addpath, rmpath, and path statements
in a startup.m file. If you use MATLAB on different platforms, you can use
separate path sections in the startup.m file for each platform, if needed, with
each section preceded by an ispc or isunix statement.

As an alternative to modifying the path in a startup.m file, you can also save
changes you make to the search path for use in future sessions. For more
information, see "Saving Changes to the Search Path" on page 5-46.

When Not to Add Directories to the Path

These are some examples of when you do not need to or should not add
directories to the search path:

- Instead of adding a directory to the path, you can make the directory be the
  current directory in MATLAB. Do this when you seldom run files in that
  directory so there is little benefit to having it on the path.

- Do not add method directories (directories that start with @) and private
  directories to the search path.

- Do not add to the path directories supplied by The MathWorks, located in
  matlabroot/toolbox, and similarly, do not add the userpath directory.
  These directories are already on the default search path.

Removing Directories from the Search Path

Remove directories from the search path when you no longer use the files in
the directories. Another reason to remove directories is to avoid potential
name clashes—for more information, see "Shadowed Functions and Name

---

16. UNIX is a registered trademark of The Open Group in the United States and other
    countries.

Clashes" on page 5-48. Remove directories from the path using any of these procedures:

- "Using the Set Path Dialog Box to Remove Directories from the Path" on page 5-45

- "Function Alternative for Removing Directories from the Path" on page 5-45

Using the Set Path Dialog Box to Remove Directories from the Path

1 Select **File > Set Path** to open the dialog box.

2 Select the directories to remove.

> **Note** If you remove the userpath directory from the search path and save the changes to the path, it also automatically clears the value for userpath, which could impact the startup directory. For more information, see "The userpath Directory" on page 5-48.

3 Click **Remove**. The directories are removed from the path.

4 If you want to use the newly modified search path in future sessions, click **Save**. For more information about saving the path, see "Saving Changes to the Search Path" on page 5-46.

5 Click **Close**

Function Alternative for Removing Directories from the Path

To remove directories from the search path, use rmpath. Then, if you want to use the newly modified search path in future sessions, run savepath—see "Saving Changes to the Search Path" on page 5-46 for more information.

You can include rmpath statements in your startup.m file to automatically modify the path when MATLAB starts. For related information, see "Automatically Modifying the Search Path at Startup" on page 5-44.

Moving Directories Within the Search Path

The order of directories on the search path is relevant — for more information, see "Shadowed Functions and Name Clashes" on page 5-48.

To modify the order of directories within the search path, you can use the Set Path dialog box. As an alternative, you can use the `path` function to move a directory to the top or bottom of the search path.

To use the Set Path dialog box to modify the order of directories on the search path, follow this procedure:

1 Select **File > Set Path** to open the Set Path dialog box.

2 Select the directory or directories you want to move.

3 Click one of the **Move** buttons, such as **Move to Top**. The order of the directories changes.

> **Note** The `userpath` directory automatically moves to the top of search path the next time you start MATLAB. For more information, see "The userpath Directory" on page 5-48.

4 If you want to use the newly modified search path in future sessions, click **Save**. For more information about saving the path, see "Saving Changes to the Search Path" on page 5-46.

5 Click **Close**.

Saving Changes to the Search Path

- "Overview of Saving the Search Path" on page 5-46
- "How and Where to Save the Search Path" on page 5-47

Overview of Saving the Search Path

When you make changes to the search path, they remain in effect during the current session of MATLAB. If you want MATLAB to use the changed

search path in subsequent sessions, you need to save the path, which is the
`pathdef.m` file, in a location where MATLAB can find it upon the next startup.

You can also automatically reflect your changes to the search path each
session by using a `startup.m` file that contains `addpath` statements. For
more information, see "Automatically Modifying the Search Path at Startup"
on page 5-44.

How and Where to Save the Search Path

You can save changes you make to the search path using the Set Path dialog
box **Save** button, or using the `savepath` function. You can include a `savepath`
statement in a `finish.m` file to instruct MATLAB to automatically save the
current path when you exit MATLAB.

The *default search path* is located in *matlabroot*/`toolbox/local`.
When you save the search path, if you do not have write access to
matlabroot/`toolbox/local`, MATLAB prompts you for a different location.
The recommended location is your MATLAB startup directory, because
MATLAB will automatically use the search path at the next startup. If you *do*
have write access to *matlabroot*/`toolbox/local` but want to save the search
path to your startup directory, use the `savepath` function.

If you have made changes in the Set Path dialog box, but want to undo them,
click **Revert**. You can then click **Close** or **Save**.

If you want to restore the default search path, in the Set Path dialog box,
select **Default**. You can then click **Close** or **Save**. Alternatively, you can use
the `restoredefaultpath` function.

When you upgrade to a new release of MATLAB, you cannot use your
existing `pathdef.m` file. MATLAB uses the default search path for the new
release. Add directories to it, and then save it in the startup directory. As
an alternative, consider modifying your search path via path statements in a
`startup.m` file, which can be used across releases—for more information, see
"Automatically Modifying the Search Path at Startup" on page 5-44.

The userpath Directory

By default, the top of the search path includes a user portion called the userpath. Its default value varies by platform. By default, the userpath directory is automatically added to the search path upon startup. The userpath directory is also the default startup directory on Microsoft Windows platforms, and can be designated as the startup directory on UNIX platforms, including the Macintosh platform.

You can change the userpath to be a different directory than the default, or specify that it not be used. There are some special properties of the userpath directory:

- If the userpath directory is also the startup directory, when you remove the userpath directory from the path, the userpath directory is no longer the startup directory.

- When you move the userpath directory within the search path, it maintains the new position during the current session, but moves to the top of the path at the start of the new session.

You make changes to the userpath via the userpath function. For more information, including examples, see the userpath reference page, and "Changing the Startup Directory Via the userpath Function" on page 1-14.

Shadowed Functions and Name Clashes

What Are Shadowed Functions and Name Clashes?

When MATLAB tries to run a file, it looks for the file in the current directory and in directories on the search path. When there are two or more files with the same name, one in the current directory and the others in directories on the search path, MATLAB runs the file in the current directory.

When there are two or more files with the same name, each located in a different directory on the search path, MATLAB runs the file located in the directory closest to the top of the search path. MATLAB considers the other files of the same name to be *shadowed*. For this reason, the order of the directories on the search path is relevant. Some functions provided with MATLAB and other products from The MathWorks have the same name; these are called overloaded functions.

In addition to files on the search path or in the current directory, other language elements in MATLAB might have the same name as an M-file. For example, if there is a variable named `foo`, and there is a function named `foo` is in a directory on the search path, when you type `foo` in the Command Window, MATLAB interprets `foo` as the variable rather than the M-file.

These situations are sometimes referred to as *namespace conflicts* or *name clashes* and can be the source of unexpected results or errors. There are other potential name conflicts such as with subfunctions and MEX-files—for more information, see "Precedence Rules" in MATLAB Programming Tips documentation.

Addressing Name Clashes

Here are some guidelines for avoiding and dealing with name clashes so that MATLAB runs the file you want it to run:

- If you know of a possible name clash and are not sure of the function MATLAB will use, run `which` for a specified function name and MATLAB returns the full path to the function it will use.

- If you have a name clash, change the current directory in MATLAB to the directory containing the file you want MATLAB to run.

  As an alternative, move the directory containing the shadowed function to the top of the search path, or anywhere ahead of the other function with the same name. Or, remove from the search path the directory containing the function you do not want to run.

- For M-file scripts, you can use `run` with the full pathname for the M-file. For example, use `run d:/mymfiles/foo.m` to ensure that version of `foo` runs.

- Avoid naming your files and variables with the same names as functions and blocks supplied with MathWorks products. If you use files created by other users of MATLAB, or if you provide files for use with MATLAB to other users, there could be name clashes. For example, if you have MATLAB and no other products, and you give a file named `driver.m` to another user, that user might experience a name clash with the `driver` function in the Database Toolbox software if the other user has that product.

  When you install new versions of products from The MathWorks, there might be new functions or blocks that have the same name as some of your

existing files and variables. When you review the release notes for the products you use, look for potential name clashes.

- If you organize your projects using directories, where one directory contains all the files for a given project, and you have files used by more than one project, avoid making a copy of the file for each project. Instead, maintain a single version of the file in one location. Consider using a directory just for shared files and include that directory on the search path.

The following illustration shows an example of a name clash and some ways to resolve it.
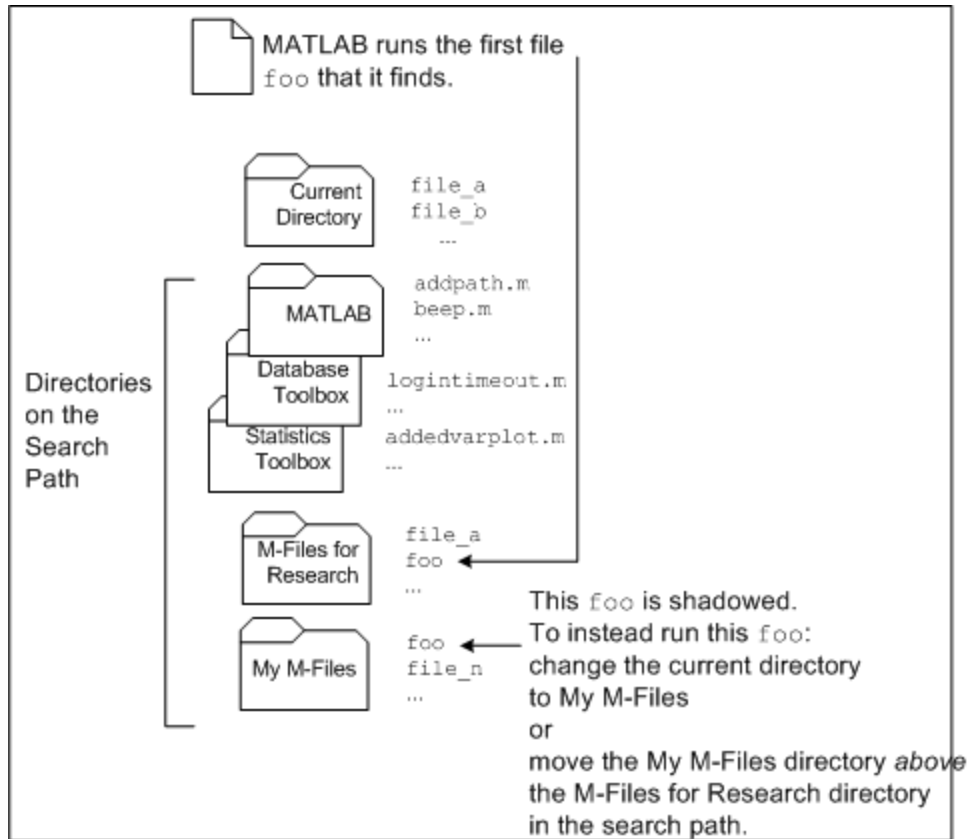
Recovering from Problems with the Search Path

If you get unexpected results that are related to the search path, you can try to correct the path file or restore the default path. You might experience path problems if you save the path on a Windows platform and then try to use the same `pathdef.m` file on a UNIX platform. Similarly, you might experience problems if the path file becomes corrupt, invalid, renamed, or lost.

For example, if an error message similar to the following appears when you start MATLAB

```
Warning: MATLAB did not appear to successfully set the search
path...
```

it indicates a problem with the search path and you will not be able to use MATLAB successfully.

To recover from problems with the search path, try the following, in order, proceeding to the next step only if needed:

1 Ensure MATLAB is using the path file you expect—run

```
which pathdef
```

2 In a text editor, view the `pathdef.m` file and the `startup.m` file, looking for obvious problems. Make changes and save them. If path problems appear to be resolved, start MATLAB again to be sure the problem does not reappear. Depending on the problem, you might not be able to even view the `pathdef.m` file.

3 Use the default search path for products from The MathWorks, located in *matlabroot*/toolbox/local. To use it, open the Set Path dialog box, select **Default**, then **Save**, then **Close**. Depending on the problem, you might not be able to even open the dialog box.

4 Run `restoredefaultpath`. This sets the search path to include only directories for installed products from The MathWorks and stores it in *matlabroot*/toolbox/local. If that seems to have corrected the problem, run `savepath`. Start MATLAB again to be sure the problem does not reappear.

Depending on the problem, this might generate a message such as

```
The path may be bad. Please save your work (if desired), and quit.
```

If so, perform step 5.

5 Perform these steps after trying step 4.

   a Run

```
restoredefaultpath; matlabrc
```

This might run for a few minutes. It sets the search path to include only directories for installed products from The MathWorks and corrects path problems encountered during startup.

   b If there is a `pathdef.m` in your startup directory for MATLAB, it caused the problem. So either remove the bad `pathdef.m` file or replace it with a good `pathdef.m` file—for example, run this

```
savepath('path_to_your_startup_directory/pathdef.m')
```

   c Start MATLAB again to be sure the problem does not reappear.

For background information, see "Overview of the Search Path" on page 5-35.

Programmatically Working with the Search Path

If you need to modify the search path in your programs, use the search path functions, and consult the following advanced topics:

- "The pathdef.m File" on page 5-52
- "Subdirectories on the Path" on page 5-53
- "Partial Pathnames" on page 5-53
- "Caution Against Saving Files in matlabroot/toolbox" on page 5-54

The pathdef.m File

The file `pathdef.m` is a string listing of the directories in the MATLAB search path. The character that separates the directories in the string is a semicolon (;). In versions of MATLAB prior to Version 7.7 (R2008b), MATLAB instead use a colon (:) as the path separator on UNIX platforms. Use the `pathsep` function to determine the character being used.

For example, if you are using MATLAB version 7.8 (R2009a), when you run

```
pathdef
```

MATLAB displays, for example

```
ans =

H:\My Documents\MATLAB;C:\Program Files\MATLAB\R2009a\toolbox\matlab\general;
C:\Program Files\MATLAB\R2009a\toolbox\matlab\ops;C:\Program Files\MATLAB\R20
09a\toolbox\matlab\lang;C:\Program Files\MATLAB\R2009a\toolbox\matlab\elmat;C
:\Program Files\MATLAB\R2009a\toolbox\matlab\elfun;...
```

When you run `path`, MATLAB displays the information with one directory per line:

```
MATLABPATH

 H:\My Documents\MATLAB
 C:\Program Files\MATLAB\R2009a\toolbox\matlab\general
 C:\Program Files\MATLAB\R2009a\toolbox\matlab\ops
 C:\Program Files\MATLAB\R2009a\toolbox\matlab\lang
 C:\Program Files\MATLAB\R2009a\toolbox\matlab\elmat
 C:\Program Files\MATLAB\R2009a\toolbox\matlab\elfun
...
```

where the line only wraps when it reaches the width of the Command Window.

Subdirectories on the Path

Use `genpath` to form a path string consisting of all the directories below the specified directory. The `genpath` function is useful in conjunction with the `addpath` function, when you want to add all of a directory's subdirectories to the search path.

Partial Pathnames

Some functions allow you to use a partial pathname, meaning when you use the functions, you do not have to specify a full path name. One example of when this is useful is when you do not want to use a specific value for the `matlabroot` directory. For more information, see `partialpath`.

Caution Against Saving Files in matlabroot/toolbox

Save any M-files you create and any M-files supplied with products from The MathWorks that you edit in a directory that is not in the *matlabroot*/toolbox directory tree. The exception is the pathdef.m file, which you can change and save in its default location, *matlabroot*/toolbox/local. If you keep your files in *matlabroot*/toolbox directories, they can be overwritten when you install a new version of MATLAB. Also note that locations of files in the *matlabroot*/toolbox directory tree are loaded and cached in memory at the beginning of each session of MATLAB to improve performance. If you save files to *matlabroot*/toolbox directories using an external editor or add or remove in from these directories using file system operations, run rehash toolbox before you use the files in the current session. If you make changes to existing files in *matlabroot*/toolbox directories using an external editor, run clear functionname before you use the files in the current session. For more information, see rehash or "Toolbox Path Caching in the MATLAB Program" on page 1-22.

Managing Files and Working with the Current Directory

Overview of Managing Files with the MATLAB Software

The MATLAB software uses the current directory and the MATLAB search path as reference points when it accesses files. If a file you want to run is not in a directory on the search path, you can run it by changing the current directory to the file's directory. The primary way you view and change the current directory, as well as manage, organize, and find files in MATLAB, is with the Current Directory browser desktop tool. You can use functions as an alternative.

Using the Current Directory Browser to Manage Files

To search for, view, open, and make changes to directories and files related to MATLAB, you can use the Current Directory browser. The Current Directory browser does not replace your operating environment's file system, but provides an alternative interface to it within MATLAB. The Current Directory browser provides features especially for working withMATLAB files. You can also use the Current Directory browser reports to help you tune and manage your M-files—for details, see "Using M-File Reports" on page 7-2.

To open the Current Directory browser, select **Desktop > Current Directory** from the MATLAB desktop, or type `filebrowser` in the Command

Window. The Current Directory browser opens. Change its size or the location as you would for any tool in the MATLAB desktop—for details, see "Opening and Arranging Desktop Tools" on page 2-5.

View and change the current directory using the address bar. Use the Actions button ⚙ on the toolbar to perform some common file operations. You can customize the toolbar using Toolbars Preferences—for details, see "Setting Toolbars Preferences for Desktop Tools" on page 2-138. You can also use the Current Directory browser to view the current directory's contents, that is, the subdirectories and files in the current directory.

Right-click the toolbar and select **Customize** to add buttons.

Select operations from the Actions button.

View subdirectories and change the current directory using the address bar.

Name △	Date Modified
html	1/22/07 6:46 PM
mydemos	4/28/08 12:54 PM
profile_results	6/5/06 3:37 PM
sea_temp	5/8/08 5:43 PM
temp	5/8/08 5:45 PM
collatz.m	7/18/07 4:17 PM
collatz_new.m	1/17/08 2:43 PM
collatzall.m	6/6/07 2:54 PM
collatzplot.m	6/6/07 4:12 PM
collatzplot_new.m	6/18/07 10:59 AM
dec_12_data.mat	7/13/06 12:08 PM
hs_err_pid440.log	
lengthofline.asv	1/17/07 11:40 PM
lengthofline.m	1/18/08 2:42 PM
lengthofline2.m	7/16/07 3:23 PM
lengthofline_orig.m	1/17/07 6:27 PM

Current Directory

File Edit View Debug Desktop Window Help

I: ▸ my_matlab_files ▸ my_mfiles ▸

Details

Select a file to view details

Functions for Managing Files

You can also use functions to manage your files in MATLAB. For example, you can delete a file by selecting it in the Current Directory browser and pressing the **Delete** key, or you can use the delete function. For each Current Directory browser feature, the documentation includes the corresponding function equivalent, and includes links to the function reference page for details. See also a summary of the functions.

When you run functions whose arguments require the use of a path name or file name, and the path name or file name includes spaces, use the function form rather than the command form of the syntax. For example, the command form

```
delete my file.m
```

generates a warning and does not delete my file.m. Instead use the function form of the syntax:

```
delete('my file.m')
```

Permissions for File Management Operations

Typically, you cannot perform operations on files and directories for which you do not have write permission. There are some exceptions. For example, you cannot copy a file to a read-only directory using the Current Directory browser. However, you can do so using movefile with the appropriate option.

Viewing and Changing the Current Directory

- "Using the Current Directory Field to View and Change the Current Directory" on page 5-58
- "Using the Current Directory Browser to View and Change the Current Directory" on page 5-58
- "Specifying the Current Directory at Startup" on page 5-59
- "Functions for Viewing and Changing the Current Directory" on page 5-60

Using the Current Directory Field to View and Change the Current Directory

A quick way to view or change the current directory is by using the current directory field in the desktop toolbar.
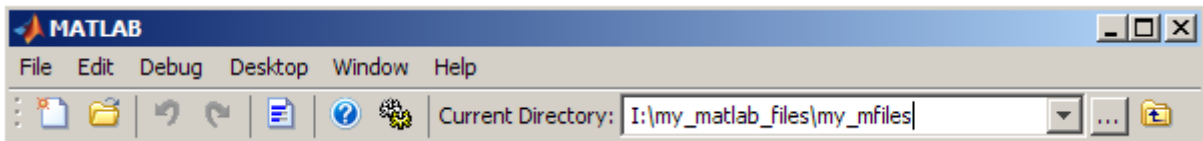
To change the current directory from this field, do one of the following:

- If you know or have copied the path for the new current directory, enter it in the field and press **Enter**.

- To use a previous current directory, click the down arrow, and then select an item from the history list. The directory you select becomes the current directory in MATLAB. The directories in the history are listed in reverse chronological order; the most recently used is at the top of the list. To clear the history list or set the number of directories saved in the list, see "Preferences for the Current Directory Browser" on page 5-88.

- To look for the new directory, click the **Browse for Folder** button.

- To move the current directory up one level, click the **Go Up One Level** button 📁.

Using the Current Directory Browser to View and Change the Current Directory

These are the common ways to view and change the current directory using the Current Directory browser:

- To make a subdirectory become the current directory, double-click it, or select the subdirectory and press **Enter** or **Return**.

- To change the current directory to a directory one level up in the directory structure, position the cursor in the contents listing portion of the Current Directory browser and press the backspace (<-) key. You can also use the go up one level button 📁 on the toolbar; if this button is not on the toolbar,

you can add it—for details, see "Setting Toolbars Preferences for Desktop Tools" on page 2-138.

- You can view subdirectories and change the current directory using the address bar. The address bar displays the full path to the current directory, but it's more than static text. Use the address-bar arrow buttons to view subdirectories, and then select a subdirectory to make it the current directory. Use the down-arrow button on the right side of the address bar to view previous current directories, and then select an entry to make it the current directory.
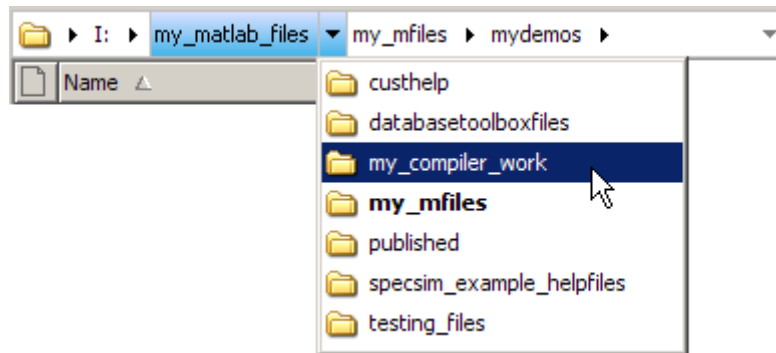
You can view and edit the full path to the current directory as a text string by clicking the empty area on the right side of the address bar. This is useful if you need to copy the path for use in a function, file, or another tool.
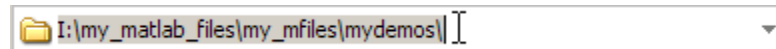
For details, right-click the empty area on the right side of the address bar and select **Help Using Address Bar** from the context menu.

- To change to a previous current directory, use the Back button ⬅ in the toolbar. After using the Back button, view the next current directory by clicking the Forward button ➡.

Specifying the Current Directory at Startup

You can specify the current directory for the MATLAB application when you start MATLAB. This directory is called the startup directory—for more information, see "Startup Directory for the MATLAB Program" on page 1-11.

Functions for Viewing and Changing the Current Directory

Function	Description
cd	Change the current directory.
pwd	View the current directory name.

Viewing the Contents of the Current Directory

- "Viewing Directories, Files, and Their Attributes in the Current Directory" on page 5-60

- "Viewing Information About the Selected File or Directory in the Current Directory Browser" on page 5-61

- "Sorting and Grouping Files and Directories in the Current Directory Browser" on page 5-63

- "Refreshing the View in the Current Directory Browser" on page 5-65

- "Functions for Viewing the Contents of the Current Directory" on page 5-66

Viewing Directories, Files, and Their Attributes in the Current Directory

In the Current Directory browser, you view the names of files and directories in the current directory. You can also view attributes for each item, which are shown in columns.

You can choose which attributes you want to see. To save space, you can hide attributes that are not relevant. To do so, select **View > Choose Columns**, and choose the attributes to view. Attributes you can show are icon (file type), size, date modified, and description.

The description is the same as the description in the details panel for a selected item. For more information, see "View Description in Details Panel" on page 5-62.

You can arrange columns by dragging a column header to a new position.

Use the View menu to show the **Description** column,
which helps you quickly locate a file of interest.
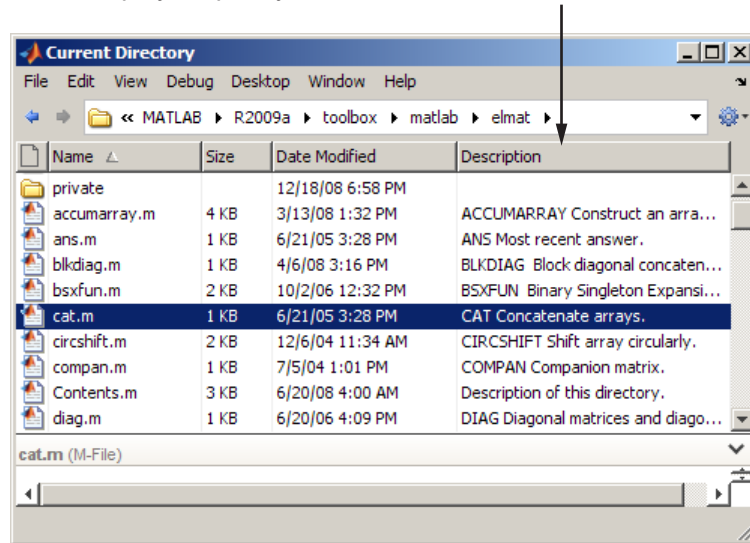
Viewing Information About the Selected File or Directory in the Current Directory Browser

For M-files, MAT-files, and Simulink models, if you want to view more information about a file, first select the file. Then view the additional information about the file in the **Details** panel in the bottom portion of the Current Directory browser.
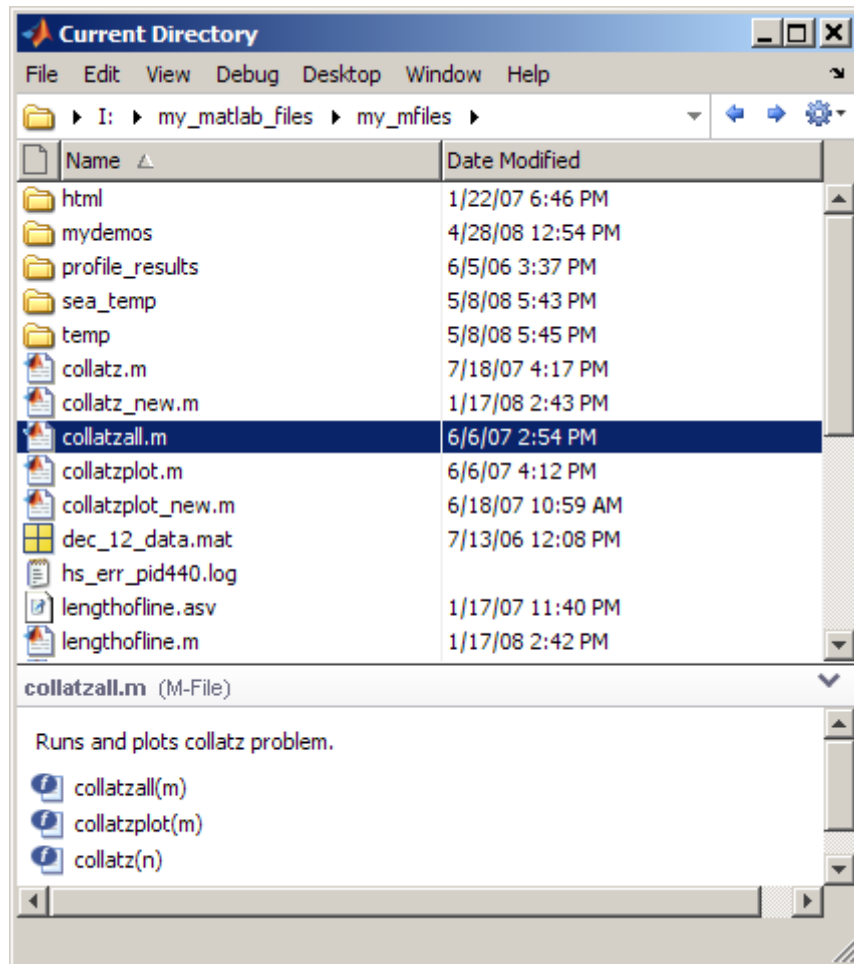
You can perform the following actions from the **Details** panel.

Resize the Details Panel. To see more or less information in the **Details** panel, change the height of the **Details** panel by dragging the separator bar up or down. Collapse the panel by clicking the down arrow in its title bar.

View Description in Details Panel. The description appears only for directories and files used with MathWorks products that have an associated description:

- For an M-file, the description is the first line of the help comments, also called the H1 line. To view more extensive help, press the **F1** key; the reference page for that function appears in the Help browser.

- For a Simulink model, the description is the one specified in the Description pane of the Model Properties dialog box. Viewing model descriptions in the Current Directory browser is useful because you do not have to use the Simulink software.

- For a directory, the description is the first comment line of the `Contents.m` file for the directory.

For more information about providing descriptions for your files and directories, see "Help for the Files You and Other Users Create" on page 4-65.

To view the description for all files in a directory, use the **Description** column. To see the column if it is not shown, select **View > Choose Columns > Description**.

View and Go To Subfunctions and Cells from Details Panel. For an M-file that contains subfunctions or cells, you see a list of the subfunctions and cells in the file. Double-click a subfunction or cell to open the M-file in the Editor at that subfunction or cell.

View and Go To Properties and Methods in Class M-Files from Details Panel. For a class M-file, you see the properties and methods in the file. Double-click a property or method to open the class M-file in the Editor at that property or method definition.

View and Load MAT-File Variables from Details Panel. For a MAT-file, you see its variables, and the class and value for each variable, similar to the view in the Workspace browser. You can select a variable in the MAT-file and load it into the workspace by dragging it to the Workspace browser.

Sorting and Grouping Files and Directories in the Current Directory Browser

To help you organize, find, and manage your MATLAB and related files and directories (folders), you can sort and group items by their attributes.

Sorting in the Current Directory Browser. You can sort the contents of a directory by any attribute. The attribute currently sorted on includes an arrow indicator in the column header. To change the attribute sorted on and specify ascending or descending sort order, use **View > Sort By**. To minimize space, you can sort by an attribute without showing the column for that attribute.

Grouping in the Current Directory Browser. To see related items together, you can group them using **View > Group By**. You cannot group by name or description. To minimize space, you can group by an attribute without showing the column for that attribute.

With grouping, the view changes to show all items with that attribute arranged in a group. Each group has a heading for the attribute value, for example **ASV File**, **M-File**, and so on. You can hide the items in a group by clicking the collapse button (–) next to the group's heading.

After grouping by an attribute, you can sort by a different attribute. The sort applies to items within each group. This is a two-level sort.

To stop grouping by an attribute, select **View > Group By**, and then select **Stop Grouping**, or select a different attribute to group by.

Similar to grouping is filtering, which is useful, for example, when you want to see *only* files of a certain type, for example, only M-files. For more information, see "Filtering the Current Directory Browser View to Find Files and Directories" on page 5-79.

The following figure illustrates grouping by type, with some groups collapsed.
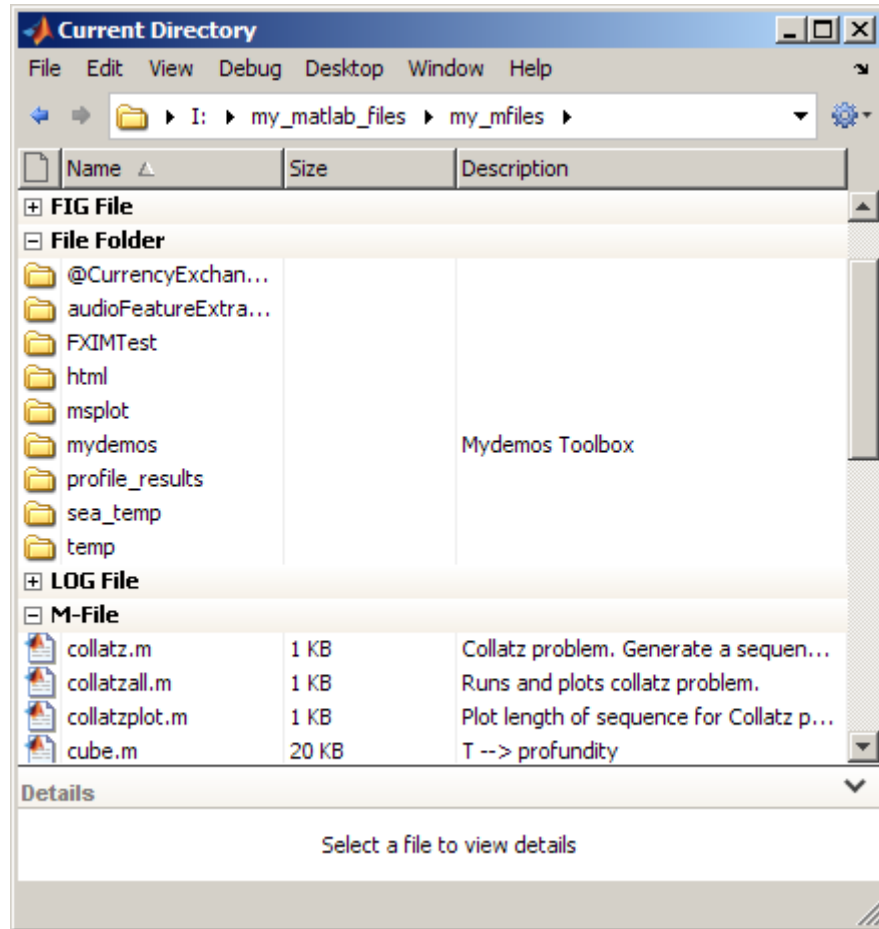
Refreshing the View in the Current Directory Browser

By default, the Current Directory browser automatically reflects changes to files and directories that you make in your operating environment. In rare situations, frequent refreshing can slow performance in MATLAB. To improve the performance, use preferences to control how often refreshing occurs. To manually refresh the view at any time, right-click in the contents listing portion of the Current Directory browser, and select **Refresh** from the context menu.

To specify the refresh preferences, select **File > Preferences > Current Directory**. For more information, click the **Help** button in the preferences panel.

Functions for Viewing the Contents of the Current Directory

Function	Description
dir	View the contents of the current working directory or another specified directory. Use a return argument with dir to get a structure containing file attributes, such as the names and last modified dates.
fileattrib	View or set file attributes. fileattrib is similar to attrib on DOS or chmod on UNIX[17] platforms.
what	Display files related to MATLAB that are in the current directory.

Performing MATLAB Operations in the Current Directory Browser

- "Running Files from the Current Directory Browser" on page 5-67

- "Opening Files from the Current Directory Browser" on page 5-67

- "Creating New Files Using the Current Directory Browser" on page 5-69

- "Creating and Updating MAT-Files in the Current Directory Browser" on page 5-70

- "Accessing Source Control Interface Features from the Current Directory Browser" on page 5-71

- "Locating a File or Directory on Disk" on page 5-71

- "Adding Directories to the Search Path from the Current Directory Browser" on page 5-72

---

17. UNIX is a registered trademark of The Open Group in the United States and other countries.

- "Accessing the File and Directory Comparison Tools from the Current Directory Browser" on page 5-72
- "Importing Data Using the Current Directory Browser" on page 5-72

Running Files from the Current Directory Browser

Running files from the Current Directory browser is convenient because they do not need to be on the search path. You can run an M-file from the Current Directory browser if it does not require input arguments. Select the file, right-click, and select **Run File** from the context menu. The results appear in the Command Window.

You can run a Microsoft Windows shortcut directly from the Current Directory browser. Double-click the shortcut icon in the Current Directory browser to perform the Windows operation.

Opening Files from the Current Directory Browser

- "Opening Files In MATLAB from the Current Directory Browser" on page 5-67
- "Opening Files Outside of MATLAB from the Current Directory Browser" on page 5-69
- "Functions for Opening Files" on page 5-69

Opening Files In MATLAB from the Current Directory Browser. You can use the Current Directory browser to open a file and view or change its contents. The file opens in the MATLAB tool associated with that file type.

Select a file or files and perform one of the following actions to open it:

- Press the **Enter** or **Return** key.
- Right-click and select **Open** from the context menu.
- Double-click the file.

The file opens in the appropriate tool, provided that the tool is installed on your system.

To open any file type in the Editor select the file, and then from the context menu, select **Open as Text**.

The following table summarizes where common file types open.

File Type	Extension	Action
Figure file	`fig`	Opens the file in a figure window. Alternatively, from the context menu, you can select **Open in GUIDE**.
HTML file	`html`	Opens the file in the MATLAB Web browser.
M-file	`m`	Opens the file in the Editor.
MAT-file	`mat`	Loads the data into the workspace. To load only selected variables from a MAT-file, select a MAT-file in the Current Directory browser contents list. In the **Details** pane, select a variable in the MAT-file. Drag the variable to the Workspace browser. The variable loads into the base workspace.
Model	`mdl`	Opens the file in the Simulink application.
P-file	`p`	You cannot open P-files.
Variable	none	Opens the numeric or string array in the Variable Editor.
Other	`custom`	Opens the file by calling the helper function `opencustom`, where `opencustom` is a user-defined function.

Opening Files Outside of MATLAB from the Current Directory Browser. To open a file using an external application, select the file and then select **Open Outside MATLAB** from the context menu. For example, if you right-click myfile.doc and select **Open Outside MATLAB**, myfile.doc opens in the Microsoft Word application (assuming that you have the .doc file association configured to start the Microsoft Word application). This is useful for file types associated with MATLAB software that are also used with an external application on the Windows platform. For example, .mat is the extension for MATLAB data files as well as for Microsoft Access files. When you double-click a .mat file in the Current Directory browser, it loads the MATLAB data file into the workspace. If, instead, you want to open the .mat file in the Microsoft Access application, right-click it and select **Open Outside MATLAB** from the context menu. MATLAB opens the file using the application you associated with that file type on the Windows platform. For more information, see "Changing File Associations for the MATLAB Program from the Windows Environment" on page 1-6.

Functions for Opening Files. To open files using functions instead of the Current Directory browser, use the following functions.

Function	Description
open	Open a file in the tool appropriate for the file, based on its file extension. Default behavior is provided for standard MATLAB file types. You can create your own variation of open for any file type, and you can override the default behavior for the standard files.
winopen	Open a file using an external application on Windows platforms.
type	View the content of an ASCII file, such as an M-file, in the Command Window.

Creating New Files Using the Current Directory Browser

For certain file types, you can create a new file based on a template that guides you to include typical elements. To create a new file using the Current Directory browser:

1 Navigate to the directory where you want to create the new file.

2 Click the Actions button and select **New File**, and then select one of the following:

- **Blank M-File** to create an empty M-file, such as for a script. You can also use this to create any text file, such as an `xml` or `html` file.

- **Function M-File** to create an M-file prepopulated with a basic M-file function structure.

- **Class M-File** to create a class definition M-file (`classdef`), prepopulated with a basic M-file class structure.

- **Model** to create a Simulink model file.

A new file named `Untitled1` appears in the list of files shown in the Current Directory browser, along with the appropriate extension and icon.

3 Type over `Untitled1` with the name that you want to give the new file.

For guidelines about naming function M-files, see "Function Name" in the MATLAB Programming Fundamentals documentation.

4 Press **Enter** or **Return**.

MATLAB creates the new file.

5 To enter or edit the contents of the new file, open it. For more information, see "Opening Files from the Current Directory Browser" on page 5-67.

Functions and Other Alternatives for Creating Files. To create a new M-file or other type of text file in the Editor, use the `edit` function.

To create various types of MATLAB files and related files, select **File > New** from the MATLAB desktop.

Creating and Updating MAT-Files in the Current Directory Browser

Starting from existing variables in the workspace, you can create a new MAT-file and you can update an existing MAT-file:

1 In the Current Directory browser, change the current directory to the one where you want to save the variables.

2 In the Workspace browser, select the variables that you want to save in a new or existing MAT-file.

3 Drag the selected variables from the Workspace browser to the Current Directory browser, releasing them as follows:

- To update an existing MAT-file with the selected variables, drop the variables onto the MAT-file name in the Current Directory browser. MATLAB warns you if the MAT-file already contains variables of the same name, and lets you choose to continue and replace the data in the existing variables, or cancel the update. If you do not want to see the warning, you can suppress it using a preference. Select **File > Preferences > General > Confirmation Dialogs**, and clear the **Confirm when overwriting variables in MAT-files** check box.

- To create a new MAT-file with the selected variables, drop the variables onto any empty area in the Current Directory browser. MATLAB creates a new, untitled MAT-file and prompts you to name it.

You can also load variables from a MAT-file in the current directory into the workspace. For details, see "Opening Files In MATLAB from the Current Directory Browser" on page 5-67.

Accessing Source Control Interface Features from the Current Directory Browser

If you have specified a source control system for MATLAB to interface with, you access it from the Current Directory browser. Select a file or files in the Current Directory browser and right-click. From the context menu, select **Source Control**, and then select a source control feature. For more information, see Chapter 10, "Source Control Interface".

Locating a File or Directory on Disk

To open the Windows Explorer tool (or the Finder tool on Apple Macintosh platforms) to the current directory location, in the Current Directory browser, right-click, and from the context menu, select **Locate on Disk**.

Adding Directories to the Search Path from the Current Directory Browser

From the Current Directory browser, you can add directories to the MATLAB search path. Add directories to the search path so MATLAB can access the files in those directories, regardless of the current directory. For related information, see "Search Path" on page 5-35.

To add directories to the search path, select the directories and right-click. From the context menu, select **Add to Path**. Then select one of these options:

- **Selected Folders** — Adds the selected directories to the search path.
- **Selected Folders and Subfolders** — Adds the selected directories and all of their subdirectories to the search path.

To add the current directory to the search path, right-click without making a selection. Then, from the context menu, select **Add to Path > Current Folder**.

Alternative Ways to Add Directories to the Search Path. You can use the addpath function and the Set Path dialog box to add directories to the search path. For more information, see "Adding Directories to the Search Path" on page 5-42.

Accessing the File and Directory Comparison Tools from the Current Directory Browser

To determine and display the differences between two files or two directories, use the MATLAB File and Directory Comparisons tool. For information on using the tool, its results, and other options, see "Comparing Files and Directories" on page 6-79.

Importing Data Using the Current Directory Browser

You can import data from a file in the Current Directory browser into the MATLAB workspace. Select the file, right-click, and select **Import Data** from the context menu. The Import Wizard opens. For more instructions, click the **Help** button in the Import Wizard.

Performing Standard File Operations using MATLAB

- "Creating New Folders Using the Current Directory Browser" on page 5-73
- "Copying Files and Directories Using the Current Directory Browser" on page 5-74
- "Cutting and Deleting Files and Directories Using the Current Directory Browser" on page 5-75
- "Moving Files and Directories Using the Current Directory Browser" on page 5-75
- "Renaming Files and Directories Using the Current Directory Browser" on page 5-76

Creating New Folders Using the Current Directory Browser

To create a new folder (directory) using the Current Directory browser:

1 Navigate to the directory where you want to create the new folder.

2 From the Actions button, select **New Folder**.

A folder icon, with the default name `NewFoldern` appears in the list of files shown in the Current Directory browser.

3 Type over `NewFoldern` with the name you want to give to the new directory.

4 Press the **Enter** or **Return** key.

The directory is added.

After creating a new folder, these are some common actions you can perform with it:

- Move files or directories into it—see "Moving Files and Directories Using the Current Directory Browser" on page 5-75.
- Add a directory to the search path—see "Adding Directories to the Search Path from the Current Directory Browser" on page 5-72

Function Alternatives for Creating New Folders. To create a directory, use the `mkdir` function. For example,

```
mkdir newdir
```

creates the directory `newdir` within the current directory.

Copying Files and Directories Using the Current Directory Browser

You can copy and paste files and directories using the Current Directory browser:

1 Select the files or directories to copy. To select multiple items, use **Shift**+click or **Ctrl**+click. For a directory, the entire contents are copied, including all subdirectories and files.

2 Right-click and select **Copy** from the context menu.

3 Navigate to the file or directory where you want to paste the items you just copied.

4 Right-click and select **Paste** from the context menu.

Copying Files and Directories Between Other Applications and the Current Directory. You can copy and paste files and directories to and from tools outside of MATLAB, such as the Windows Explorer tool. You can use any of these methods:

- Current Directory browser menu items
- Keyboard shortcuts
- Drag and drop for your platform, such as **Ctrl**+drag on Windows platforms

Functions for Copying Files and Directories. To copy and paste files or directories, use the `copyfile` function. For example, to make a copy of the file `myfun.m` in the current directory, assigning it the name `myfun2.m`, type

```
copyfile('myfun.m','myfun2.m')
```

Cutting and Deleting Files and Directories Using the Current Directory Browser

To cut or delete files and directories:

1 Select the files and directories to remove.

2 Right-click and select **Cut** or **Delete** from the context menu.

   The files and directories are removed.

Files and directories that you delete from the Current Directory browser go to the Recycle Bin (or Trash Can) if your operating system settings specify that action. A confirmation dialog box displays before the items are deleted, if you have set that option in your operating system. For example, to set the option on Windows platforms, right-click the Recycle Bin, select **Properties** from the context menu, and then, under the **Global** tab, select the check box to **Display delete confirmation dialog**.

If you do not want a selected items to go to the Recycle Bin, press **Shift+Delete**.

Function Alternatives for Cutting and Deleting Files. To delete a file, use the `delete` function. For example,

```
delete('d:/mymfiles/testfun.m')
```

deletes the file `testfun.m`. You can recover deleted files if you use the `recycle` function or the equivalent preference described in "Default Behavior of the Delete Function" on page 2-109.

To delete a directory and optionally its contents, use `rmdir`. For example,

```
rmdir('myfiles')
```

removes the directory `myfiles` from the current directory.

Moving Files and Directories Using the Current Directory Browser

To move files and directories to another directory within the current directory, drag the items to the new directory.

To move files and directories from the current directory to a directory outside of the current directory:

1 In the current directory, select the files and directories that you want to move.

2 Right-click, and from the context menu, select **Cut**.

The files and directories are moved from the current directory to the clipboard.

3 In the Current Directory browser, navigate to the directory where you want to move the items.

4 Right-click, and from the context menu, select **Paste**.

The items appear in the new location.

Moving Files and Directories Between the Current Directory Browser and Other Applications. You can move files and directories to and from tools outside of MATLAB, such as the Windows Explorer tool. You can use any of these methods:

- Current Directory browser menu items
- Keyboard shortcuts
- Drag and drop

Function Alternative for Moving Files. If you want to use a function to move files and directories, use `movefile`.

Renaming Files and Directories Using the Current Directory Browser

To rename a file or directory:

1 Select the item that you want to rename.

2 Right-click, and select **Rename** from the context menu.

3 Type over the existing name with the new name for the file or directory, and press **Enter** or **Return**.

The file or directory is renamed.

If you try to rename an M-file to an invalid name, such as `*myfile.m`, a warning appears. If you want to run the M-file, change the name to make it valid. For more information about valid names for M-files, see "Naming M-files".

Function Alternatives for Renaming Files and Directories. If you want to use a function to rename a file or directory in the Current Directory browser, use the `movefile` function. For example,

```
movefile('myfile.m','projectresults.m')
```

renames `myfile.m` to `projectresults.m`.

Finding Files and Directories

There are different ways to find files, depending on where you are looking and what you are looking for:

- "Finding Files and Directories in the Current Directory" on page 5-77
- "Filtering the Current Directory Browser View to Find Files and Directories" on page 5-79
- "Finding Files and Content Within Files in Any Directory" on page 5-82
- "Functions for Finding Files" on page 5-87
- "Other Ways to Find Files" on page 5-87

Finding Files and Directories in the Current Directory

To find a file or directory by name in the current directory, position the pointer in the contents list of the Current Directory browser and type the first letters of the file or directory name. As you type, the Current Directory browser searches downward from the top of the window and selects the first entry that matches what you typed. This feature is referred to as *typeahead* or *find as you type*.

For example, type m. The Current Directory browser searches to find the first entry beginning with m. In this example, it finds the directory mydemos.
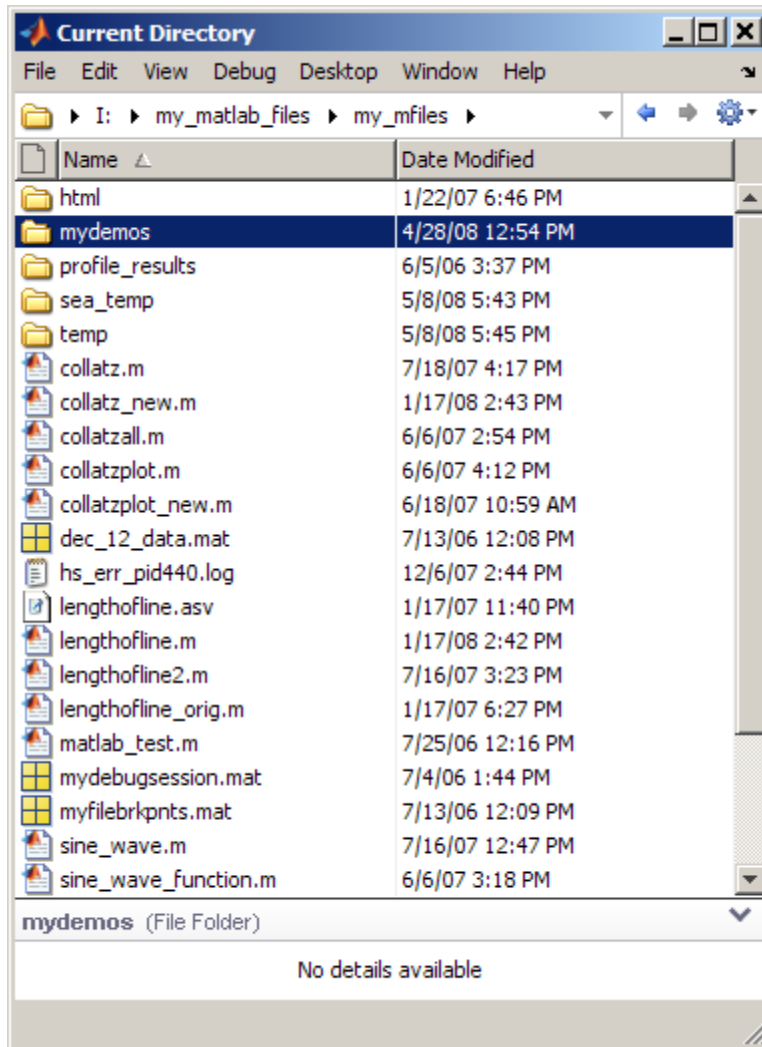
Type mat. The Current Directory browser searches to find the first entry beginning with mat. In this example, it finds the file matlab_test.m.
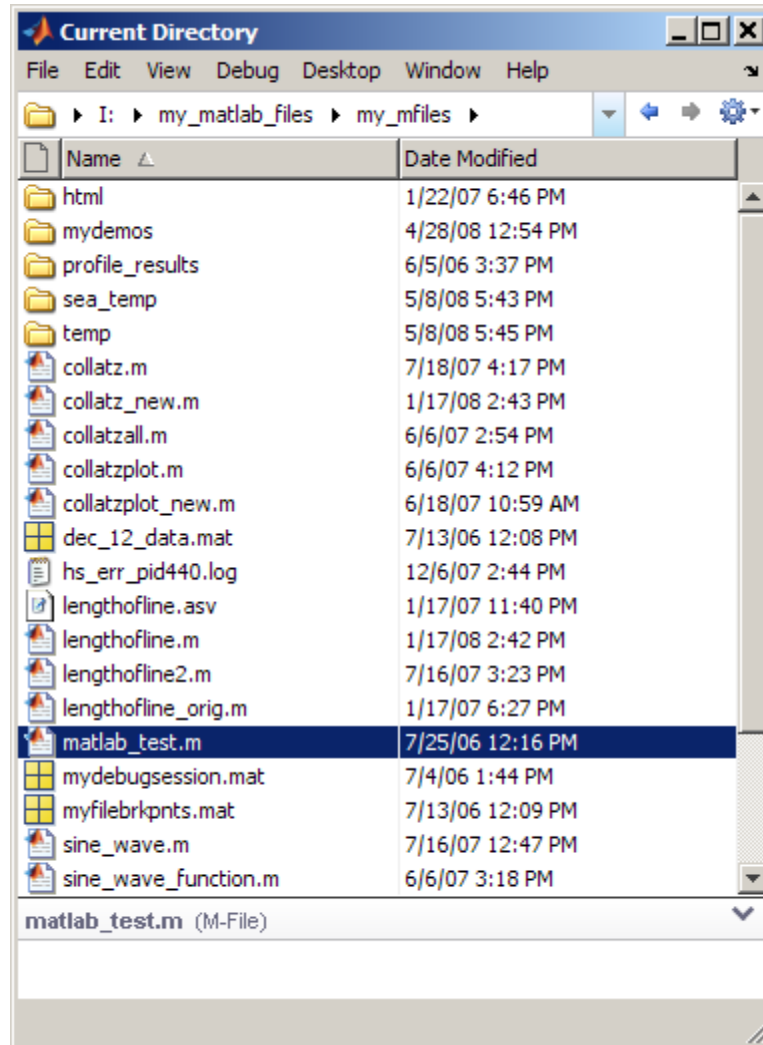
Filtering the Current Directory Browser View to Find Files and Directories

To show only items that contain a text string in their name, use the filter field in the Current Directory browser. This is also known as instant search.

You can use the wildcard character, `*`, in the string, and you can use it multiple times in the string. For example, you can show only files of a specified type, such as `.m` by entering `*.m`.

Follow these instructions to use the filter field:

1 If the filter field does not appear in the Current Directory browser, show it by selecting **File > Preferences > Current Directory**. Select the **Show filter field** check box and click **OK**.

   The filter field appears above the address bar in the Current Directory browser.

2 Type the letters in the filter field.

Type a string in the filter field to show
only items whose names contain that string.
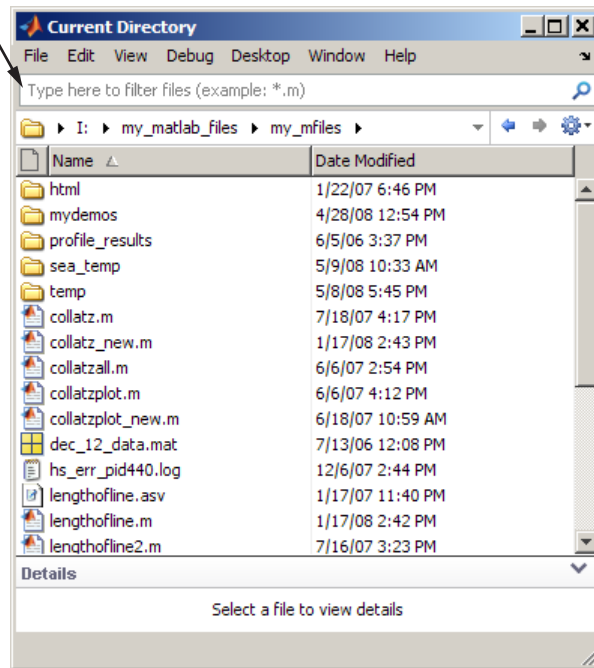
For example, type m, and the list shows only file and directory names that include m.
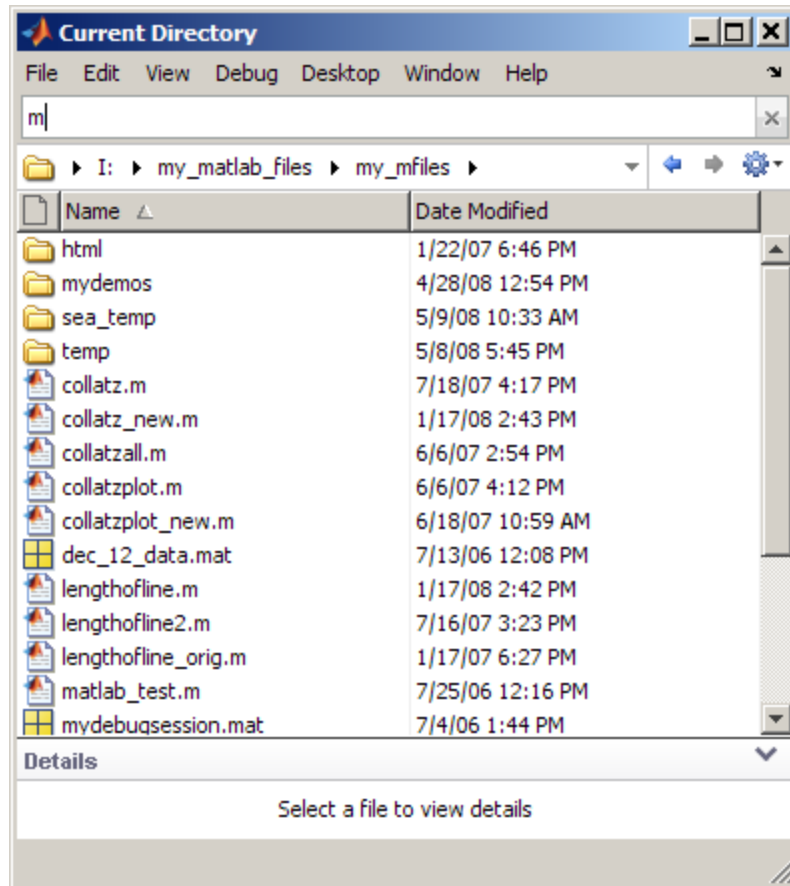
3 Further filter the list by typing additional letters. You can use *s (wildcard characters). Continuing the example, type a so the results show only items whose names contain ma.
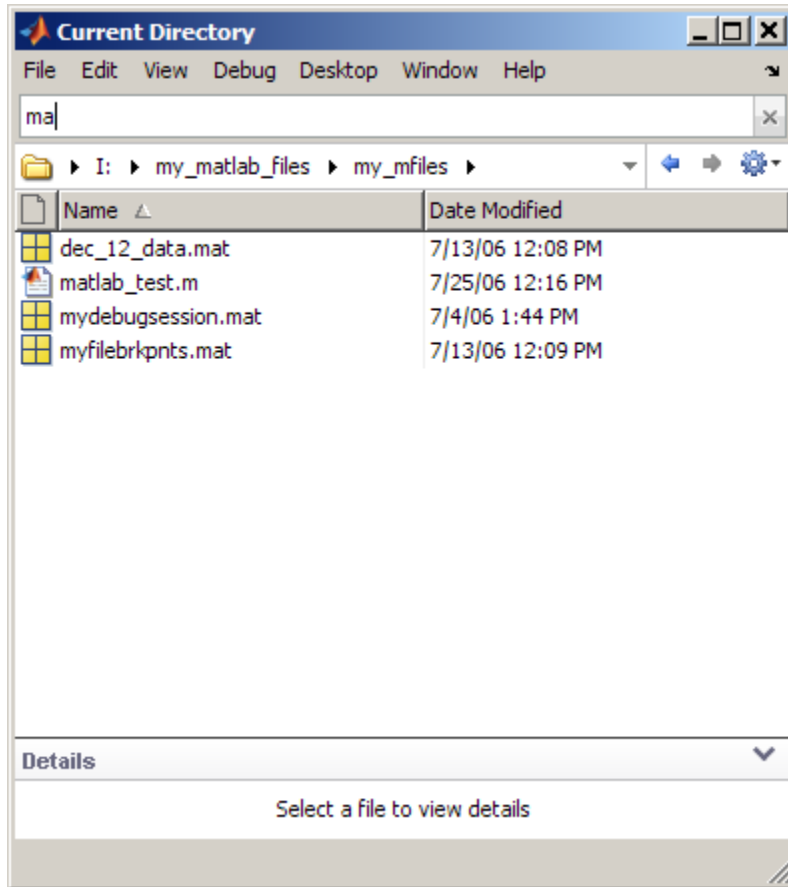
4 To clear the filter results and show all items in the current directory, click the Close box in the filter field, or press the **Esc** key.

Finding Files and Content Within Files in Any Directory

To search for file and directory names across multiple directories, or for specified text in multiple files, use the Find Files tool.

Results of find.
Click a column heading
to change sort order.

Close current
tab pane.

Type filename or
text (or both) you
want to find.

Restrict file types.

Select directories
to search in.

Start the find
operation.

Option to specify
exact match.
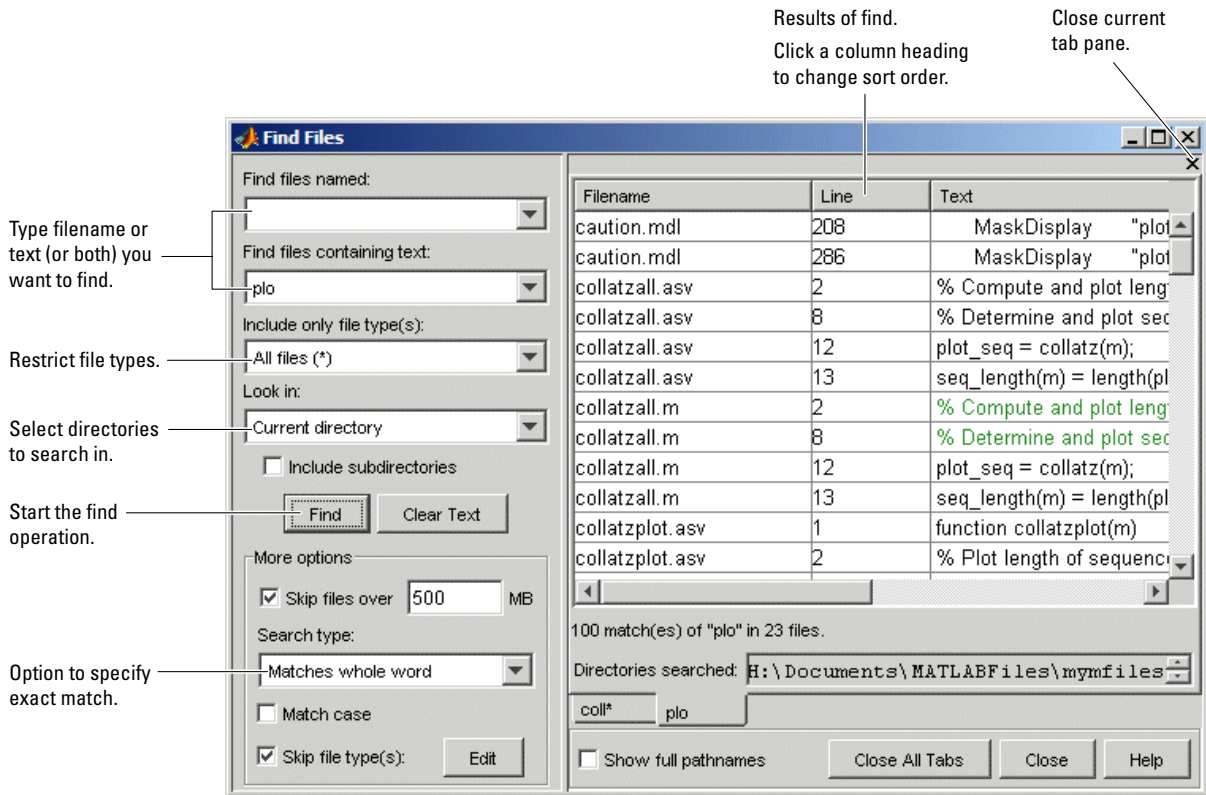
Follow these instructions to use the Find Files tool:

1 Open the Find Files tool by selecting **Edit > Find Files** from any desktop tool, such as the Current Directory browser or the Editor.

The Find Files dialog box opens.

2 Type the file name and/or text you are searching for:

- To search for files, type the file name in the **Find files named** field. You can use the wildcard character (*) in the file name. For example, type coll* to search for file names that start with coll.

- To search for text within files, type the text in the **Find files containing text** field. For example, search for `plot`. Alternatively, you can select text in the Command Window or Editor and that text appears in the field.

  Under **More options**, use the **Search type** to specify `Matches whole word`, or specify a partial match by selecting `Contains text`.

- To search for text strings in specified file names only, type entries in both fields. Use the **Clear Text** button to clear the entries in both fields.

Click the down arrow next to each field to select previous entries from the current session of MATLAB.

3 To restrict the types of files to search, select an option in **Include only file type(s)**. For example, select `*.m` to limit the search to M-files only.

To ignore files of other or multiple specified types, select **All files (*)** for **Include only file type(s)**. Then select the **Skip file types** check box (under **More options**). For details on how to specify the types to skip, see "Skip File Types in Find Files" on page 5-85.

4 Select the directories to search in from the **Look in** list box. Select the MATLAB current directory or MATLAB search path, or use the `Browse` option to select another directory. Alternatively, you can type the full path for one or more directories into this field, with each path separated by a semicolon (`;`). To include subdirectories in the search, select the **Include subdirectories** check box.

5 To further restrict the search, use additional entries under **More options**:

- When you are searching for text within files, you can ignore large files that might take a long time to search. To do so, select the check box for **Skip files over** the specified size.

- If uppercase and lowercase are relevant, select **Match case**.

6 To execute the search, click **Find**. While the search is in progress, the **Find** button label changes to **Stop Find**. To abort a search, click **Stop Find**.

Search results appear in the pane on the right side of the Find Files dialog box, with a summary of the results at the bottom of the pane. For text searches, the line number and line of code are shown. To see the full path names for the files, select the **Show full pathnames** check box.

7 To sort the results based on a column, click that column heading. Click the column heading again to reverse the sort order for that column. For example, click **Line** to sort results by line number.

Opening Files from the Find Files Results List. To open files shown in the results list, do one of the following.

Action	Result
Double-click a file.	Opens the file in the Editor.
Ctrl+click to select the files, and press **Enter** or **Return**.	Opens multiple files in the Editor.
Right-click selected files and choose **Open** from the context menu.	Opens one or more files in the tool associated with the file type (not the Editor).
Right-click the selected files and choose **Open as Text** from the context menu.	Opens any file type in the Editor.
Right-click the selected files and choose **Open Outside MATLAB** from the context menu.	Open files of any type outside of MATLAB.

For text searches, the file opens at the line number shown in the results section of the Find in Files dialog box. After you have opened a file in the Editor, you can change specified text using the Find & Replace tool.

Previous Results of Find Files. To see the results of a previous search, select its tab at the bottom of the results pane. Find Files shows up to 10 search result tabs while the tool is open, but does not maintain the results after you close the tool.

MATLAB maintains the state for options in the Find Files tool even after you end the session.

Skip File Types in Find Files. In the Find Files tool, you can restrict the search to look in all file types except those you specify. Follow these steps:

1 For **Include only file type(s)**, you must select **All files (*)**.

2 To specify the file types you want the search to ignore, select the **Skip file type(s)** check box and click **Edit**.

The Edit Skipped File Extensions dialog box opens.

To skip a file type not shown in this list, enter the extension and click **Add**. The extension then appears in the list.

Be sure its **State** is selected.

When **Skip file type(s)** in the **Find Files** tool is selected, the search ignores file types in this list whose **State** is selected.

For the example shown, the search would ignore P-files because **State** is selected, but would look in TAR files because **State** is cleared.

If you do not want a file type to appear in this list, select the name of the extension and click **Remove**.

Edit Skipped File Extensions

To exclude a new file type from searches (e.g. .foo), type the extension and click Add.

Add

State	Extension
	.mat
☑	.mex
☑	.p
☑	.pdf
☑	.ppt
☑	.pub
☑	.scc
☑	.sl
☑	.so
☐	.tar
☐	.zip

Remove   Restore Defaults

Select an extension in the list to exclude files of that type from searches. Clear it to include files of that type in searches.

OK   Help
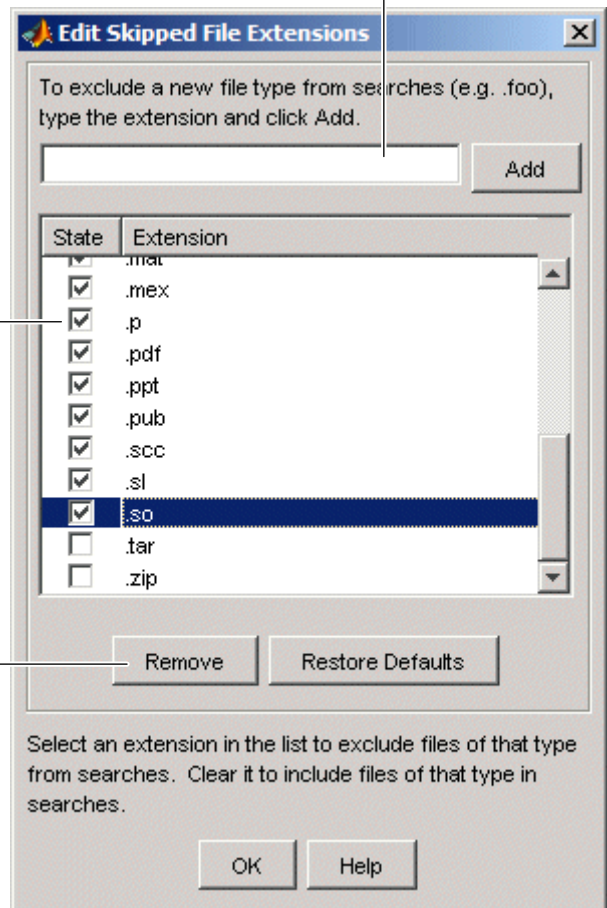
3 Select and clear file types in the Edit Skipped File Extension dialog box. Find Files will not look in any file type in the list whose **State** check box is selected. It will look in any file type in the list whose **State** check box is cleared.

If you want Find Files to skip a file type not shown in the list, enter the file extension in the field at the top of the dialog box and click **Add**. The type appears in the list. Be sure its **State** check box is selected. For the example shown, the scc file type was added.

You can reduce the size of the list by removing any file extensions. Select the name of the extension and click **Remove**.

4 Click **OK** to accept the changes and close the **Edit Skipped File Extensions** dialog box.

When you perform a find in the Find Files tool, the search ignores the selected file types.

Functions for Finding Files

Function	Description
exist	Determine if a variable, function, or directory exists.
lookfor	Search for the specified text in the first line of help for all M-files on the search path.
which	Displays the full path name to a file

Other Ways to Find Files

Here are topics about other ways to find files by name and to find content within files:

- "Viewing the Contents of the Current Directory" on page 5-60

- "Finding Functions Using the Function Browser" on page 3-41.

- "Searching Documentation and Demos with the Help Browser" on page 4-18.

• "Accessing User-Contributed Files — File Exchange" on page 4-93.

Reports for Files in the Current Directory

MATLAB provides reports that help you refine the M-files in a directory and improve their performance. Reports are also useful when you prepare files for use by others, such as for a finished project you are presenting, to share on MATLAB Central, or for a toolbox you will distribute to other users.

Access the reports using the Actions button in the Current Directory browser. Select **Reports**, and then select the type of report you want to run for the files in the current directory. For more information, see Chapter 7, "Tuning and Managing M-Files".

Preferences for the Current Directory Browser

Using preferences, you can specify:

• The number of recently used current directories to maintain in the history list

• How often the contents listing refreshes when changes are made from the operating environment's file system

• If the filter field appears in the address bar

To access preferences, select **File > Preferences > Current Directory**. The **Current Directory Preferences** pane appears in the Preferences dialog box.
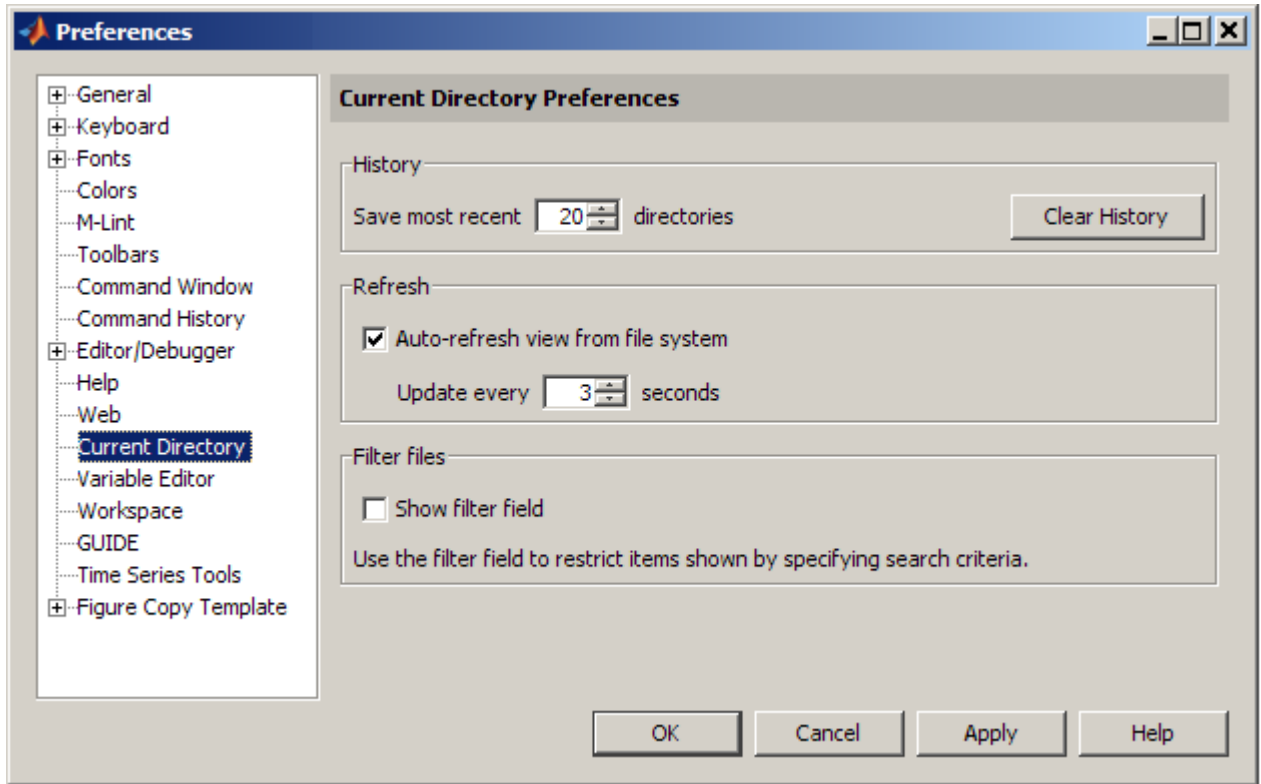
History

The drop-down list in the Current Directory field and in the Current Directory browser shows the history of current directories, that is, the most recently used current directories.MATLAB maintains the list of current directories for use in future sessions.

- Saving directories — Use the **Save most recent directories** field to specify how many directories are on the list at the start of the next MATLAB session.

- Removing directories — To remove all current directory entries in the list, click **Clear History**. The list is cleared immediately.

For more information, see "Using the Current Directory Field to View and Change the Current Directory" on page 5-58 and "Using the Current Directory Browser to View and Change the Current Directory" on page 5-58.

Refresh

If you experience general slowness in MATLAB, access files on a network, and you have the Current Directory browser open, the slowness could be because the Current Directory is automatically refreshing its view from the operating environment file system.

By default, the **Auto-refresh view from file system** check box is selected, with an update time of 3 seconds. This means that every 3 seconds, the Current Directory browser checks for and reflects any changes you made to files and directories in the current directory using applications other than MATLAB. This manner of refreshing works on the Apple Macintosh platform and platforms running some Linux[18] operating systems. On Windows platforms and platforms running some Linux operating systems, the update time is ignored and the Current Directory browser refreshes regularly, if the preference is selected.

To alleviate slowness, try increasing the update time. For extremely slow performance situations, clear the check box to turn auto-refresh off. For more information, see "Refreshing the View in the Current Directory Browser" on page 5-65.

Filter Files

To find only items that contain a specified text string in their name, use the filter field in the Current Directory browser. By default, the filter field is not shown. To show the filter field, select the **Show filter field** check box in Current Directory preferences. For more information about using the filter field, see "Filtering the Current Directory Browser View to Find Files and Directories" on page 5-79.

See Also

- "Setting Toolbars Preferences for Desktop Tools" on page 2-138

---

18. Linux is a registered trademark of Linus Torvalds.

- "Setting Fonts Preferences for Desktop Tools" on page 2-121
- "Managing Files and Working with the Current Directory" on page 5-55
- Using the MATLAB Chapter 2, "Desktop"

6

Editing and Debugging M-Files

MATLAB software provides powerful tools for creating, editing, and debugging files, as detailed here. For information about the MATLAB language and writing M-files, see the Programming Fundamentals documentation.

Begin with Existing Code

In this section...

Create M-Files from Command Window and History

Before you begin writing MATLAB code in a blank file, consider starting with existing resources for the code, and then use the MATLAB Editor to modify the code.

You create and run MATLAB statements in the Command Window, modify those statements to your satisfaction, and then create an M-file that includes the statements. To facilitate this process, in the Command History, select the MATLAB statements you want to include in the M-file. Right-click and select **Create M-File**. The Editor opens a new file that includes the statements you selected from the Command History. You can also copy the statements from the Command History and paste them into an existing M-file.

Use Existing M-Files and Examples

If you can find existing M-files that accomplish what you want to do, copy, and use the code in your own M-file, assuming you have legal permission to do so. Following are some resources you can use.

MATLAB and Toolbox Functions

You can access and reuse the code in most MATLAB and toolbox functions that have a `.m` file extension. You cannot use MATLAB and toolbox functions that are built-in. They are efficient but their code is not accessible.

If there is a MATLAB function that is similar to what you need to do and it is not built-in, open the file in the Editor and use it as a basis for your file. Be sure to save the file using a different name and in a directory that is not in `matlabroot/toolbox`. See "Saving M-Files" on page 6-98 for details.

Demos and Examples

The MATLAB product and its toolboxes include demonstration programs. You can view the code in the demos and copy it for use in your own M-files. To see the demos, type demo, which opens the Help browser to the **Demos** pane. For more information about demos, see "Viewing and Running Demos" on page 4-41.

There are also code examples in the online documentation. To see a list of examples for a product, type helpbrowser to open the Help browser. In the **Contents** pane, click **+** for a product to view the help topics, and then select the 💡 **Examples** entry.

File Exchange

The MathWorks Web site features a user-contributed code library, from which you can download free M-files contributed by users and developers of MATLAB software, Simulink software, and related products. To view the files available to download, go to the MATLAB Central File Exchange page on the MathWorks Web site, http://www.mathworks.com/matlabcentral/fileexchange, or access it using the **Help > Web** menu in any desktop component.

Ways to Edit, Evaluate, and Debug M-Files

There are several methods for creating, editing, evaluating, and debugging files with MATLAB software.

Creating and Editing Files — Options	Instructions
MATLAB Editor	See "Starting, Creating Files, and Closing the Editor" on page 6-7, and "Creating New Files in the Editor" on page 6-8. You can create, open, edit, and save M-files as well as other file types in the MATLAB Editor—see "Creating and Editing Other Text File Types" on page 6-12.
Any text editor, such as Emacs or vi	To specify another editor as the default for use with MATLAB software, select **File > Preferences > Editor/Debugger**, and for **Editor**, specify the **Text editor**. Click the **Help** button in the Preferences dialog box for details. Use that editor by default, or use any other editor you open. Regardless of the editor you use, you can debug M-files using the MATLAB Editor or debugging functions.

Debugging M-Files – Options	Instructions
General debugging tips	See "Finding Errors, Debugging, and Correcting M-Files" on page 6-120.
MATLAB Editor	See • "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 6-123 to identify errors and make improvements. • "Debugging Process and Features" on page 6-154 to help you isolate run-time problems.
MATLAB debugging functions (for use in the Command Window)	See function alternatives in "Debugging Process and Features" on page 6-154.

See Customizing the Editor by Setting Preferences for information on setting up the editing and debugging environment to best meet your needs.

For information about the MATLAB language and writing M-files, see the Programming Fundamentals documentation.

Starting, Creating Files, and Closing the Editor

In this section...
"Starting the Editor" on page 6-7
"Creating New Files in the Editor" on page 6-8
"Opening Existing Files in the Editor" on page 6-10
"Arranging Editor Documents" on page 6-12
"Creating and Editing Other Text File Types" on page 6-12
"Closing the Editor" on page 6-12

Starting the Editor

The MATLAB Editor provides a graphical user interface for basic text editing features for any file type, as well as for M-file debugging. The Editor is a single tool that you can use for editing, debugging, or both. There are various ways to start the Editor. The Editor automatically starts when you open a document or create a new one. Once started, you can customize the Editor to suit your needs.

This figure shows an example of the Editor outside of the desktop opened to an existing M-file, and calls out some of the tool's useful features.

The * after the file name indicates that the file has not been saved since it was last modified.

Set the right-hand text limit using **Preferences > Editor/Debugger > Display**.

Toolbar for cell features.

Set breakpoints in the alley.

Color highlight syntax elements.

Status bar information includes cursor line and column number.
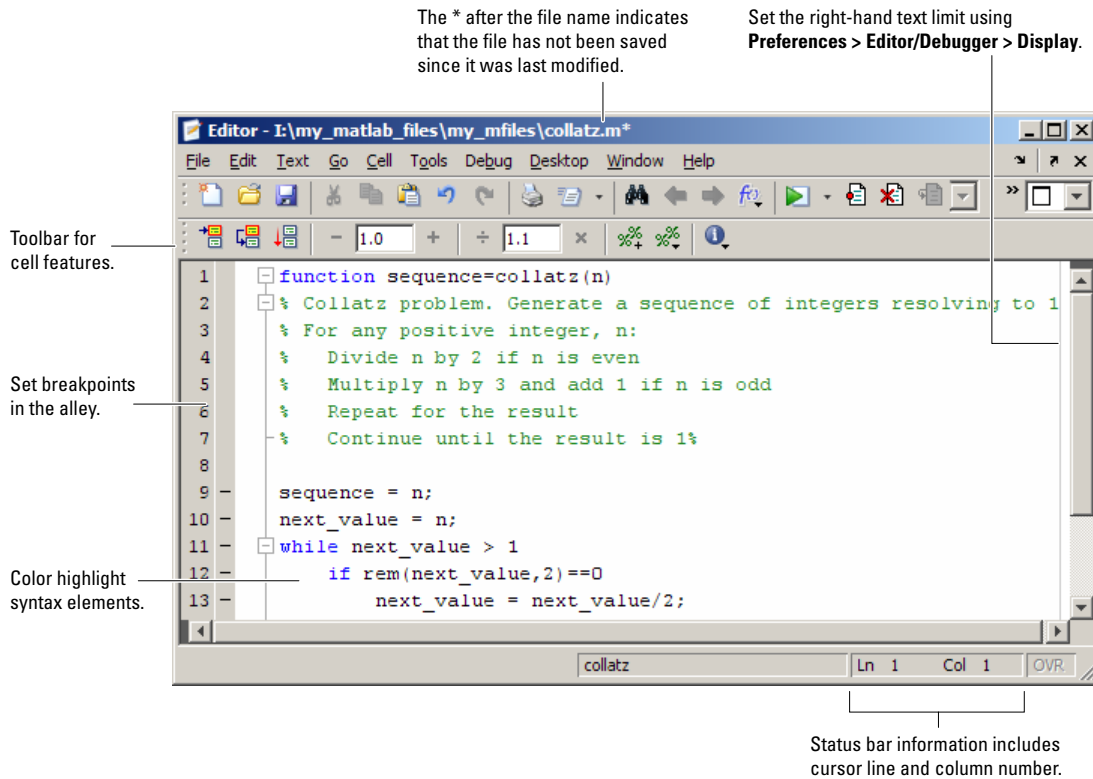
Creating New Files in the Editor

You can create a new blank or prepopulated file in the Editor. A prepopulated file provides elements typically included in the file type you choose to open.

With the Editor as the current (active) window, choose **File > New**, and then choose one of the following:

- **M-File** to create an empty M-file, such as for a script. You can also use this option to create a text file, such as an XML or HTML file.

  Alternatively, to create an empty M-file in the Editor, click the New M-File button on the MATLAB desktop toolbar.

- **Function M-File** to create an M-file prepopulated with basic M-file function elements.

- **Class M-File** to create a class definition M-file (`classdef`), prepopulated with basic M-file class structure elements.

The Editor opens an untitled file in the MATLAB current directory.

The location of the new file and the Editor are determined by document positioning guidelines. You can rearrange the document location to suit your needs. For details, see "Opening and Arranging Desktop Documents" on page 2-21.

Other tools also provide features for creating new M-files. For example:

- In the Command History window, you can select statements, right-click, and then select **Create M-File** from the context menu.
- In the Current Directory browser, you can create a file from the context menu, as described in "Creating New Files Using the Current Directory Browser" on page 5-69.

Function Alternative for Creating New Files

Type `edit` in the Command Window to create a file in the Editor.

Type `edit filename.ext` to create the file `filename.ext`. If `filename.ext` exists in the current directory or on the MATLAB search path, this opens the existing file. If `filename.ext` does not exist in the current directory or on the MATLAB search path, a confirmation dialog box might appear asking if you want to create a new file titled `filename.ext`:

- If you click **Yes**, the Editor creates a blank file titled `filename.ext`. If you do not want the dialog to appear in this situation, select that check box in the dialog. Then, the next time you type `edit filename.ext`, the file is created without first prompting you.
- If you click **No**, the Editor does not create a new file. If you do not want the dialog to appear in this situation, select that check box in the dialog. In that case, the next time you type `edit filename.ext`, a "file not found" message appears.

For more information about the confirmation dialog box, see "Confirmation Dialogs Preferences" on page 2-112.

Opening Existing Files in the Editor

To open an existing file in the Editor, click the Open file button 📂 on the desktop or Editor toolbar, or select **File > Open**.

The **Open** dialog box appears, listing all M-files. You can see different files by changing the selection for **Files of type** in the dialog box. Type or select a file name, and click **Open**. If you access the **Open** dialog box from the desktop, the current directory files are shown, but if you access it from the Editor, the files in the directory for the current file are shown. For special considerations on the Macintosh platform, see "Using File Browser GUIs on Macintosh Platforms to Navigate Within the MATLAB Root Directory" on page 2-88.

The Editor opens, if it is not already open, with the file displayed. You can have multiple Editor files open at once, and the location of the files and the Editor are determined by document positioning guidelines. You can rearrange the documents to suit your needs. For details, see "Opening and Arranging Desktop Documents" on page 2-21.

To make a document in the Editor become the current document, click it, or select it from the **Window** menu or document bar.

M-File Cells

If you open an M-file that contains M-file cells, yellow highlighting and gray horizontal lines might appear in the M-file, along with an information toolbar. Cell mode is used for publishing results and rapid code iteration. An M-file cell is denoted by a %% at the start of a line. MATLAB software interprets any M-file that contains %% at the start of a line as including cells and the Editor reflects the cell toolbar state and the cell display preferences, such as yellow highlighting of the current cell and gray lines between cells.

The first time you open an M-file that contains cells, an information bar appears below the cell toolbar, providing links for details about cell mode. To dismiss the information bar, click the close box on the right side of the bar. The information bar does not appear again, but you can get the same

quick access to the information about M-file cells from the Show Cell Mode information button on the cell toolbar.

To hide the cell toolbar, right-click in the toolbar and select **Cell Toolbar** from the context menu. If you do not want cell mode enabled, select **Cell > Disable Cell Mode**. If cell mode is disabled when you quit a MATLAB session, it is disabled the next time you start a MATLAB session; the converse is true as well.

Other Methods for Opening Files in the Editor

These are other ways to open files in the Editor:

- Drag a file from another MATLAB desktop tool or a Microsoft tool into the Editor. For example, drag files from the Current Directory browser, or from Windows Explorer.

- Open files from the Current Directory browser—see Opening Files.

- Select a file to open from the most recently used files, which are listed at the bottom of the **File** menu in the Editor and all other desktop tools. You can change the number of files appearing on the list—select **File > Preferences > Editor/Debugger** and in the **Most recently used file list**, specify the **Number of entries**.

- In the Editor or another desktop tool such as the Command Window, select a file name, right-click, and select **Open Selection** from the context menu to open that file. For details, see "Opening a Selection in an M-File" on page 6-71.

- Set a preference that specifies that a MATLAB session, upon startup, is to automatically open the files that were open when the previous MATLAB session ended. Select **File > Preferences > Editor/Debugger** and in the **Opening files in editor area**, select the check box for **On restart reopen files from previous MATLAB session**.

Function Alternative for Opening an M-File. Use the `edit` or `open` function to open an existing file in the Editor. For example, type

```
edit collatz.m
```

6-11

to open the file `collatz.m` in the Editor, where `collatz.m` is on the search path or in the current directory. Use the relative or absolute path for the file you want to open if it is not on the search path or in the current directory.

Arranging Editor Documents

You can arrange the size and location of M-files and other text documents you open in the Editor. Editor documents follow the same arrangement practices as other desktop documents. For details, see "Opening and Arranging Desktop Documents" on page 2-21.

Creating and Editing Other Text File Types

You can edit any type of text file using the MATLAB Editor. For example, you can open and edit an HTML file. Note that you can run or debug only M-files from the Editor.

When working with C/C++, Java, and TLC programming languages, as well as XML or HTML, you can specify syntax highlighting and indenting preferences appropriate to those languages. Select **File > Preferences > Editor/Debugger > Language**. For details, click the **Help** button in the dialog box.

Closing the Editor

To close the Editor, click the **Close** box in the title bar of the Editor. This is different from the **Close** box in the menu bar of the Editor, which closes the current file when multiple files are open in a single window.

Close box for current file.

Close box for Editor/Debugger.

Editor - I:\my_matlab_files\my_mfiles\collatz.m
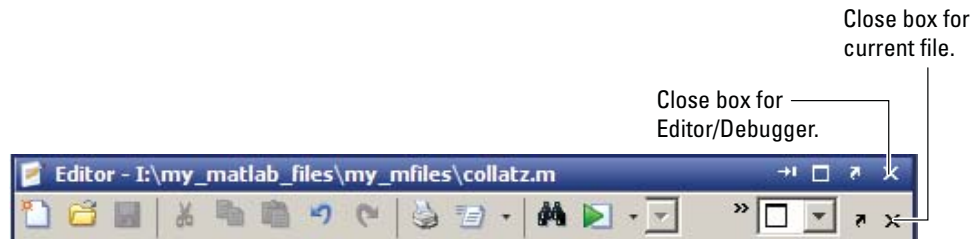
If multiple files are open, with each in a separate window, close each window separately. To close all files at once, select **Close All Documents** from the

Window menu. Note that this will close other desktop documents as well, such as variables in the Variable Editor, and it will close the tools as well, that is, the Editor and Variable Editor, for example.

When you close the Editor and any of the open files have unsaved changes, you are prompted to save the files.

Customizing the Editor by Setting Preferences

Overview of Setting Preferences for the Editor/Debugger

Using preferences, you can specify the default behavior for various aspects of the Editor.

To set preferences for the Editor, select **File > Preferences > Editor/Debugger**. The Preferences dialog box opens showing **Editor/Debugger Preferences**.
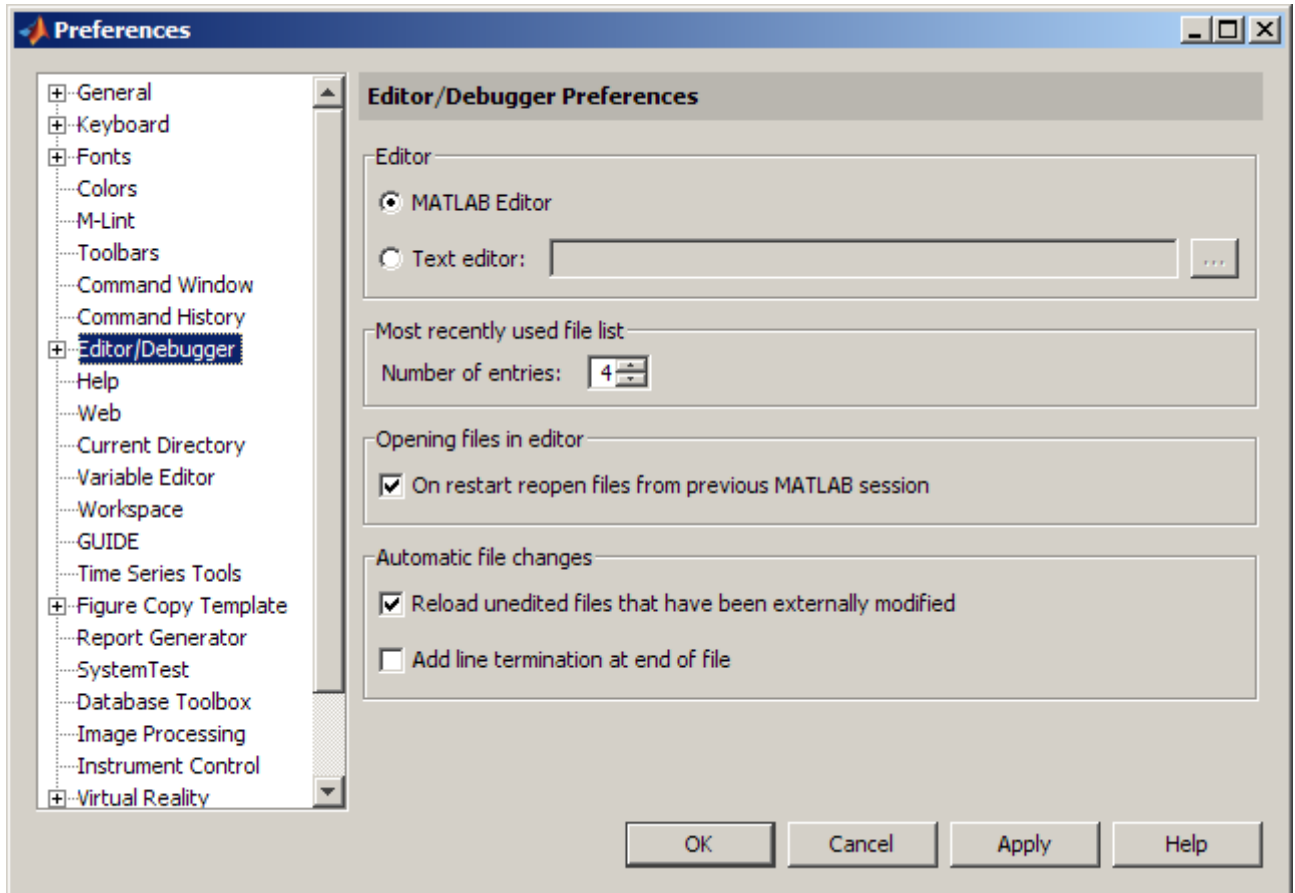
Click the **+** next to Editor/Debugger in the left pane to view all categories of Editor/Debugger preferences. Select a category and that preference pane displays. Make changes and click **Apply** or **OK**.

Click the **Help** button in the Preferences dialog box for details about Editor/Debugger preferences.

You can also set preferences for the Editor toolbars. Select **File > Preferences > Toolbars**, and from the **Toolbar** drop-down list select **Editor** or **Editor Cell Mode**, depending on the toolbar for which you want to set preferences.

Setting General Preferences for the Editor/Debugger

Select **File > Preferences > Editor/Debugger** to specify the preferences on the main pane, listed here. Or click the **+** to see additional Editor/Debugger preferences.

- "Editor" on page 6-16
- "Most Recently Used File List" on page 6-16
- "Opening Files in Editor" on page 6-17
- "Automatic File Changes" on page 6-17

Editor

MATLAB Editor. Selecting **MATLAB Editor** means that the MATLAB desktop uses the built-in Editor.

Text editor. To specify a text editor other than the MATLAB Editor, such as Emacs or vi, to be used when you open an M-file from within MATLAB, select **Text editor**. Specify the full path for the editor application you want to use.

For example, specify `c:/Applications/Emacs.exe` in the **Text editor** field, and then open a file using **Open** from the **File** menu in the MATLAB desktop. The file opens in Emacs instead of in the MATLAB Editor.

Note that even if you use another editor, Editor/Debugger preferences are still available because some of the Editor/Debugger preferences apply to other MATLAB tools.

Most Recently Used File List

Use this preference to specify the number of files that appear in the list of most recently used files at the bottom of the **File** menu. You select a file from the list to open it.

Opening Files in Editor

On restart. To start a MATLAB session and automatically open the files that were open when you last shut down MATLAB, select the item **On restart open files from previous MATLAB session**. If the item is not selected and you close MATLAB when there are files open in the Editor, the next time you start MATLAB, the Editor is not opened upon startup.

Automatic File Changes

Reload unedited files that have been externally modified. This option is useful when you edit files in the MATLAB Editor and outside of MATLAB.

With the option selected, the Editor automatically reloads another version of the file as described here. When a file is open in the Editor and the same file is also open outside of MATLAB, the Editor automatically uses the version open outside of MATLAB if you have not edited the file in the Editor.

When you clear this preference, MATLAB displays a dialog box notifying you that the current file was changed outside of MATLAB, and asks if you want to use the latest version.

Add line termination at end of line. With this preference selected, if the last line in the file is not empty, the Editor automatically adds a new empty line to the end of the file. Select this preference if you use your files with other products that expect a new line (sometimes referred to as a `<CR>`) at the end of the file.

Setting Display Preferences

Select **File > Preferences > Editor/Debugger > Display** to specify these preferences:

- "General Display Options" on page 6-18
- "Right-Hand Text Limit" on page 6-18
- "Cell Display Options" on page 6-20

See also Desktop Preferences.

General Display Options

Highlight Current Line. Select this preference to highlight the current line, that is, the row with the caret (also called the cursor). This is useful, for example, to help you see where copied text will be inserted when you paste. Then specify the color used to highlight the line.

Show line numbers. Select this preference to show line numbers along the left edge of the Editor window. Showing line numbers allows you to easily use various Editor features, such as **Edit > Go To Line**. When you clear this item, line numbers are not shown.

Enable datatips in edit mode. Select this preference to see datatips while editing an M-file. By default, datatips do not display while editing (edit mode), although they always display while debugging (debug mode). In edit mode, the datatips display the values of variables in the base workspace, so this is useful for script M-files rather than function M-files. In edit mode for a function M-file, the datatip for a variable displays a value if that variable also exists in the base workspace—the datatip displays the value of the base workspace variable, not the value of the variable in the function M-file.

While you are debugging, you cannot turn off the display of datatips, and they show the value of the variables in the workspace selected in the **Stack**. For more information about datatips, see "Viewing Values as Data Tips in the Editor" on page 6-166.

Right-Hand Text Limit

By default, a gray vertical line that is 1-pixel wide appears at column 75 in the Editor to indicate when a line of code becomes wider than desired. You might want to set a right-hand text limit for reasons such as the following:

- To prevent the need to scroll from left to right to see an entire line of text in the Editor

- To keep each line below a limit imposed by another text editor in which you intend to view the code

- To keep each line below a character limit required to ensure that the file will print without cropping text

You might need to print a test page to determine the most appropriate value for the printer printing the file, the margin settings for the printer, and the font size you are using.

Note This limit is a visual cue only and does not prevent text from exceeding the limit. For information on setting a value to automatically wrap comment text at a specified column number, see "Formatting Comments in M-Files" on page 6-42.
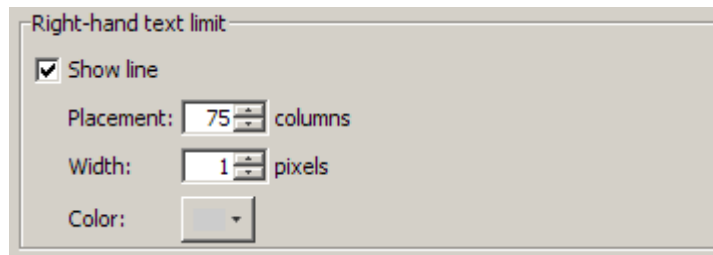
Show line. Select this preference to display the vertical line in the Editor. Clear it to hide the vertical line.

Placement. Type or select a new value for this field to change the column where the vertical line appears,

The actual column number where you set this preference is most useful when the font preference for the Editor is a monospaced font. In a monospace font, all characters are the same width. See "Setting Fonts Preferences for Desktop Tools" on page 2-121 for details.

Width. Type or select a value for this field to change the width (in pixels) of the vertical line.

Color. Click the down arrow next to the color block to open a palette of colors from which you can choose a new color for the vertical line. Increasing this value might improve the visibility of the vertical line on liquid crystal displays (LCDs) and projectors.

Cell Display Options

Use cells for testing and publishing code, as described in "Using Cells for Rapid Code Iteration and Publishing Results" on page 6-185.

Highlight cells. Select this item to highlight the current cell and select a color for the highlighting. Pale yellow is the default color.

Show lines between cells. Select this preference to display a faint gray horizontal line above each new cell. The line does not appear in the published M-file, nor in the printed M-file.

Setting Tab and Indent Preferences

Select **File > Preferences > Editor/Debugger > Tab** to specify the preferences that follow.

Tabs and Indents

Tab size. Specify the amount of space inserted when you press the **Tab** key. When you change the **Tab size**, it changes the tab size for existing lines in that file, unless the **Tab key inserts spaces** also is selected.

Tab key inserts spaces. Select this item if you want a series of spaces to be inserted when you press the **Tab** key. If the item is not selected, a tab acts as one space whose length is determined by **Tab size**.

Indent size. Specify the indent size for smart indenting.

Emacs-style Tab key smart indenting. This indenting convention is based on the style used by the Emacs editor and is similar to the **Enable smart indenting** preference. With this preference selected, lines are indented according to smart indenting preferences when you position the cursor in a line or select a group of lines, and then press the **Tab** key. Then, you cannot use tabs within a line.

See Also

Information about additional preferences for indenting is provided in these sections:

- "Setting Language Preferences" on page 6-21 for additional indenting preferences

- "Setting Keyboard Preferences for Desktop Tools" on page 2-116 for key binding, tab completion, and delimiter matching preferences

- Desktop Preferences

Setting Language Preferences

Select **File > Preferences > Editor/Debugger > Language** to specify preferences, such as syntax highlighting, for the type of file you are editing. Language preferences apply based on the current file name extension. To specify language preferences, first select the **Language**. The preference dialog box displays the preferences for that language. Make changes to preferences for that language and click **Apply**.

For example, when you edit a file with an .html extension, the language preferences for XML/HTML apply to that file. If you have a Java file open at the same time, the Java language preferences apply to the Java file.

File Extensions

To use the language preferences, the file name you edit must have one of the extensions shown in the **File extensions** list. Use the **Add** and **Remove** buttons to change the extensions associated with the selected language. Below are the default file extensions for each language. If a file has no extension (for example, a new untitled file), an M-file is assumed. For an extension other than those in the list, there is no syntax highlighting. Changes you make to the file extensions do not apply to currently open files, so close files and open them again.

Language Preference	Default File Extensions
MATLAB (MATLAB M-file)	.m (uppercase or lowercase)
TLC	.tlc
C/C++	.c, .cpp, .h, .hpp
Java	.java
XML/HTML	.xml, .xsl, .wsdl, .html, .htm, .shtml

For details about preferences for each language, see

- "Setting MATLAB Language Preferences" on page 6-22
- "Setting TLC Language Preferences" on page 6-27
- "Setting C/C++ Language Preferences" on page 6-28
- "Setting Java Language Preferences" on page 6-30
- "Setting XML/HTML Language Preferences" on page 6-31

See Also
See also:"Setting Keyboard Preferences for Desktop Tools" on page 2-116 for the Editor/Debugger and Command Window

Setting MATLAB Language Preferences
Select **File > Preferences > Editor/Debugger**, and then from the **Languages**, choose **MATLAB** to specify these preferences for editing M-files:

- "Syntax" on page 6-22
- "Indenting" on page 6-23
- "Formatting Comments" on page 6-27
- "File extensions" on page 6-27

See also "Setting Preferences for M-Lint" on page 6-145.

Syntax

Enable syntax highlighting. Select the **Enable syntax highlighting** check box to show colors that help you identify certain constructs like matching if/else statements. Click **Set syntax colors**, and the **Colors** preference pane opens, where you specify the colors used for syntax highlighting.

By default, syntax highlighting for M-files applies to files with the m extension. The extension can be uppercase or lowercase. Because syntax highlighting is used in the Command Window, Command History, and other tools, as well as

in the Editor, the syntax highlighting colors preferences apply to all tools that use syntax highlighting. For a description of syntax highlighting, see "Setting Colors Preferences for Desktop Tools" on page 2-131.

Indenting

You can set preferences to specify if you want MATLAB to automatically apply indenting to your M-file code. The indenting options you specify, however, apply only to lines you enter after changing the preference; they do not affect the indenting for existing lines. For information on changing the indenting for existing lines, see "Manual Indenting" on page 6-51.

Enable Smart Indenting. Select this option to specify that you want:

- The body of loops to be automatically indented within the start and end of the loop statement.
- The lines you indent using tabs or spaces to result in subsequent lines automatically aligning with the indented line.
- Functions to be automatically indented as specified with the **Function indenting format** option.

After entering a new line, use **Shift+Tab** to move text back to a previous indent level if you want.

Although you can manually insert tabs at the start of a line, be aware that smart indenting may not work properly. Use the **Text** menu entry for **Smart Indent** to correct selected lines.

Deselect **Enable smart indenting** to specify that you want:

- Lines to be aligned on the left by default
- To specify indenting manually using the tab and space keys
- You do not want functions automatically indented as specified by the **Function indenting format** option

When you deselect **Enable smart indenting** you can still indent functions, as specified by the **Function indenting format** option by selecting the functions in the Editor, and then pressing Ctrl-I.

To set the indent size for smart indenting and the tab size for manual indenting, see "Setting Tab and Indent Preferences" on page 6-20.

Example of Smart Indenting.

```
sequence = n
next_value = n;
while next_value > 1
    if rem(next_value,2)==0
        next_value = next_value/2;
    else
        next_value = 3*next_value+1;
    end
    sequence = [sequence, next_value]
end
```

Created with Smart indenting selected. Did not insert tabs manually. Indenting was inserted automatically as text was entered.

Example of Smart Indenting Deselected with Manual Tabs.

```
sequence = n
next_value = n;
while next_value > 1
    if rem(next_value,2)==0
    next_value = next_value/2;
    else
    next_value = 3*next_value+1;
    end
    sequence = [sequence, next_value]
end
```

Created with Smart indenting deselected. Indentation created by manually inserting tab before each indented line
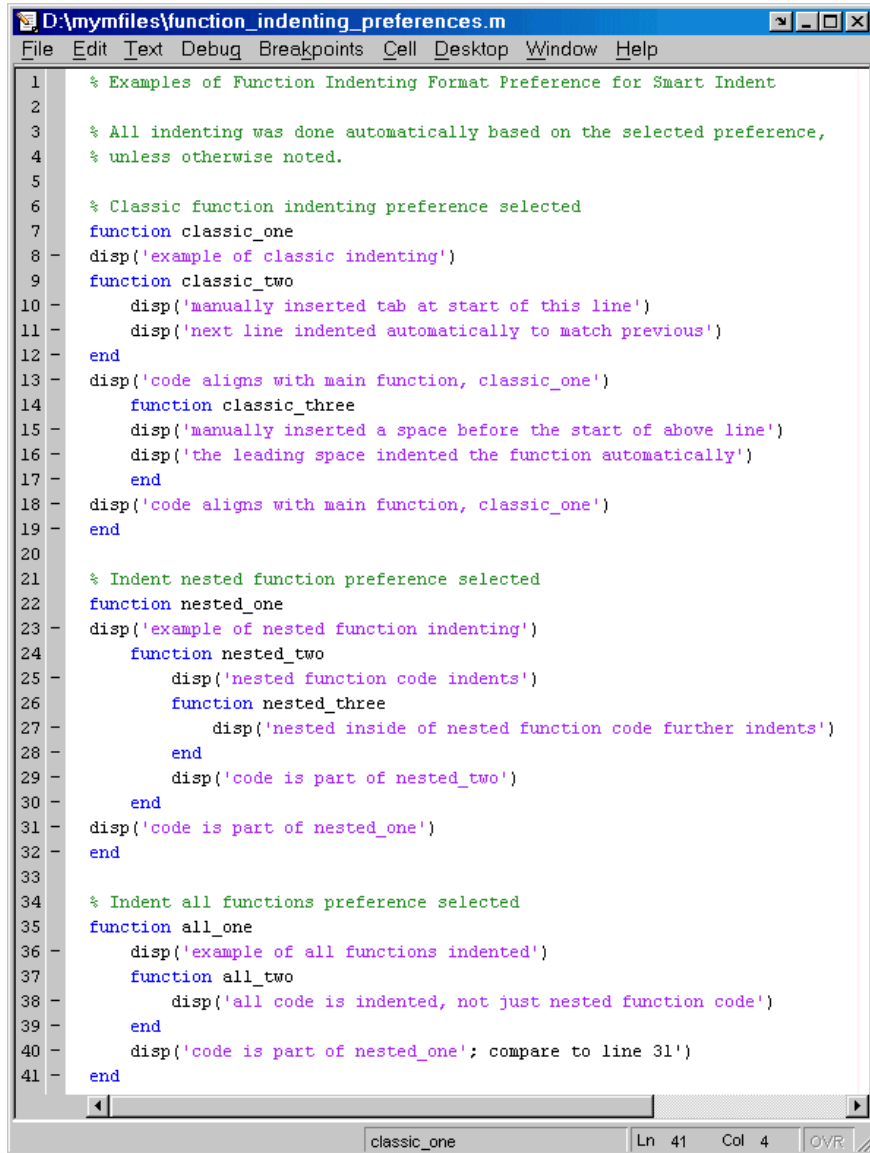
Example of Smart Indenting Deselected Without Tabs.

```
sequence = n
next_value = n;
while next_value > 1
if rem(next_value,2)==0
next_value = next_value/2;
else
next_value = 3*next_value+1;
end
sequence = [sequence, next_value]
end
```

Created with Smart indenting disabled and no tabs manually inserted.

Function Indenting Format. If you select **Enable smart indenting** preference, your selected function indenting format is enabled automatically. If you deselect **Enable smart indenting**, you can apply the function indenting format to selected text in the editor by pressing Ctrl-I. For nested functions, you must provide an `end` statement for each function statement in the file for indenting to be applied. The styles for indenting functions are

- **Classic** — All lines align left unless you manually insert a tab or space. If you insert a space before a `function` line, that line automatically indents, as do subsequent lines, until the next `end` statement.

- **Indent nested functions** — Automatically indents a nested function (`function` within a `function`) line. Statements within the nested function are indented further. An `end` statement aligns with its matching function if `end` appears at the start of the line.

- **Indent all functions** — Automatically indents statements within a function or within a nested function. An `end` statement aligns with its matching function if `end` appears at the start of the line.

Examples of Function Indenting Format Preference.

```
% Examples of Function Indenting Format Preference for Smart Indent

% All indenting was done automatically based on the selected preference,
% unless otherwise noted.

% Classic function indenting preference selected
function classic_one
disp('example of classic indenting')
function classic_two
    disp('manually inserted tab at start of this line')
    disp('next line indented automatically to match previous')
end
disp('code aligns with main function, classic_one')
    function classic_three
    disp('manually inserted a space before the start of above line')
    disp('the leading space indented the function automatically')
    end
disp('code aligns with main function, classic_one')
end

% Indent nested function preference selected
function nested_one
disp('example of nested function indenting')
    function nested_two
        disp('nested function code indents')
        function nested_three
            disp('nested inside of nested function code further indents')
        end
        disp('code is part of nested_two')
    end
disp('code is part of nested_one')
end

% Indent all functions preference selected
function all_one
    disp('example of all functions indented')
    function all_two
        disp('all code is indented, not just nested function code')
    end
    disp('code is part of nested_one'; compare to line 31')
end
```

Formatting Comments

Specify the **Max width**, that is, the maximum width, in number of columns, for M-file comments when you select the **Autowrap comments** preference. The maximum width also applies when you use the **Wrap Selected Comments** feature. See "Formatting Comments in M-Files" on page 6-42 for details.

For example, assume you select **Autowrap comments** and set the maximum width to be 75 characters, which is the width that fits on a printed page using the default font for the Editor. When typing a comment line, as you reach the 75th column, the comment automatically continues on the next line.

See also the "Right-Hand Text Limit" on page 6-18 preference for the Editor/Debugger.

File extensions

See "File Extensions" on page 6-21 for general Language preferences.

Setting TLC Language Preferences

Target Language Compiler (TLC) is an integral part of Real-Time Workshop®. Select **File > Preferences > Editor/Debugger > Language > TLC** to specify these preferences for editing TLC files:

- "Syntax Highlighting" on page 6-27
- "File Extensions" on page 6-28

See also "Setting Preferences for M-Lint" on page 6-145.

Syntax Highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify TLC constructs, such as commands and macros. Optionally, change the colors used for these elements.

Commands. Color for TLC commands, such as `LibBlockMatrixParameter`.

Comments. Color for the comment indicator, `%%` or `/%...%/`, and its associated text.

C Strings. Color for C strings.

Macros. Color for TLC macros.

File Extensions
See "File Extensions" on page 6-21 for general Language preferences.

Setting C/C++ Language Preferences
Select **File > Preferences > Editor/Debugger > Language** to specify these preferences for editing C or C++ language files:

- "Syntax Highlighting" on page 6-28
- "Indenting" on page 6-29
- "File extensions" on page 6-30

Syntax Highlighting
Select the **Enable syntax highlighting** check box to show colors that help you identify certain C constructs, such as methods. Specify the colors used for these elements.

Show methods. Text style for methods to appear in when you type them: `Bold`, `Italic`, or `Plain` (no special highlighting).

Keywords. Color for keywords, such as `if`.

Strings. Color for terms enclosed in double quotation marks, for example, `"default"`.

Characters. Color for terms enclosed in single quotation marks, for example, `'a'`.

Comments. Color for text following the comment indicator, `//`, as well as for the block comment indicators, `/*` and `*/`, and the code in between.

Preprocessor. Color for text following the preprocessor symbol, `#`.

Bad characters. Color for illegal characters.

Indenting

You can set preferences to specify if you want MATLAB to automatically apply indenting to your C/C++ files. The indenting options you specify, however, apply only to lines you enter after changing the preference; they do not affect the indenting for existing lines. For information on changing the indenting for existing lines, see "Manual Indenting" on page 6-51.

Enable Smart Indenting. Select this option to specify that you want:

- The body of loops to be automatically indented within the start and end of the loop statement.
- The lines you indent using tabs or spaces to result in subsequent lines automatically aligning with the indented line.

After entering a new line, use **Shift+Tab** to move text back to a previous indent level if you want.

Although you can manually insert tabs at the start of a line, be aware that smart indenting may not work properly. Use the **Text** menu entry for **Smart Indent** to correct selected lines.

Deselect **Enable smart indenting** to specify that you want:

- Lines to be aligned on the left by default
- To specify indenting manually using the tab and space keys

To set the indent size for smart indenting and the tab size for manual indenting, see "Setting Tab and Indent Preferences" on page 6-20.

For examples see:

- "Example of Smart Indenting" on page 6-24
- "Example of Smart Indenting Deselected with Manual Tabs" on page 6-24
- "Example of Smart Indenting Deselected Without Tabs" on page 6-25

File extensions

See "File Extensions" on page 6-21 for general Language preferences.

Setting Java Language Preferences

Select **File > Preferences > Editor/Debugger > Language > Java** to specify these preferences for editing Java files:

- "Syntax Highlighting" on page 6-30

- "Indenting" on page 6-31

- "File extensions" on page 6-31

See also "Setting Keyboard Preferences for Desktop Tools" on page 2-116 for the Editor/Debugger and Command Window and "Setting Language Preferences" on page 6-21.

Syntax Highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify certain Java constructs, such as methods. Specify the colors used for these elements.

Show methods. Text style for methods to appear in when you type them: `Bold`, `Italic`, or `Plain` (no special highlighting).

Keywords. Color for keywords, such as `if`.

Strings. Color for terms enclosed in double quotation marks, for example, `"alive"`.

Characters. Color for terms enclosed in single quotation marks, for example, `'a'`.

Comments. Color for text following the comment indicator, `//`, as well as for the block comment indicators, `/*` and `*/`, and the code in between.

Bad characters. Color for illegal characters.

Indenting

Select or deselect **Enable smart editing** to specify whether you want the Editor to apply indenting to your Java files when you press the **Enter** key to type in a new line. The effects are the same as those for C/C++ files. For details and examples, see the C/C++ preference for indenting.

File extensions

See "File Extensions" on page 6-21 for general Language preferences.

Setting XML/HTML Language Preferences

Select **File > Preferences > Editor/Debugger > Language > XML/HTML** to specify these preferences for editing XML, XSL, WSDL, HTML, HTM, and SHTML files:

- "Syntax Highlighting" on page 6-31
- "Indenting" on page 6-32
- "File extensions" on page 6-32

Syntax Highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify XML, WSDL, and HTML constructs, such as elements and tags. Optionally, change the colors used for these items.

Attribute Name. Color for attribute names.

Attribute Value. Color for values, such as the source for an image, for example `"myimage.gif"`, in `<img src= myimage.gif >`.

CDATA Section. Color for a CDATA section in XML files.

Character. Color for characters and entities, such as the nonbreaking space, ` `.

Comment. Color for text contained within comment indicators, `<!--` and `-->`.

DOCTYPE Declaration. Color for DOCTYPE declarations.

Error. Color for invalid entries, such as <+1> for font size, which is deprecated in favor of style sheets in the HTML 4.01 specification.

See the W3C Web site for details.

Operator. Color for operators, such as the equal sign (=).

Processing Instruction. Color of the instructions to the application that processes the XML file.

Tag. Color for tags and elements, such as <font> or <img>.

Indenting

Select or deselect **Enable smart editing** to specify whether you want the Editor to apply indenting to your files when you press the **Enter** key to type in a new line. The effects are the same as those for C/C++ files. For details and examples, see the C/C++ preference for indenting.

File extensions

See "File Extensions" on page 6-21 for general Language preferences.

Setting Code Folding Preferences

Select **File > Preferences > Editor/Debugger > Code Folding** to set these preferences for hiding and revealing code in M-files, including function and class code, function and class help code, programming control blocks, and so on:

- "Enable Code Folding for M-Files" on page 6-33
- "Enable Code Folding by Programming Construct" on page 6-33
- "Fold Initially" on page 6-33

See also "Appearance of an M-File — Making Files More Readable" on page 6-50, and Working with M-Files.

Enable Code Folding for M-Files

Select this option to enable code folding in M-files; clear this option to disable code folding in M-files. (Use the table that appears below this option to enable and disable code folding for selected programming constructs within an M-file.)

This option has the following effects:

- If you select this option, code folding is enabled for all programming constructs that you enable in the table that appears below this option in the Preferences dialog box.

- If you clear this option, code folding is disabled for all programming constructs, regardless of the programming constructs that you may have previously enabled in the table that appears below this option in the Preferences dialog box.

Enable Code Folding by Programming Construct

If you select the **Enable code folding for M-files** option, you can independently enable or disable code folding for individual programming constructs (function code, function help, programming control blocks, class code, and so on) by selecting or clearing the **Enable** check box that corresponds to each construct in the table on the Preferences dialog box. By default, code folding is enabled for all programming constructs except if/else blocks and switch/case blocks.

Fold Initially

If you enable code folding for a programming construct, you can specify how the Editor displays that construct the first time that you open an M-file that existed prior to MATLAB version 7.5. Select or clear the **Fold Initially** check box associated with a construct to direct the Editor to collapse or expand the associated construct, respectively, the first time you open an M-file that existed prior to MATLAB version 7.5 (R2007b) using MATLAB version 7.5 or later.

Setting Autosave Preferences

Select **File > Preferences > Editor/Debugger > Autosave** to specify these preferences for the Editor's autosave feature.

Enable autosave in the MATLAB Editor

The MATLAB Editor automatically saves a copy of the current version of the file you are editing. Clear this check box if you do not want the MATLAB Editor to automatically save the copy.

Save Options

Save every n minutes. Specify how often you want the Editor to automatically save a copy of the file you are editing.

Save untitled files. Select this check box if you want the Editor to automatically save a copy of new files that you have not yet saved, which are therefore untitled. If selected, the first autosave file is `Untitled.asv`. If the directory already contains a file named `Untitled.asv`, the autosave file is named `Untitled2.asv`, and so on for additional unnamed files.

If the autosave feature creates `Untitled.asv` and you subsequently save the file as `filename.m`, the next autosave version is `filename.asv`. `Untitled.asv` remains until you delete it.

Close Options

Automatically delete autosave files. With this preference selected, MATLAB deletes the autosave file when you close the source file in the Editor.

Filename

Specify the extension used for autosave files. The default setting for Microsoft Windows platforms is the extension `.asv` (for *autosave*), making the autosave file `filename.asv`. For Microsoft Windows and UNIX platforms, you can select **Replace extension with** and specify any extension.

For UNIX platforms, the default is **Append file with** the tilde (~) character, making the autosave file `filename.m~`. For Windows and UNIX platforms, you can select **Append file with** and specify a different string to append to the file.

Location

Specify the full path for the directory in which autosave files are stored. By default, each autosave file is stored in the same directory as the source file, that is, the file you are editing. You can specify a single directory for all autosave files, such as a directory you create called `autosave_files`. For the Editor to create an autosave file, you must have write-permission for the specified location.

Additional Information about Editor/Debugger Preferences

Additional information that relates to the Editor and preferences includes

- "Setting Autosave Preferences" on page 6-33

- "Setting Fonts Preferences for Desktop Tools" on page 2-121

- "Setting Colors Preferences for Desktop Tools" on page 2-131

- "Specifying Options for MATLAB Using Preferences" on page 2-104

Entering Statements in the Editor

In this section...
"Using Command Window Features in the Editor" on page 6-36
"Changing the Case of Selected Text" on page 6-37
"Undoing and Redoing Editor Actions" on page 6-37
"Adding Comments" on page 6-37
"Completing Statements in the Editor — Tab Completion" on page 6-43

Using Command Window Features in the Editor

After opening an existing file or creating a new file in the Editor, enter statements in the file. Use the same practices as for entering statements in the Command Window as described in Chapter 3, "Running Functions — Command Window and History":

- "Case and Space Sensitivity" on page 3-14

- "Entering Multiple Lines Without Running Them" on page 3-16

- "Entering Multiple Functions in a Line" on page 3-17

- "Entering Multiple-Line (Long) Statements — Line Continuation" on page 3-17

- "Suppressing Output" on page 3-48

- "Formatting and Spacing Numeric Output" on page 3-49

- "Matching Delimiters (Parentheses)" on page 3-25

- "Viewing Function Syntax While Entering a Statement — Function Hints" on page 3-34

- "Getting Help for the Selected Function in the Command Window or Editor" on page 3-39

- "Finding Functions Using the Function Browser" on page 3-41

Changing the Case of Selected Text

To change the case of text in the Editor, select the text. Then, from the **Text** menu, select one of the following:

- **Change to Upper Case** to change all text to uppercase
- **Change to Lower Case** to change all text to lowercase
- **Reverse Case** to change the case of each letter

This is useful, for example, when copying syntax from help in an M-file, where function and variable names are distinguished by the use of uppercase. But because of that, the code will not run in the MATLAB Editor or Command Window. In this example, the text was copied and pasted from the output of `help get`.

```
V = GET(H, 'Default')
```

Select all of the text. Select **Text > Change to Lower Case**. The text becomes

```
v = get(h, 'default')
```

If instead you select **Reverse Case** for

```
V = GET(H, 'Default')
```

the case changes to

```
v = get(h, 'dEFAULT')
```

Undoing and Redoing Editor Actions

You can undo many of the Editor actions listed in **Edit** and **Text** menus. Select **Edit > Undo**. You can undo multiple times in succession until there are no remaining actions to undo. Select **Edit > Redo** to reverse an undo.

Adding Comments

Comments in an M-file are strings or statements that do not execute. Add comments in an M-file to describe the code or how to use it. Comments determine what text displays when you run `help` for a file name. Use

comments when testing your files or looking for errors—temporarily turn lines of code into comments to see how the M-file runs without those lines. These topics provide details:

- "Commenting in M-Files Using the MATLAB Editor" on page 6-38
- "Commenting in Java and C/C++ Files Using the MATLAB Editor" on page 6-39
- "Commenting in M-File Using Any Text Editor" on page 6-39
- "Commenting Out Part of a Statement" on page 6-41
- "Formatting Comments in M-Files" on page 6-42

Commenting in M-Files Using the MATLAB Editor

You can comment the current line or a selection of lines in an M-file:

1 For a single line, position the cursor in that line. For multiple lines, click in the line and then drag or **Shift**+click to select multiple lines.

2 Select **Comment** from the **Text** menu, or right-click and select it from the context menu.

A comment symbol, %, is added at the start of each selected line, and the color of the text becomes green or the color specified for comments—see "Syntax Highlighting" on page 6-50.

To uncomment the current line or a selected group of lines, select **Uncomment** from the **Text** menu, or right-click and select it from the context menu.

Click in the area to the left of a line to select that line.
To select multiple lines, click+drag or shift+click.

Select **Text -> Comment** to make all the selected lines comments.
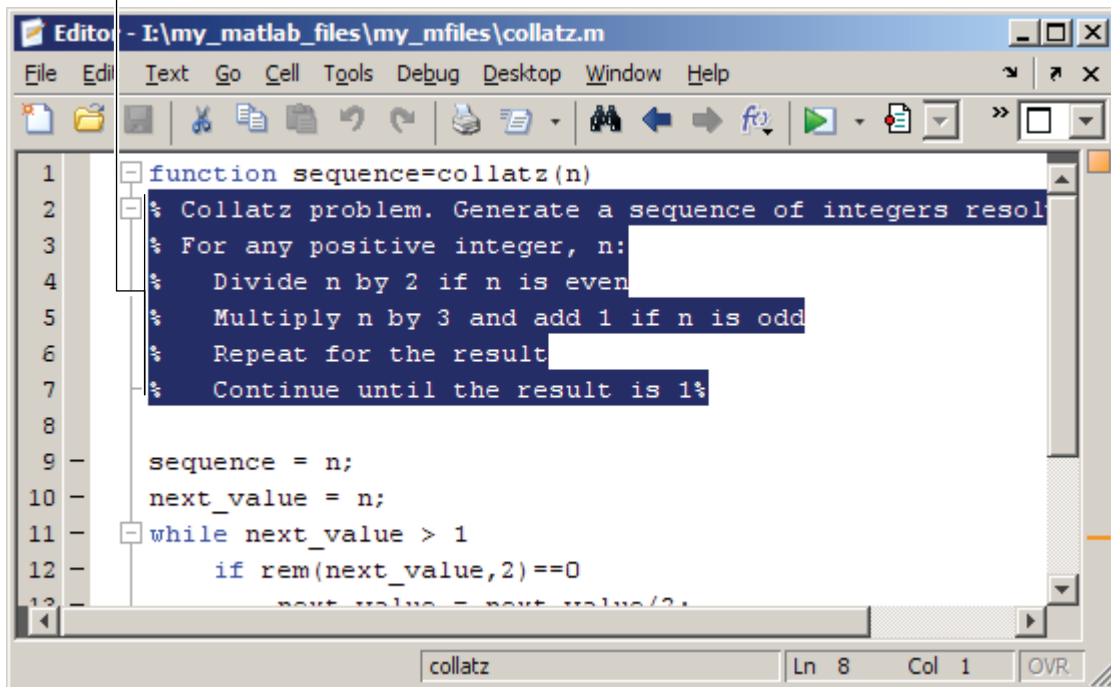
Commenting in Java and C/C++ Files Using the MATLAB Editor

For Java and C/C++ files, selecting **Text > Comment** adds the // symbols at the front of the selected lines. Similarly, **Text > Uncomment** removes the // symbols from the front of selected lines in Java and C/C++ files.

Commenting in M-File Using Any Text Editor

You can make any line in an M-file a comment by typing % at the beginning of the line. To put a comment within a line, type % followed by the comment text; MATLAB software treats all the information after the % on a line as a comment.

```
% This is a comment.
This is not a comment.
```

MATLAB ignores this comment line when you run the M-file.

This line produces an error when you run the M-file.

To uncomment any line, delete the comment symbol, %.

To comment a contiguous group of lines, type %{ before the first line and %} after the last line you want to comment. This is referred to as a block comment. The lines that contain %{ and %} can contain spaces, but not contain any other text. After typing the opening block comment symbol, %{, all subsequent lines assume the syntax highlighting color for comments until you type the closing block comment symbol, %}. Remove the block comment symbols, %{ and %}, to uncomment the lines.
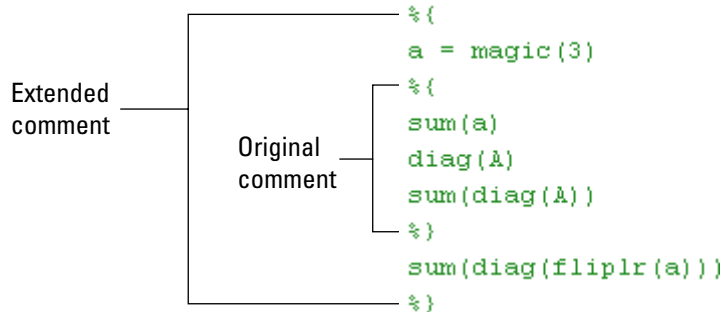
This examples shows some lines of code commented out. When you run the M-file, the commented lines will not execute. This is useful when you want to identify the section of a file that is not working as expected.

Comment a block of code by adding %{ before the first line and %} after the last line.

```
a = magic(3)
%{
sum(a)
diag(A)
sum(diag(a))
%}
sum(diag(fliplr(a)))
```

You can easily extend a block comment without losing the original block comment, that is, create a nested block comment, as shown in the following example.

Create a nested comment, that is, a block
comment within a block comment.

```
%{
a = magic(3)
%{
sum(a)
diag(A)
sum(diag(A))
%}
sum(diag(fliplr(a)))
%}
```

Extended comment — `%{` ... `a = magic(3)` ... `%}`

Original comment — `%{` ... `sum(diag(A))` ... `%}`

Commenting Out Part of a Statement

To comment out the end of a statement in an M-file, put the comment
character, %, before the comment. When you run the file, MATLAB software
ignores any text on the line after the %.

```
a = zeros(10) % Initialize matrix
```

Any text following a % within a line
is considered to be a comment.

To comment out text within a multiline statement, use the ellipsis (...).
MATLAB ignores any text appearing after the ... on a line and continues
processing on the next line. This effectively makes a comment out of anything
on the current line that follows the .... The following example comments
out the Middle Initial line.

```
header = ['Last Name, '...
'First Name, '...
... 'Middle Initial, '...
'Title']
```

MATLAB ignores the text following the ... on the line

```
... 'Middle Initial, '...
```

Note that `Middle Initial` is green, which is the syntax highlighting color for a comment.

MATLAB continues processing the statement with the next line

```
'Title']
```

MATLAB effectively runs

```
headers = ['Last Name, ' ...
'First Name, ' ...
'Title']
```

Formatting Comments in M-Files

To make comment lines in M-files wrap when they reach a certain column:

1 Specify the maximum column number using preferences for the Editor. Select **Language > M**. For **Comment formatting**, set the **Max width**.

2 Select contiguous comment lines that you want to limit to the specified maximum width.

3 Select **Text > Wrap Selected Comments**.

The selected comment lines are reformatted so that no comment line in the selected area is longer than the maximum. Lines that were shorter than the specified maximum are merged to make longer lines if they are at the same level of indentation.

To automatically limit comment lines to the maximum width while you type, select the **Comment formatting** preference to **Autowrap comments**.

For example, assume you select **Autowrap comments** and set the maximum width to be 75 characters, which is the width that will fit on a printed page using the default font for the Editor. When typing a comment line, as you reach the 75th column, the comment automatically continues on the next line.

Completing Statements in the Editor — Tab Completion

The Editor helps you automatically complete the names of these items as you type them in an M-file:

- Functions or models on the search path or in the current directory

- Class directories (including @-directories) and package directories

- Variables, including structures, in the current workspace, where the current workspace is shown in the **Stack** on the toolbar.

- Handle Graphics properties for figures in the current workspace

- MATLAB objects

Type the first few characters of the item name and then press the **Tab** key. To use tab completion, you must have the tab completion preference for the Editor selected. For details, see "Setting Keyboard Preferences for Desktop Tools" on page 2-116.

Tab completion is also available in the Command Window. There are a few minor differences in how tab completion works in the Command Window, the most notable being that Command Window tab completion supports the following, whereas the Editor tab completion does not:

- Completion of file names

- Help browsing

Note Tab completion does not complete the names of variables you define in an M-file, but only those variables in the current workspace. This means that while editing, it only completes the names of variables in the base workspace. While debugging, it only completes the names of variables in the current function workspace.

These examples demonstrate how to use tab completion:

Basic Example — Unique Completion

This example illustrates a basic use for tab completion in the Editor. In an M-file opened in the Editor, type the beginning of a function or model on the MATLAB search path or in the current directory, for example,

```
horz
```

and press **Tab**. The Editor automatically completes the name, which for this example displays the function name

```
horzcat
```

Complete the statement, adding any arguments, operators, or options. If the Editor does not complete the name horzcat but instead moves the cursor to the right, you do not have the preference set for tab completion. The Editor also moves the cursor to the right when you try to complete a file name; file name tab completion is not supported in the Editor, but is supported in the Command Window.

You can use tab completion anywhere in the line, not just at the beginning. For example, if you type

```
a = horz
```

and press **Tab**, the Editor completes horzcat.

The Editor also completes the names of variables in the current workspace. For example, if there is a variable costs_march in the currently selected workspace, type cost and press **Tab**. The Editor completes the variable name

costs_march. If the Editor displays No Completions Found, costs_march does not exist in the current workspace.

Multiple Possible Completions

If there is more than one name that starts with the characters you typed, when you press the **Tab** key, the Editor displays a list of all names that start with those characters. For example, assume you had created the variable costs_march in the base workspace. In an M-file in the Editor, type

```
cos
```

and press **Tab**. The Editor displays
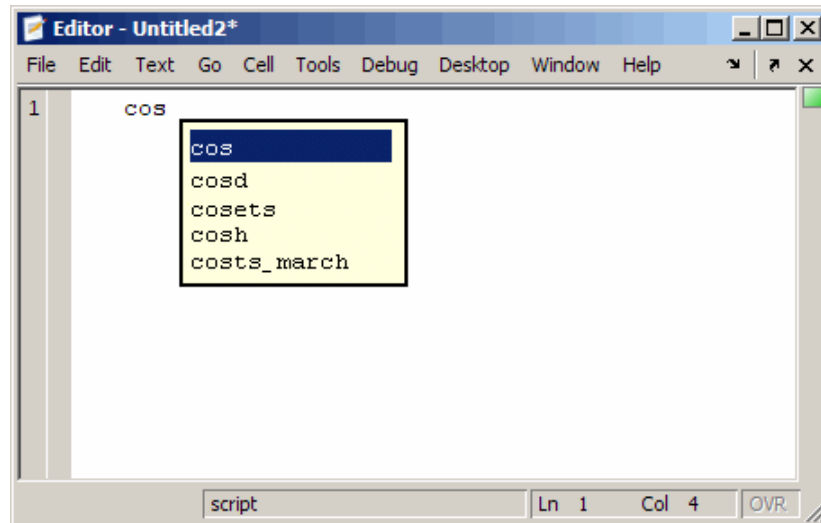
The resulting list of possible completions includes the variable name you created, costs_march, but also includes functions and models that begin with cos, including cosets from Communications Toolbox, if it is installed and on the MATLAB search path.

Continue typing to make your entry unique. For example, type the next character, such as t in the example. The Editor selects the first item in the list that matches what you typed, in this case, costs_march. Press **Enter** (or

Return) or **Tab** to select that item, which completes the name in the M-file. In the example, the Editor displays `costs_march` at the prompt.

You can navigate the list of possible completions using up and down arrow keys, and **Page Up** and **Page Down** keys. You can clear the list without selecting anything by pressing **Esc**. The list of possible completions might include items that are not valid commands, such as private functions.

Narrowing Completions Shown

You can narrow the list of completions shown by typing a character and then pressing **Tab** if the **Keyboard** preference **Tab key narrows completions** is selected. This is particularly useful for large lists. For example, type `cam` and press **Tab** to see the possible completions. There is a scroll bar with the list because there are too many completions to be seen at once.
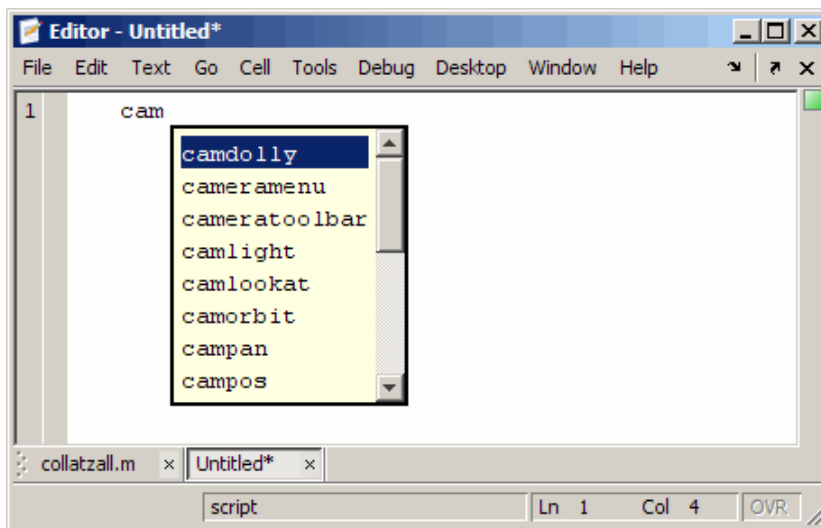
Type `p` and press **Tab** again. The Editor narrows the list, showing only all possible `camp` completions.
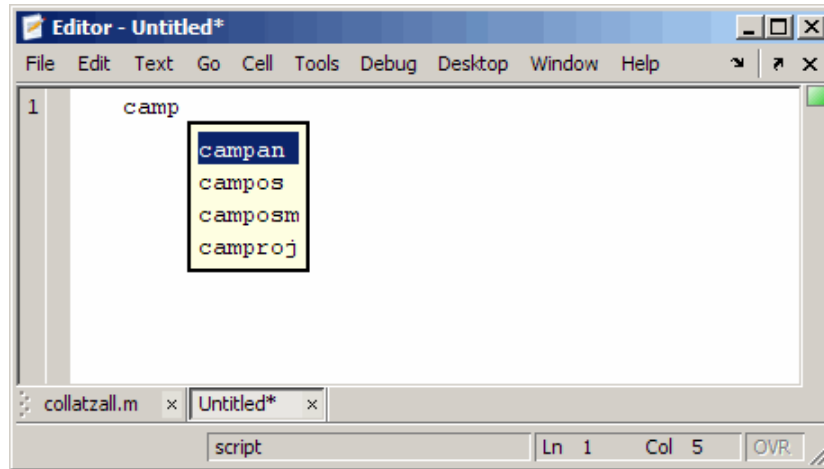
Continue narrowing the list in the same way. For the above example, type `o` and press **Tab** to further narrow the list. Press **Enter** or **Return** to select an item, which completes the name at the prompt.

Tab Completion for Structures

For structures that are in the current workspace, after the period separator, press **Tab**. For example, type

```
mystruct.
```

and press **Tab** to display all fields of `mystruct`. If you type a structure and include the start of a unique field after the period, pressing **Tab** completes that structure and field entry.

For example, type

```
mystruct.n
```

and press **Tab**, which completes the entry `mystruct.name`, where `mystruct` is in the current workspace and contains no other fields that begin with `n`.

Tab Completion for Properties

Complete property names for figures in the current workspace using tab completion, as in this graphics example. Here, f is a figure. Type

```
set(f, 'pap
```

and press **Tab**. The Editor displays

Select a property from the list. For example, type

```
u
```

and press **Enter**. The Editor completes the property, including the closing quote.

```
set(f, 'paperunits'
```

Continue adding to the statement, as in this example,

```
set(f, 'paperunits', 'c
```

and press **Tab**. The Editor automatically completes the property

```
set(f, 'paperUnits', 'centimeters'
```

because centimeters is the only possible completion.

Using Tab for Spacing

If the preference for tab completion is selected, and you want to also use the **Tab** key to add spacing within your statements, add a space before pressing **Tab**. For example, to create this statement

```
if a=mate     %test input value
```

add a space after `mate` and then press **Tab**. If you do not include the space, the following happens instead:

```
if a=material
```

This is because the tab completion feature automatically causes `mate` to complete as the `material` function.

Alternatively, turn off the tab completion preference to use **Tab** for spacing in the Editor.

Appearance of an M-File — Making Files More Readable

In this section...

Note You can specify the default behaviors for some of these features—see "Specifying Options for MATLAB Using Preferences" on page 2-104.

Syntax Highlighting

Some entries appear in different colors to help you better find matching elements, such as if/else statements. Similarly, unterminated strings have a different color than terminated strings. This is called syntax highlighting and is used in the Command Window and History, as well as in the Editor. For more information, see the Command Window documentation for "Highlighting Syntax to Help Ensure Correct Entries" on page 3-24.

When you paste or drag a selection from the Editor to another application, such as Microsoft Word, the pasted text maintains the syntax highlighting colors and font characteristics from the Editor. MATLAB software pastes the selection to the clipboard in RTF format, which many Microsoft Windows and Macintosh applications support.

Indenting

Automatic Indenting

You can set an indenting preference so that program control entries are automatically indented to make reading loops, such as while/end statements, easier. To do so, select **File > Preferences > Editor/Debugger > Language**, and select a **Language**, for example, MATLAB. Under **Indenting**, select **Enable smart indenting**, and then click **OK**. Deselect **Enable smart indenting** instead if you want to indent manually. For more information about indenting preferences, click **Help** in the Preferences dialog box. Specify the indenting size and other options by selecting **File > Preferences > Editor/Debugger > Tab**.

Manual Indenting

You can manually apply smart indenting to selected lines—select the lines and then select **Smart Indent** from the **Text** menu, or right-click and select it from the context menu. This feature indents lines that start with keyword functions or that follow lines containing certain keyword functions. Smart indenting can help you to read the code sequence.

To move the current or selected lines further to the left, select **Decrease Indent** from the **Text** menu. To move the current or selected lines further to the right, select **Increase Indent** from the **Text** menu.

You can also indent a line by pressing the **Tab** key at the start of a line. Or select a line or group or lines and press the **Tab** key. Press **Shift+Tab** to decrease the indent for the selected lines. This works differently if you select the Editor/Debugger **Tab** preference for **Emacs-style Tab key smart indenting**—when you position the cursor in any line or select a group of lines and press **Tab**, the lines indent according to smart indenting practices.

For more information about manual indenting, select **File > Preferences > Editor/Debugger > Tab** and then click **Help**.

Function Indenting

You can select from three indenting options when you enter a subfunction or a nested function (a function within a function) in the Editor. For details, see "Function Indenting Format" on page 6-25.

Line and Column Numbers

Line numbers display along the left side of the Editor window. You can elect not to show the line numbers using preferences. For details, select **File > Preferences > Editor/Debugger > Display**, and then click **Help**.

The line and column numbers for the current cursor position are shown in the far right side of the status bar in the Editor.

Highlight Current Line

You can set a preference to highlight the current line, that is the line with the caret (also called the cursor). This is useful, for example, to help you see where copied text will be inserted when you paste.

To highlight the current line, select **File > Preferences > Editor/Debugger > Display**, and under **General display options**, select the check box for **Highlight current line**. You can also specify the color used to highlight the line.

Current line (where the caret/cursor) is highlighted.
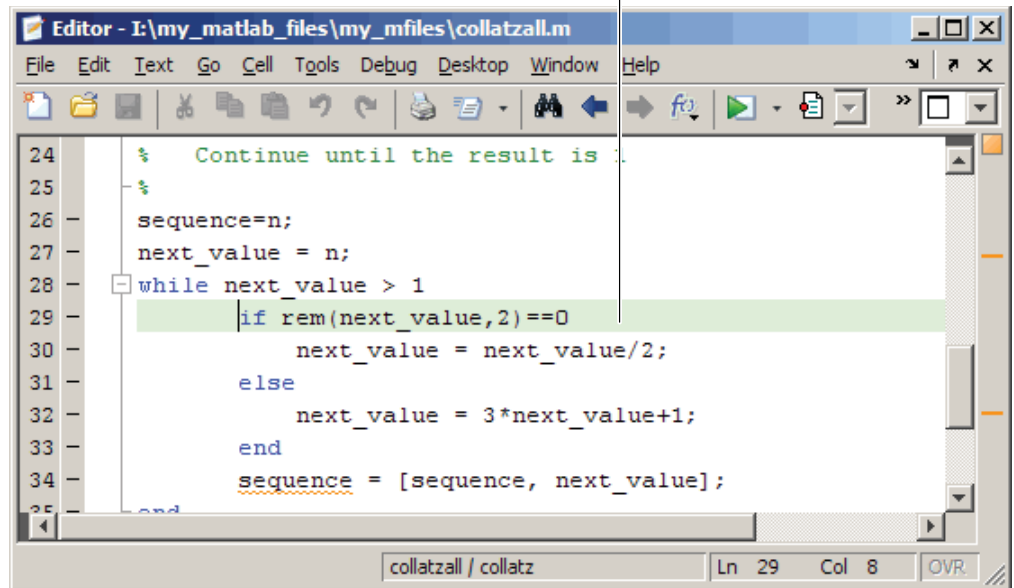
Right-Hand Text Limit

By default, a light gray vertical line (rule) appears at column 75 in the Editor, indicating where a line becomes wider than desired. This is useful, for example, if you need to keep each line below a limit imposed by another text editor in which you intend to view the code. You can change the width and color of the vertical line, as well as the column number at which the vertical line appears. If you want, you can hide the vertical line. For more information, select **File > Preferences > Editor/Debugger > Display**, and then click **Help**.

Note This limit is a visual cue only and does not prevent text from exceeding the limit. For information on setting a value to automatically wrap comment text at a specified column number, see "Formatting Comments in M-Files" on page 6-42.

Class, Function, or Subfunction

The right side of the Editor status bar shows the class, function, or subfunction where the cursor is currently placed, depending on the type of file you are viewing, as follows:

- Class file — The name of the class followed by the name of the current function (if any) that the cursor is within. This is true regardless of the type of function in which the cursor is placed (nested, in a methods block, outside a classdef file, and so on).

- Function file — The name of the main function followed by the name of the current function the cursor is within (if any). This is true regardless of the type of function in which the cursor is placed (subfunction or nested).

Code Folding — Expanding and Collapsing M-File Constructs

Code folding is the ability to expand and collapse certain M-file programming constructs. This improves readability when an M-file contains numerous subfunctions or other blocks of code that you want to hide when you are not currently working with that part of the file.

You can set preferences to enable or disable the ability to expand and collapse the following M-file programming constructs:

- Help block comments
- Cells used for rapid code iteration and publishing
- Class code
- Class enumeration blocks
- Class event blocks
- Class method blocks
- Class properties blocks
- For and parfor blocks
- Function and class help
- Function code

- If/else blocks

- Single program, multiple data (spmd) blocks

- Switch/case blocks

- Try/catch blocks

- While blocks
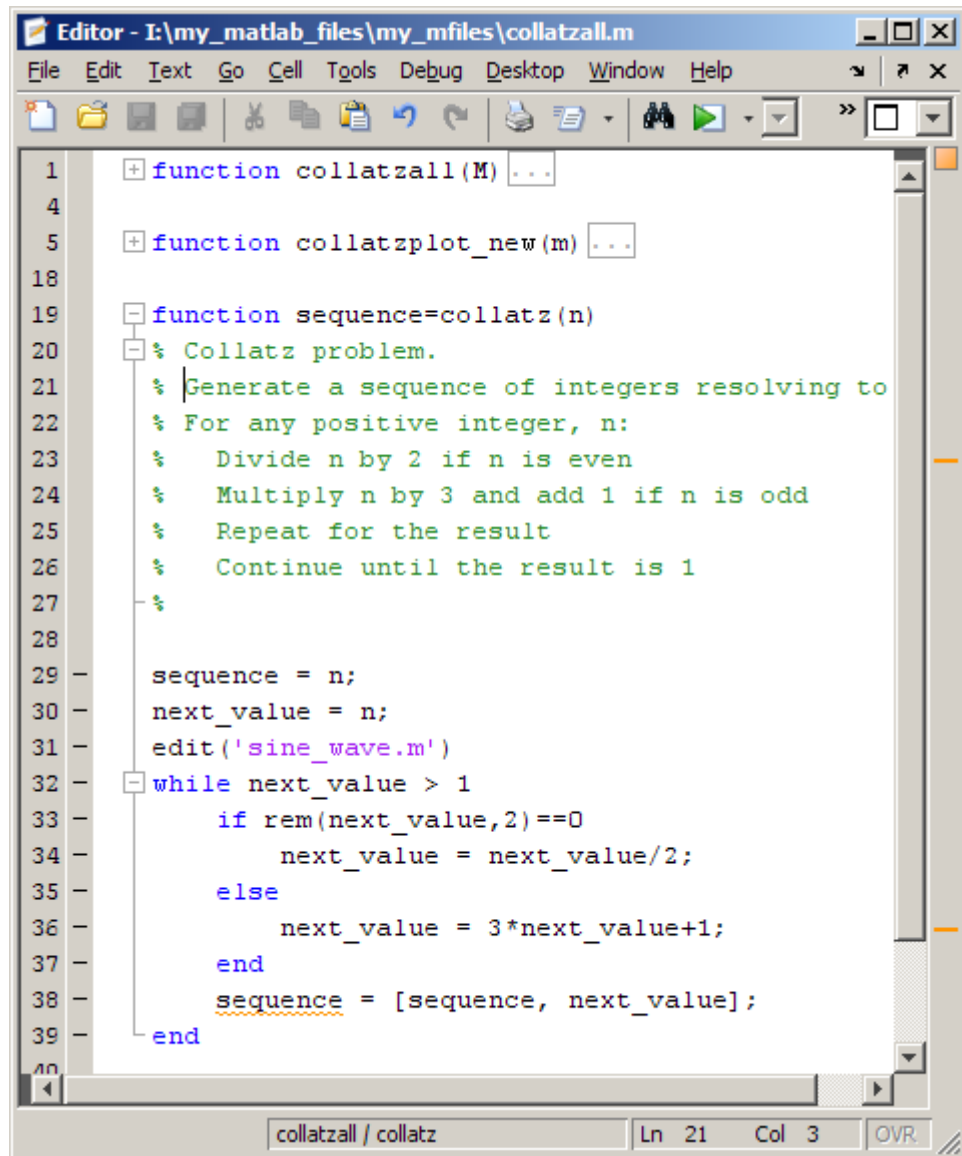
By default, code folding is enabled for all programming constructs except if/else blocks and switch/case blocks. Select **File > Preferences > Editor/Debugger > Code Folding**, and then click **Help** for details on setting preferences.

When you fold a construct, all the code associated with that construct is collapsed such that the Editor displays only the first line of the construct prepended by the plus sign (⊞) and appended with an ellipsis (⋯) to indicate there is more code. When you expand a construct, all the code associated with that construct appears and the first line of the construct is prepended with a minus sign (⊟).

To open the code used in the images in this section, enter the following in the Command window:

```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...
'examples','collatzall.m'))
```

The following image shows the `collatzall` and `collatzplot_new` functions collapsed and the `collatz` function code expanded.

When you expand a function or class, but collapse its associated help block code, the Editor displays all the function or class code and just the H1 line of

the help code. The H1 line ends with a commented ellipsis `%...%` to indicate there is additional help code, as shown in the following image.
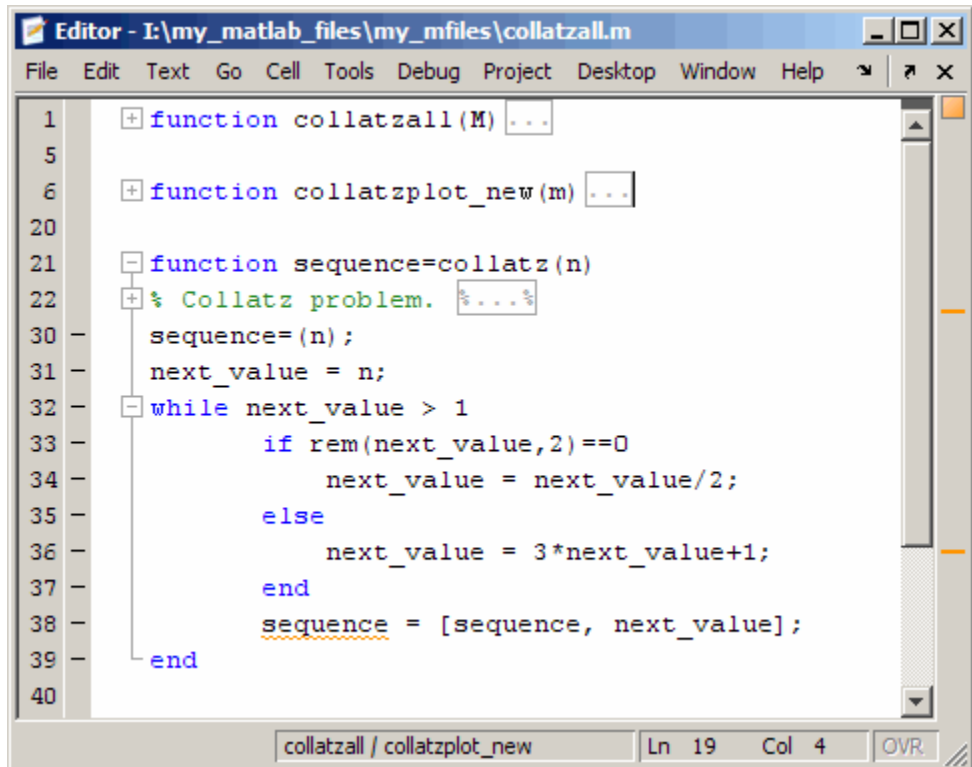
To expand code for a construct that is currently collapsed, do one of the following:

- Click the plus sign ⊞ to the left of the construct that you want to expand.

- Place your cursor in the code that you want to expand, right-click, and then select **Code Folding > Expand** from the context menu.

To collapse code for a construct that is currently expanded, do one of the following:

- Click the minus sign ⊟ to the left of the construct that you want to collapse.

- Place your cursor in the code that you want to collapse, right-click, and then select **Code Folding > Collapse** from the context menu.

To expand or collapse all of the code in an M-file, place your cursor anywhere within the M-file, right-click, and then select **Code Folding > Expand All** or **Code Folding > Collapse All** from the context menu.

For information on the structure of an M-file, including a description of a function definition line and an H1 line, see Basic Parts of an M-File in the Programming Fundamentals documentation.

Viewing Folded Code in a ToolTip

You can view code that is currently folded by positioning the pointer over its ellipsis [...]. The code appears in a ToolTip. This lets you quickly view the code without unfolding it.

The following image shows the ToolTip that appears when you place the pointer over the ellipsis on line 6 of collatzall.m when the collatzplot_new function is folded.
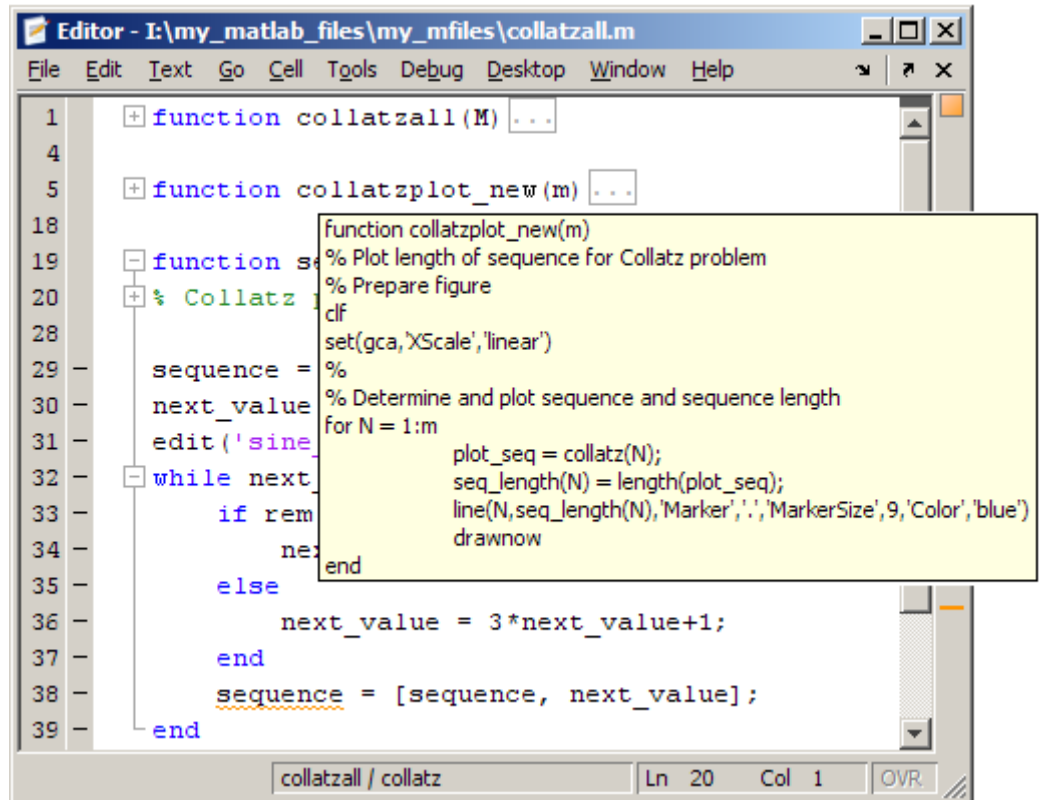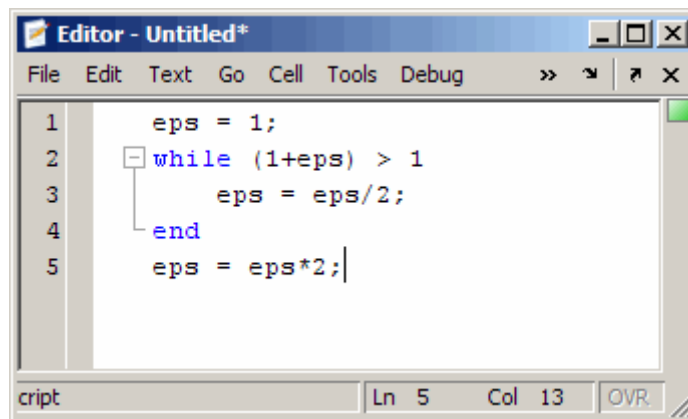
Code Folding Behavior and Preferences

Be aware of the following:

- You can change the current code folding settings, by selecting **File > Preferences > Editor/Debugger > Code Folding**. If needed, click **Help** for assistance.

- By default, the first time you open an M-file that existed before MATLAB Version 7.5 (R2007b) using MATLAB Version 7.5 (R2007b) or later, code folding is enabled and all constructs are expanded.

- Constructs that are collapsed when you close an M-file remain collapsed when you reopen the file.

- If you copy a collapsed construct from one region of an M-file and paste it in another region, the construct is expanded in the pasted location.

- If you print a file with one or more collapsed constructs, those constructs are expanded in the printed version of the file.

- If your code contains syntax errors, the code folding indicators may appear to be placed in the wrong location. For example, suppose your code currently appears as shown in the first figure that follows. If you delete the `while` statement, it introduces a syntax error at line 3, as shown in the second figure that follows. Notice that the minus sign remains in the same location it held for the syntactically correct code. After you correct the syntax error, the Editor adjusts and displays the code folding indicators appropriately.

```
Editor - Untitled*                              _ □ X
File  Edit  Text  Go  Cell  Tools  Debug    »  ↘  ↗ X
1        eps = 1;
2    ⊟ eps = eps/2;
3    └ end
4        eps = eps*2;

cript                        Ln  2      Col  1    OVR
```
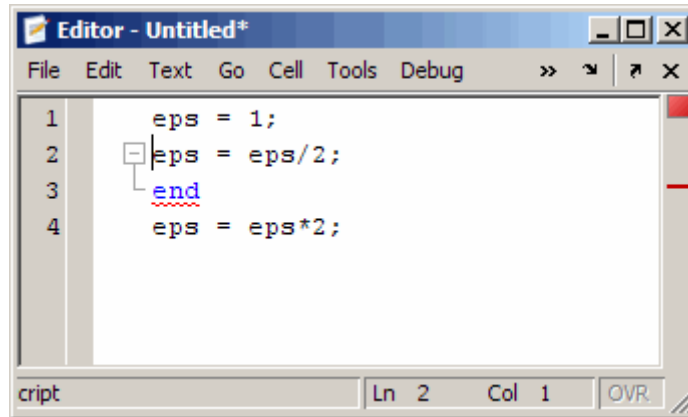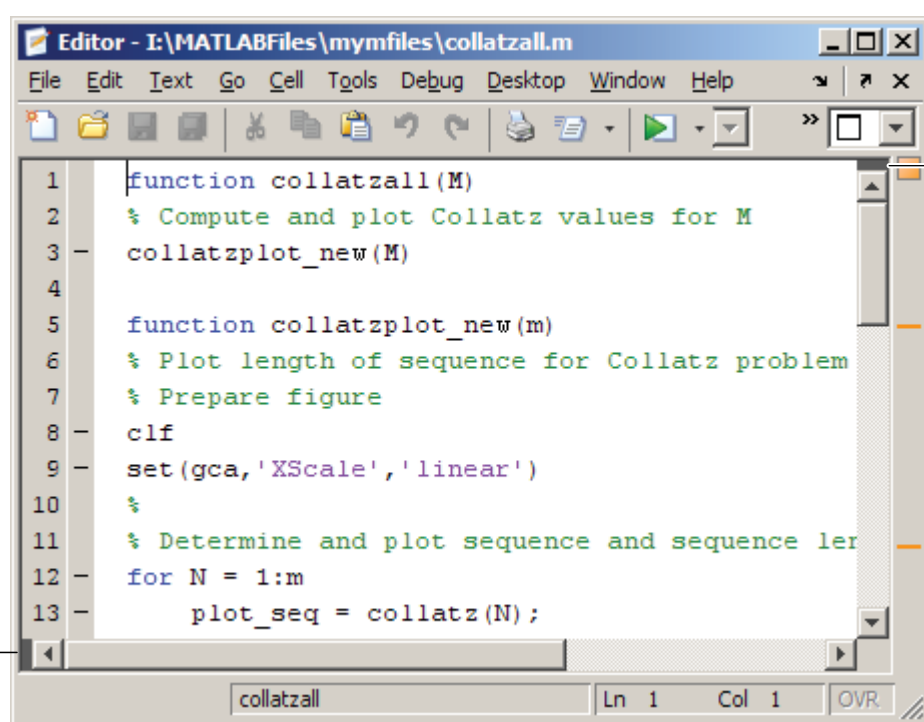
Split Screen Display

You can simultaneously display two different parts of a file in the Editor. This makes it easy to compare different lines in a file or to copy and paste from one part of a file to another.

Split the screen horizontally by selecting **Window > Split Screen > Top/Bottom**. Or to split it vertically, select **Left/Right**.

Alternatively, when there is a scroll bar, split the document into top and bottom views by dragging the splitter bar, as shown in the following illustration, down from above the vertical scroll bar. Similarly, to split into left and right views, drag the splitter bar from the left of the horizontal scroll bar. The mouse pointer assumes a double-headed arrow shape when you position it on the splitter bar.

Drag splitter bar down to
create top and bottom views.
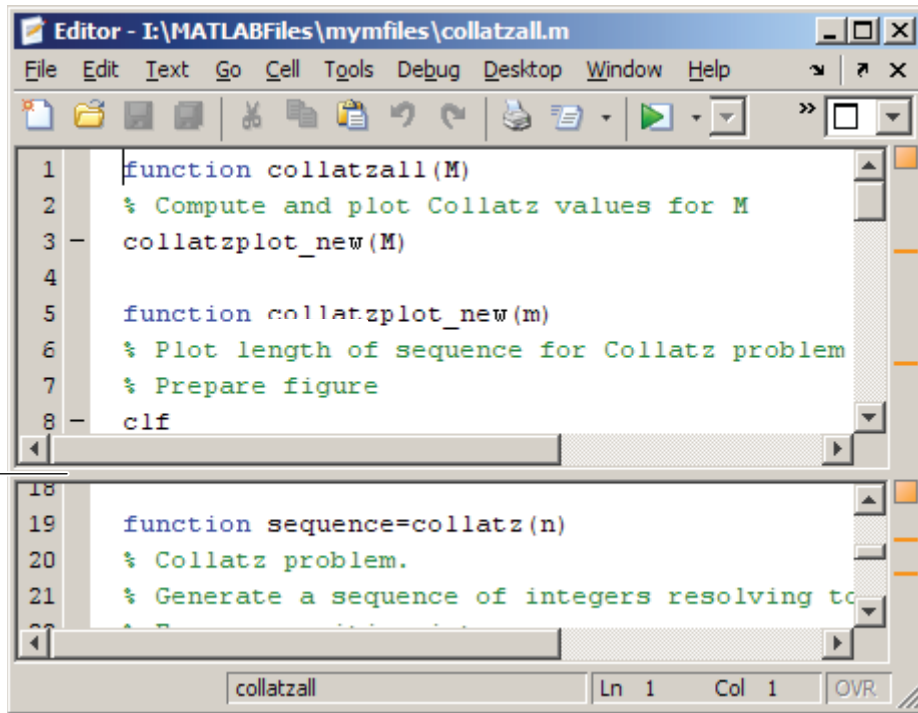
```
Editor - I:\MATLABFiles\mymfiles\collatzall.m              _ □ ×

File  Edit  Text  Go  Cell  Tools  Debug  Desktop  Window  Help      ↘ | ↗ ×

  1      function collatzall(M)
  2      % Compute and plot Collatz values for M
  3 –    collatzplot_new(M)
  4
  5      function collatzplot_new(m)
  6      % Plot length of sequence for Collatz problem
  7      % Prepare figure
  8 –    clf
  9 –    set(gca,'XScale','linear')
 10      %
 11      % Determine and plot sequence and sequence ler
 12 –    for N = 1:m
 13 –        plot_seq = collatz(N);

        collatzall              Ln  1      Col  1      OVR
```

Drag splitter
bar right to
create left and
right views.

Double-click the splitter to remove the split.

Drag the splitter to resize the views.
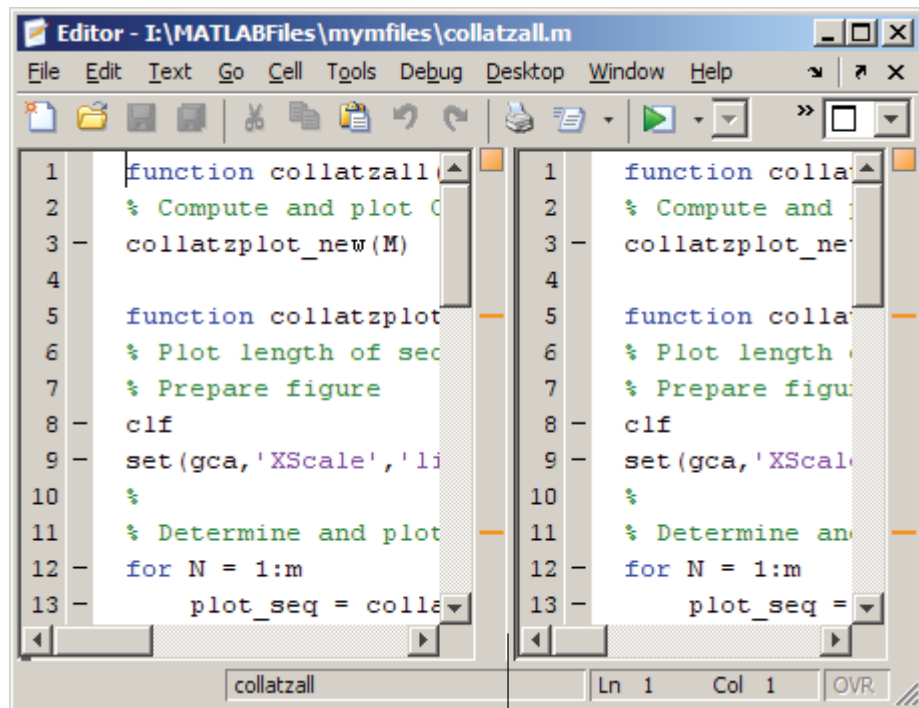
Double-click the splitter to remove the split.

Drag the splitter to resize the views.

Resize of the views by dragging the splitter. The mouse pointer assumes an arrow shape when you position it on the splitter.

Only one view is active at any time, meaning, you will see only the cursor in one of the views. To change the active view, select **Window > Split Screen > Switch Focus** or its keyboard equivalent, which is shown with the menu item. The cursor returns to its last position in that view.

Make changes to the document in either view. Both views of the file are always current, so you see the changes in either view.

You split each open document individually, so there can be multiple views at once. You can split some documents horizontally, others vertically, and

leave others unsplit. When you open a document, it always opens unsplit, regardless of its split status when you last had it open.

You can remove a document split using any of these methods:

- Drag the splitter to an edge of the window.

- Double-click the splitter.

- Select **Window > Split > Screen > Off**.

See also "Positioning Documents" on page 2-26 for instructions on displaying multiple documents simultaneously.

Navigating in an M-File

In this section...
"Going to a Line Number" on page 6-66
"Going to a Function (Subfunctions and Nested Functions)" on page 6-66
"Going to a Bookmark" on page 6-67
"Navigating Back and Forward in Files" on page 6-68
"Opening a Selection in an M-File" on page 6-71

Note See also "Finding Text in Files" on page 6-73.

Going to a Line Number

Select **Go > Go To**. In the resulting **Go To** dialog box, select the **Line number** option, enter a line number, and click **OK**. The cursor moves to that line number in the current M-file.

Going to a Function (Subfunctions and Nested Functions)

To go to a function within an M-file (either a subfunction or a nested function), select **Go > Go To**. In the resulting **Go To** dialog box, select the **Function** option, and then select an entry from the list of subfunctions and nested functions in the file. Click **OK**.
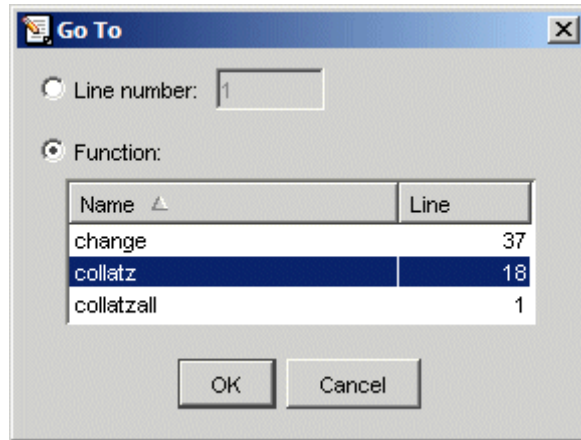
Functions in the list appear alphabetically by name. To order them by their position in the file, click the **Line** column heading. The list does not include functions that are called from the M-file, but only shows lines in the current M-file that begin with a `function` statement.

Alternatively, click the Show Functions button f_2 on the toolbar. Then select the subfunction or nested function you want to go to from the list. For both class and function files, the functions are listed in alphabetical order—except that in function files, the name of the main function always appears at the top of the list.

Going to a Cell

For M-file scripts that contain cells, the **Go To** dialog box lists cell titles.

Going to a Bookmark

You can set a bookmark at a line in a file in the Editor so you can quickly go to the bookmarked line. This is particularly useful in long files. For example, while working on a line, if you need to look at another part of the file and then return, set a bookmark at the current line, go to the other part of the file, and then go back to the bookmark.

To set a bookmark, position the cursor anywhere in the line and select **Go > Set/Clear Bookmark**. A bookmark icon appears to the left of the line.

```
11 - □ while next_value > 1
```

To go to a bookmark, select **Next Bookmark** or **Previous Bookmark** from the **Go** menu.

To clear a bookmark, position the cursor anywhere in the line and select **Go > Select/Clear Bookmark**.

Bookmarks are not maintained after you close a file.

Navigating Back and Forward in Files

Use **Go > Back** (and **Go > Forward**) to go to lines you previously edited or navigated to in a file. The feature goes to the lines in the sequence you accessed them. As an alternative to the menu items, use the Back and Forward buttons on the toolbar.

Use Back and Forward buttons or menu items to navigate to lines you previously edited or navigated to.
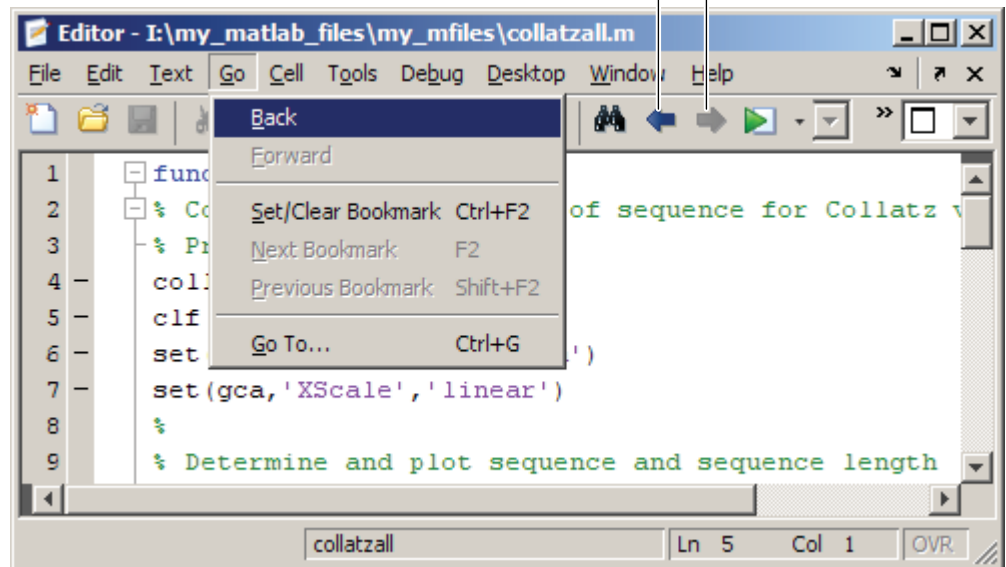
For example, if you open a file and make changes at lines 3, 9, and 6, use **Go > Back** to return to line 9, then 3, then 1, and then use **Go > Forward** to go from 1 to 3 to 9 to 6, and then return to 3. Detailed instructions to accomplish this are:

1 Select **Go > Back** to return from line 6 to line 9.

2 Select **Go > Back** again to return to line 3.

3 Select **Go > Back** again to return to line 1, which is the first line you originally navigate to in a file by virtue of opening it.

4 Use **Go > Forward** to reverse the direction of the feature—select **Go > Forward** to navigate to line 3.

5 Select **Go > Forward** to navigate to line 9.

6 Reverse the direction of the feature again—select **Go > Back** to navigate to line 3.

Lines Navigated to Using Go Back

Use **Go > Back** and **Forward** to go to lines you previously edited or to which you navigated using these features:

Feature	Examples	Notes
Opening a file (first line in the file)	**File > Open**	None
Changes made using text-editing tools	**Delete** key, or **Text > Increase Indent**	Edits made to a selection of lines are represented by the first line in the selection. Changes made using **Cell > Insert Cell Break** and **Cell > Insert Text Markup** are not considered as having been previously navigated to.
Changes made using Find and Replace	**Edit > Find and Replace**	Changes made using **Replace All** are not considered as having been previously navigated to.

Feature	Examples	Notes
Find features	**Edit > Find and Replace**, **Find Next**, **Find Previous**, and **Find Selection**	None
Incremental search	**Ctrl+S** and **Ctrl+R**	None
Show Function button	*fx*	None
Opening a selection	**File > Open Selection**	None
Go to	**Go > Go To** line number, function, or cell title	None
Bookmark navigation	**Go > Next Bookmark** and **Previous Bookmark**	A line at which you **Set/Clear Bookmark** is not considered as having been previously navigated to.
Hyperlink access	From warnings or errors in the Command Window, from **Find Files** results, and from reports like the Profiler	None
Debugging navigation	Lines with breakpoints that were stopped at while running, and lines stepped to	A line at which you set a breakpoint is not considered as having been previously navigated to, unless it was actually stopped at during execution.
Cell mode navigation	**Cell > Next Cell** and **Previous Cell**, and **Cell > Evaluate Current Cell and Advance**	Lines accessed using **Cell > Evaluate Current Cell** are not considered as having been previously navigated to.

Interrupting the Sequence of Go Back and Forward

If you use **Go > Back** and **Go > Forward**, and then edit another line or navigate to another line using the list of features described in the above table, the **Go > Back** or **Go > Forward** feature sequence is interrupted. You can still go to the lines preceding the interruption point in the sequence, but you cannot go to any lines after that point. Any lines you edit or navigate to after interrupting the sequence are added to the sequence after the interruption point.

For example:

1 Open a file and edit lines 2, then 4, and then 6.

2 Use **Go > Back** to move back to line 4, and then back to line 2.

3 You could then **Go > Forward** to lines 4 and 6, or **Go > Back** to line 1.

Instead, make an edit at line 3. Now you cannot **Go > Forward** to lines 4 and 6 and you can only **Go > Back** to line 2 and then line 1.

Closed Files and Behavior of Go Back and Forward

Go > Back and **Forward** do not go to lines in closed files.

Split Screen and Behavior of Go Back and Forward

When you have a split screen display, **Go > Back** and **Forward** go to the view in which the line was originally navigated to or edited in. If you remove the split, **Go > Back** and **Forward** do not go to any lines that were visited in the lower (or right) view.

Opening a Selection in an M-File

You can open a subfunction, function, file, variable, or Simulink model from within a file in the Editor. Position the cursor in the name and then right-click and select **Open Selection** from the context menu. Based on what the selection is, the Editor performs a different action, as described in the table that follows.

Selection	Action
Subfunction	Cursor moves to the subfunction within the current M-file. If no subfunction by that name is found in the current M-file, the Editor runs the open function on the selection, which opens the selection in the appropriate tool, as shown for the other selection types in this table.
M-file or other text file	Opens in the Editor.

Selection	Action
Figure file (`.fig`)	Opens in a figure window.
Variable	Opens in the Variable Editor.
Model	Opens in Simulink.
Other	If the selection is some other type, **Open selection** looks for a matching file in a private directory in the current directory and performs the appropriate action.

Finding Text in Files

Finding Text in the Current File

Within the current file, select the text you want to find. From the **Edit** menu, select **Find Selection**. The next occurrence of that text is selected. Select **Find Selection** again (or **Find Next**) to continue finding more occurrences of the text.

To find the previous occurrence of selected text (find backwards) in the current file, select **Find Previous** from the **Edit** menu. The previous occurrence of the text is selected. Repeat to continue finding the previous occurrences of the text.

Finding and Replacing Text in the Current File

You can search for specified text within multiple files, and then replace the text within a file.

Finding Text

To search for text in files, click the Find button 🔍 in the Editor toolbar, or select **Edit > Find and Replace**. Complete the resulting **Find Replace** dialog box.
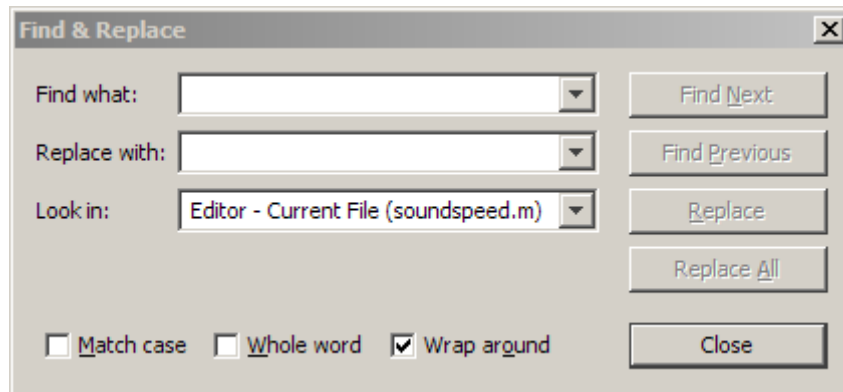
The search begins at the current cursor position. The Editor finds the text you specified and highlights it. To find another occurrence, click **Find Next** or **Find Previous**, or use the keyboard shortcuts **F3** and **Shift+F3** (or **Command+F3** and **Command+Shift+F3** with Macintosh key bindings).

The MATLAB software beeps when a search for **Find Next** reaches the end of the file, or when a search for **Find Previous** reaches the top of the file. If you have **Wrap around** selected, it continues searching after beeping.

Use **F3** and **Shift+F3** to continue finding the specified text even after closing the **Find Replace** dialog box. You can go to another file and find the specified text in it.

Change the selection in the **Look in** field to search for the specified text in other Microsoft desktop tools.

Replacing Text

After finding text using the **Find Replace** dialog box, you can replace the text in the current file:

1 In the **Replace with** field, type the text that is to replace the found text.

2 Click **Replace** to replace the text currently selected, or click **Replace All** to replace all instances in the current file.

The text is replaced. For **Replace All**, the number of instances that were replaced appears in the Editor status bar.

3 To save the changes to the file, select **Save** from the **File** menu.

You can repeat this for multiple files.

Function Alternative for Finding Text

Use lookfor to search for the specified text in the first line of help for all M-files on the search path.

Finding Files or Text in Multiple Files

To find directories and file names that include specified text, or whose contents contain specified text, use **Edit > Find Files**. For details, see "Finding Files and Content Within Files in Any Directory" on page 5-82.

Incremental Search

With the incremental search feature, the cursor moves to the next or previous occurrence of the specified text in the current file. It is similar to the Emacs search feature. Incremental search is also available in the Command Window—see "Performing an Incremental Search in the Command Window" on page 3-55.

To use the incremental search feature in the Editor, follow these steps:

1 Position the cursor where you want the search to begin.

2 How you begin the incremental search depends on your setting for the Editor/Debugger key bindings preference and in which direction you want to search:

- Press **Ctrl+S** to search forward or **Ctrl+R** to search backward for Emacs and Macintosh key bindings.

- Press **Ctrl+Shift+S** to search forward or **Ctrl+Shift+R** to search backward for Windows key bindings.

An incremental search field appears in the left side of the status bar of the current file window. **F Inc Search** means search *F*orward from the cursor. The field label is instead **R Inc Search** when you search backwards.
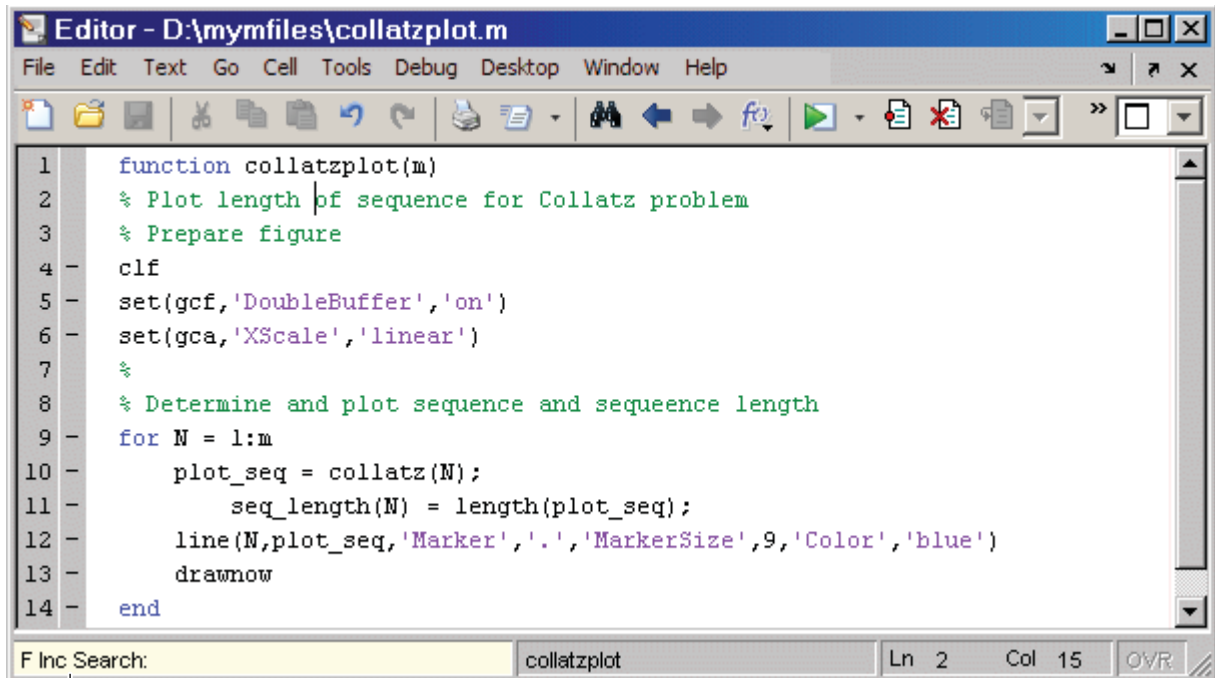
Incremental search field.

F means search *forward* from the cursor.

3 In the incremental search field, type the text you want to find. For example, type `plot`.

As you type the first letter, `p`, the first occurrence of that letter after the cursor is highlighted. In the example shown, the cursor is in the middle of line 2, so the first occurrence of `p`, the `p` in `problem` on line 2, is highlighted.

```
1    function collatzplot(m)
2    % Plot length of sequence for Collatz problem
3    % Prepare figure
```

Incremental search is case sensitive for uppercase letters. In the above example, searching for uppercase `P`, would instead find the `P` in `Prepare` on line 3.

When you type the next letter in the term you are searching for, the first occurrence of the term becomes highlighted. In the example, when you add the letter `l` to the `p` so that the incremental search field now has `pl`, the `pl` in `plot` on line 8 is highlighted. When you add `ot` to the term in the incremental search field, the whole word `plot` in line 8 is highlighted.

- If you mistype in the incremental search field, use the backspace key to remove the last letters and make corrections.

- After finding the `p`, press **Ctrl+W** to highlight the rest of the word found, in this case `plot`, which also puts the complete word in incremental search field.

4 To find the next occurrence of `plot` in the file, press **Ctrl+S**. To find the previous occurrence of the text, press **Ctrl+R**.

5 If MATLAB beeps, it either means the search is at the end or beginning of the file, or it means that the text was not found.

- When the text is not found, `Failing` appears in the incremental search field. Modify the search term in the incremental search field and try again. Use **Ctrl+G** to automatically remove characters back to the last successful search. For example, if `plode` fails, **Ctrl+G** removes the `de` from the search term because `plo` does exist in the file.

- When at the end or beginning of the file, press **Ctrl+S** or **Ctrl+R** again to wrap to the beginning (or end) of the file and continue the search. Use **Ctrl+G** after a finding a string to clear the search and return the cursor to the starting point.

6 To end the incremental search, press **Esc** or **Enter**, or any other noncharacter or number key except **Tab** or backspace.

The incremental search field no longer appears in the status bar. The cursor is now located at the position where the string was last found.

If you press **Ctrl+S** or **Ctrl+R** after displaying the blank incremental search field, the search term from your previous incremental search appears in the field. Then the backspace key deletes the entire previous search term, rather than just the last letter.

Comparing Files and Directories

What Is the File and Directory Comparisons Tool?

The File and Directory Comparisons tool determines and displays the differences between two files or two directories.

You can use this tool to:

- Compare lines in two text files (some other applications refer to this as a file diff operation).

- Compare variables in two MAT-files.

- Determine whether the contents of two binary files are the same.

- Compare two directories to determine which file names are unique to each directory.

- Compare two directories to determine if files with the same name in each directory have the same content.

Comparing Two Text Files

When you use the File and Directories Comparisons tool to compare two text files, a window opens and presents the two files side by side, along with symbols to indicate how you can adjust the files to make them match. This

is useful, for example, when you want to compare the latest version of a text file to an autosave version.

To compare two text files, follow these steps:

1 Open one of the text files you want to compare in the Editor.

To open the example file provided, `lengthofline.m`, run the following command in the Command Window:

```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...
  'examples','lengthofline.m'))
```

2 Select **Tools > Compare Against > Browse**. Navigate to the file you want to compare against, select the file, and click **Open**. To open the example file provided, select `lengthofline2.m` from the directory where you found `lengthofline.m`. Other options available are the following:

- **Tools > Compare Against > Autosave Version** to compare the open file to the Editor's automatic copy, *file name*`.asv`. For more information, see "Autosave" on page 6-99.

- **Tools > Compare Against Version on Disk** to compare an open file that has been changed, but not saved, to the saved version.

The File and Directory Comparisons tool opens, displaying the files side by side and highlighting lines that do not match, as follows:

- Pink highlighting and an x at the start of a line indicate that the content of the lines differs between the two files.

- Green highlighting and a > at the start of a line indicate a line that exists in the file presented on the right, but not in the file presented on the left.

- Green highlighting and a < at the end of a line indicate a line that exists in the file presented on the left, but not in the file presented on the right.
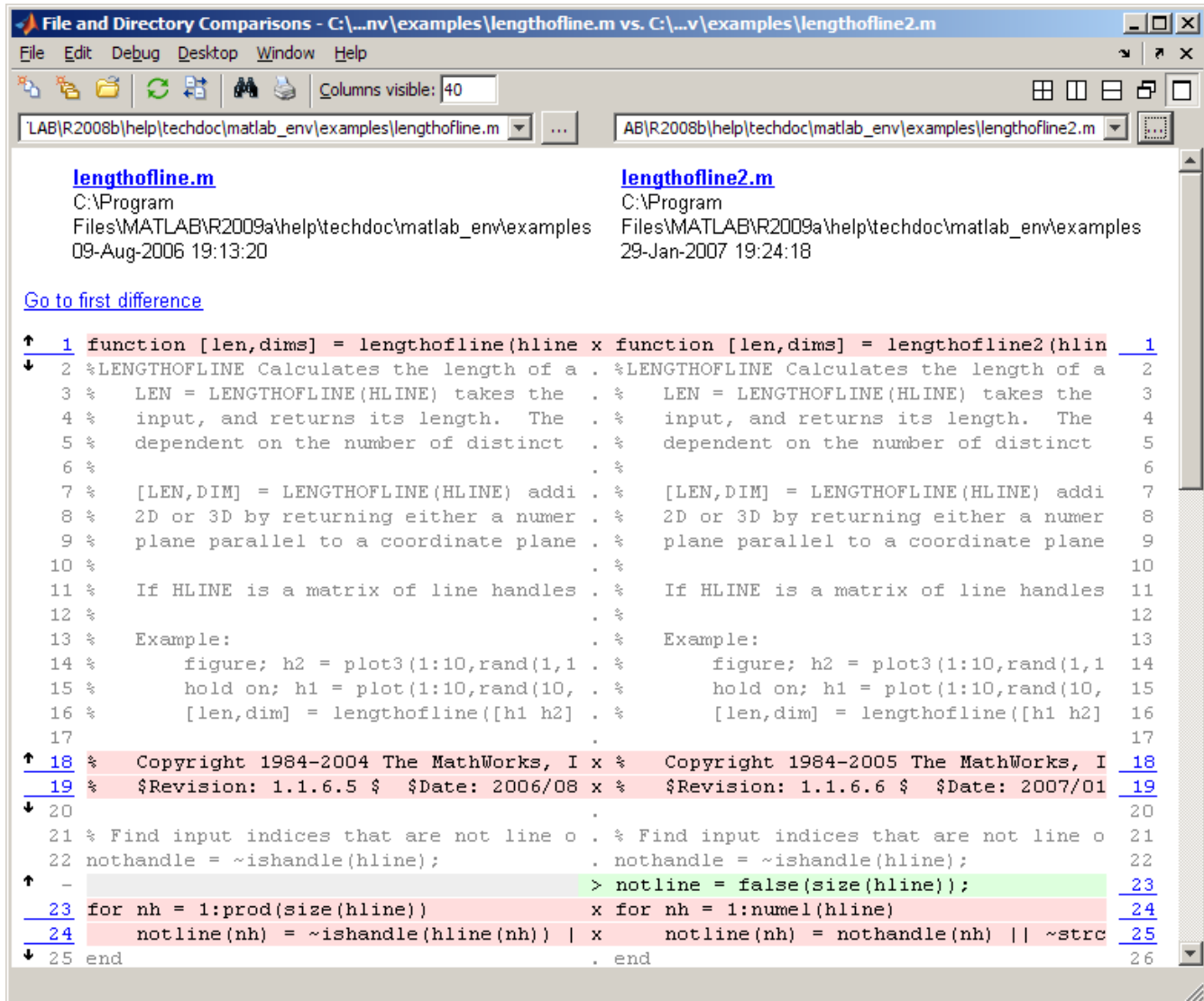
3 Use the features of the File Comparison and Directory Comparison tool to work with the results.

Typically, when this tool compares two text files, it does not do a simple line-by-line comparison. In the previous image, for example, the tool determines that `lengthofline2.m`, has a line of code that does not exist in `lengthofline.m`, and highlights it (line 23) in green. Also notice that the tool takes the additional line into account and determines that the line containing the `end` statement in each file matches, even though the `end` statement does not occur on the same line number.

If the files being compared are extremely long, however, the tool may run out of memory in attempting to perform the file comparison. It then displays the message, `Maximum file length exceeded.  Defaulting to line-by-line comparison.` In this case, the tool highlights the lines containing the `end` statement because in performing a simple line-by-line comparison it finds that the last line in one file does not match the last line in the other file.

Comparing Two MAT-Files

When you use the File and Directory Comparisons tool to compare two MAT-files, a window opens and presents the variables in the two files side by side. The tool enables you to:

- See which variables are common to each file and which are unique.

- Load the contents of the variables into the Variable Editor.

- Load the MAT-files into the workspace.

To compare two MAT-files, follow these steps:

1 Select **Desktop > File and Directory Comparisons**.

The File and Directory Comparisons window opens a dialog box that is empty except for the title bar, menu bar, and a toolbar.
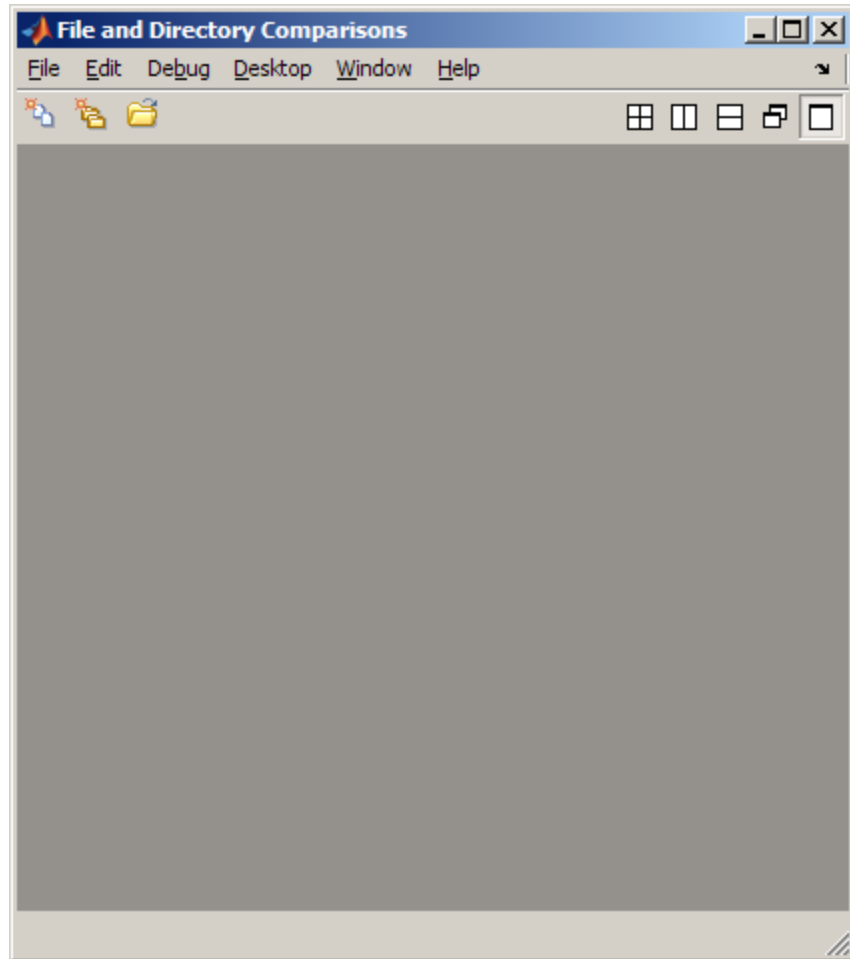
2 Click the New file comparison button:

3 In left side of the File and Directory Comparisons window, click the **Select a File** button , and then browse to and select the name of one of the MAT-files that you want to compare.

4 In right side of the File and Directory Comparisons window, enter the full file name of the other MAT-file that you want to compare.

The File and Directory Comparisons tool displays the file variable names side by side and highlights variables that do not match, as follows:

- Pink highlighting indicates that the values of the variables differ between the two files.

- Green highlighting indicates a variable that exists in the file presented on the right, but not in the file presented on the left.

- Purple highlighting indicates a variable that exists in the file presented on the left, but not in the file presented on the right.

The following image shows the results when you compare *matlabroot*/toolbox/matlab/demos/gatlin2.mat to *matlabroot*/toolbox/matlab/demos/gatlin.mat.
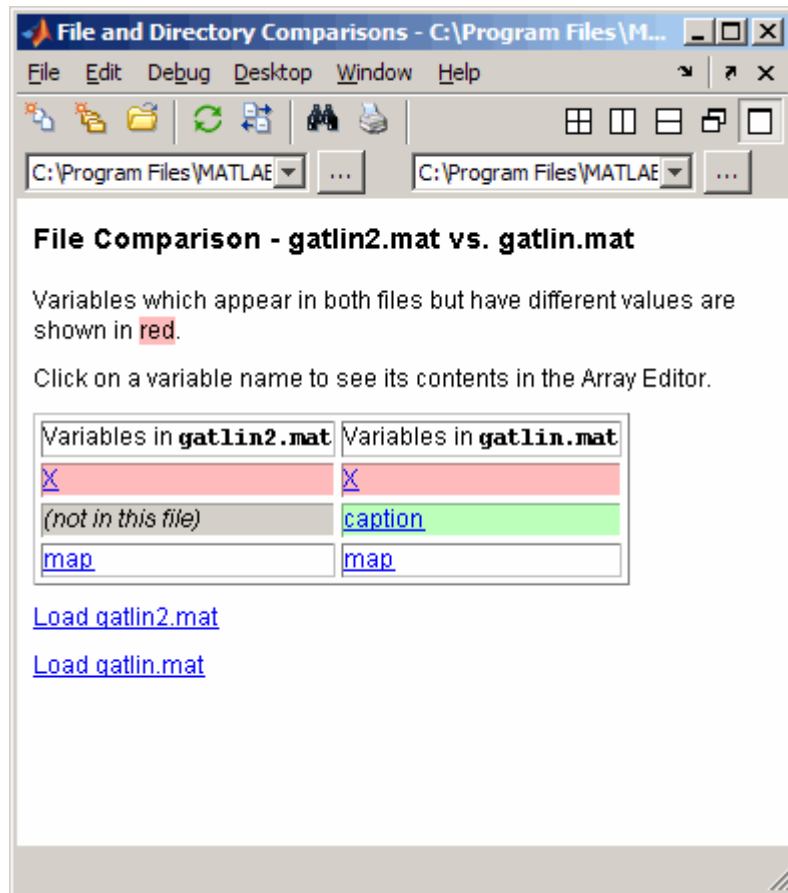
5 Click a variable name to view its contents in the Variable Editor.

6 Click a load link to load the specified file's variables into the workspace.

Comparing Two Binary Files

When you use the File and Directory Comparisons tool to compare two non-MAT-file binary files, such as DLL files or MEX-files, the tool returns a message indicating whether the files are the same.

To compare two binary files, follow the same steps in "Comparing Two MAT-Files" on page 6-82. If the files are the same, the tool displays the message: `The files are` **`identical`**. If the files differ, the tool displays the message: `The files are` **`different`**`. MATLAB cannot display the differences between files of these types.`

Comparing Two Directories

When you use the File and Directory Comparisons tool to compare two directories, a window opens and presents the contents of the directories, side by side. The tool enables you to:

- Determine the files that the directories have in common.

- Determine if files with identical names that are common to both directories also have identical content.

- Open for comparison two files that are common to both directories, but have different content.

- Open for comparison two subdirectories that are common to both directories, but have different content.

- Open a file for viewing in the Editor.

To compare two directories, follow these steps:

1 Select **Desktop > File and Directory Comparisons**.

The File and Directory Comparisons window opens.

2 Select **File > New Directory Comparison** or click the New directory comparison button .

The File and Directory Comparisons window refreshes with a `Type a directory name here` field on each side of the tool.

3 Type or browse to a directory on each side of the tool.

The File and Directory Comparisons tool displays the contents of the directories side by side and highlights files and subdirectories that do not match, as follows:

- Light red highlighting indicates that the contents of the files differ.
- Dark red highlighting indicates that the contents of the subdirectories differ
- Light green highlighting indicates a file that exists in the directory on the right, but not in the directory on the left.
- Dark green highlighting indicates a subdirectory that exists in the directory on the right, but not in the directory on the left.
- Light blue highlighting indicates a file that exists in the directory on the left, but not in the directory on the right.
- Dark blue highlighting indicates a subdirectory that exists in the directory on the left, but not in the directory on the right.

The following image shows an example of the File and Directory Comparisons tool when two directories are compared.
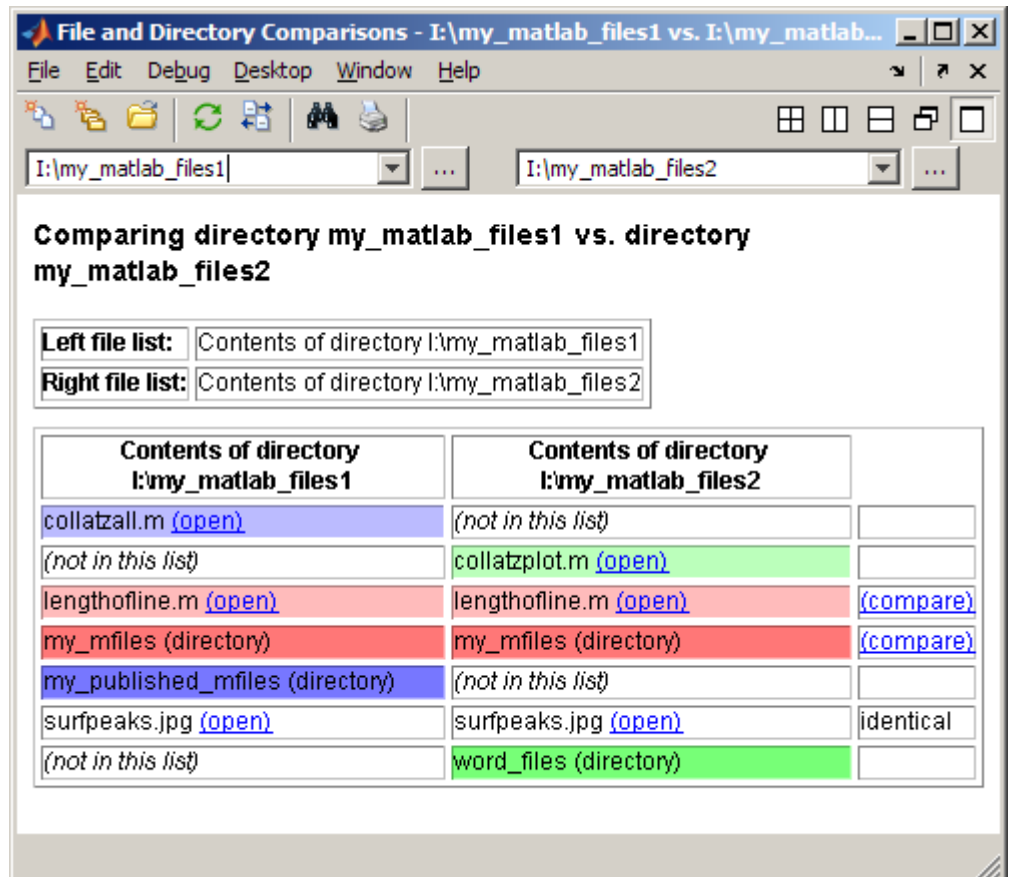
4 Click the open link next to a file name to open that file in the Editor.

5 Click the compare link next to a set of directory names that are highlighted in dark red to refresh the File and Directory Comparisons tool with the two highlighted directories presented for comparison.

6 Click the compare link next to a set of file names that are highlighted in light red to refresh the File and Directory Comparisons tool with the two highlighted files presented for comparison.

Using Features of the File and Directory Comparisons Tool

The File and Directory Comparisons tool provides features that let you do any of the tasks described in the following sections:

- "Navigating from One Difference to the Next" on page 6-89

- "Increasing or Decreasing Line Lengths Shown for Text Files" on page 6-90

- "Exchanging Positions" on page 6-90

- "Showing Updated Files" on page 6-90

- "Finding Text" on page 6-90

- "Viewing a Summary of Differences" on page 6-91

- "Replacing a File or Directory Being Compared with Another File or Directory" on page 6-91

- "Viewing New Comparisons" on page 6-91

- "Viewing Previous Comparisons" on page 6-91

Navigating from One Difference to the Next

Because text files can be lengthy, the File and Directory Comparisons tool provides:

- A link at the top of the tool to the first difference in the file

- Up and down arrows to enable you to navigate quickly from one set of differences to the next

Click the up arrow to navigate to a previous set of lines that differ, and the down arrow to navigate to the next set of lines that differ. When you click the first up arrow in the file, it navigates to the top of the file. Similarly, when you click the last down arrow in the file, it navigates to the bottom of the file.
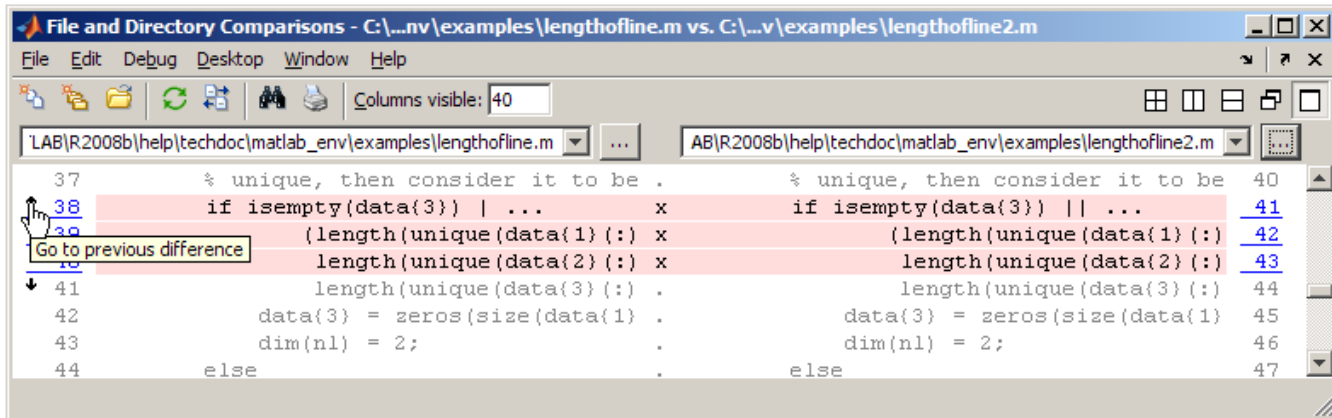
Increasing or Decreasing Line Lengths Shown for Text Files

When comparing text files, the display is 60 columns wide, by default. To increase the display width, type a high number in the **Columns visible** field, and then drag the vertical edges of the window to make it wider. If keeping the window size narrow results in more columns appearing for the file on the left than for the file on the right, reduce the number for **Columns visible** to display a sufficient number of columns for both files, given the window width.

Exchanging Positions

To move the file or directory on the left side to the right side and vice versa, select **File > Swap Sides**, or click the Swap sides button.

Showing Updated Files

After making changes to and saving the files in the Editor, update the results in the File Comparisons tool by selecting **File > Refresh** or clicking the Refresh button.

Finding Text

To find a phrase in the current display, select **Edit > Find**, or click the Find text button. The resulting Find dialog box is the same as the one you use in the Command Window. For more information, see "Finding Text Currently Displayed in the Command Window" on page 3-53.

Viewing a Summary of Differences

To see a summary of difference between two text files, scroll to the bottom of the File and Directory Comparisons tool. There you find a list such as the following:

- Number of matching lines:52

- Number of unmatched lines in left-hand file: 12

- Number of unmatched lines in right-hand file: 15

Replacing a File or Directory Being Compared with Another File or Directory

If the tool is currently comparing files, you can replace an existing file in the tool by doing one of the following, using the file and directory field in the File and Directory Comparisons toolbar :

- Drag a file from Windows Explorer to a file field.

- Type the path to a file in a file field.

- Click the Browse for file button next the a file field , and then select a file.

- Place the cursor in a file field, and then select **File > Open**.

If the tool is currently comparing directories, you can replace an existing directory by typing the path to a directory, or browsing to a directory in the file field.

Viewing New Comparisons

You can perform another file comparison by selecting **File > New File Comparison** or clicking the New file comparison button .

You can perform another directory comparison by selecting **File > New Directory Comparison** or clicking the New directory comparison button .

Viewing Previous Comparisons

You can see the results of previous comparisons in the current session by selecting that comparison's entry in the document bar (as shown at the

bottom of the window in the illustration in "Comparing Two Text Files" on page 6-79). If you close the File and Directory Comparisons tool, the current and previous comparisons are lost.

Alternative Ways to Access the Tool

In addition to the methods shown in the previous sections, you can also access the File and Directory Comparisons tool using one of these methods:

- From the MATLAB desktop, select **Desktop > File and Directory Comparisons**.

- From the Current Directory browser, select a file or directory, right-click, and from the context menu, select **Compare Against**.

- For two files or subdirectories in the same directory, from the Current Directory browser, select the files or directories, right-click, and from the context menu, select **Compare Selected Files** or **Compare Selected Directories**.

Supply the files or directories to compare as described in "Comparing Two Text Files" on page 6-79 and "Comparing Two Directories" on page 6-86, respectively.

Function Alternative for Comparing Files and Directories

Use the visdiff function to open the File and Directory Comparisons tool from the Command Window. For example, type:

```
visdiff('lengthofline.m', 'lengthofline2.m')
```

Keyboard Shortcuts in the Editor

Following is the list of keys that serve as shortcuts for using the Editor, excluding the shortcut keys for menu items. You can view those shortcut keys (sometimes called hot keys) on the menus. See also general desktop "Performing Desktop Tasks Using Keyboard Shortcuts" on page 2-67.

Key or Mouse Action for Microsoft Windows Preference	Key or Mouse Action for Emacs Preference	Key or Mouse Action for Apple Macintosh Preference	Operation
↑	↑ or **Ctrl+P**	↑	Move to previous line.
↓	↓ or **Ctrl+N**	↓	Move to next line.
Ctrl+Home	**Ctrl+Home**	**Cmd+Home**	Move to top of file.
Ctrl+End	**Ctrl+End**	**Cmd+End**	Move to end of file.
Ctrl+↑	**Ctrl+**↑	**Home**	Scroll up one line without moving cursor position (with cell mode disabled). Scroll to top of current cell or top of previous cell, and move the cursor there (with cell mode enabled).

Key or Mouse Action for Microsoft Windows Preference	Key or Mouse Action for Emacs Preference	Key or Mouse Action for Apple Macintosh Preference	Operation
Ctrl+↓	**Ctrl+↓**	**End**	Scroll down one line without moving cursor position (with cell mode disabled). Scroll to top of next cell, and move the cursor there (with cell mode enabled).
Page Down	**Page Down** or **Ctrl+V**	**Page Down**	Move down one screen.
Page Up	**Page Up** or **Alt+V**	**Page Up**	Move up one screen.
←	**←** or **Ctrl+B**	**←**	Move back one character.
→	**→** or **Ctrl+F**	**→**	Move forward one character.
Ctrl+←	**Ctrl+←**	**Option+←**	Move left one word.
Ctrl+→	**Ctrl+→**	**Option+→**	Move right one word.
Home	**Home** or **Ctrl+A**	**Cmd+←**	Move to first nonwhitespace character of line. Press twice to move to first column of line.

Key or Mouse Action for Microsoft Windows Preference	Key or Mouse Action for Emacs Preference	Key or Mouse Action for Apple Macintosh Preference	Operation
End	**End** or **Ctrl+E**	**Cmd+→**	Move to end of line.
Delete	**Delete** or **Ctrl+D**	**Forward Delete**	Delete character after cursor.
Backspace	**Backspace**	**Delete**	Delete character before cursor.
None	**Ctrl+K**	None	Cut contents (kill) to end of line.
Double-click	Double-click	Double-click	Select current word. To select additional words, hold mouse after second click and continue dragging left or right.
Triple-click	Triple-click	Triple-click	Select current line. To select additional lines, hold mouse after second click and continue dragging up or down.
Click middle mouse button over document name in document bar	Click middle mouse button over document name in document bar	None	Close document named in document bar. MATLAB prompts you to save the document if it contains unsaved changes.
Ctrl+Shift+←	**Ctrl+Shift+←**	**Option+Shift+←**	Select word to the left.
Ctrl+Shift+→	**Ctrl+Shift+→**	**Option+Shift+→**	Select word to the right.
Shift+Home	**Shift+Home**	**Cmd+Shift+←**	Select to beginning of line.

Key or Mouse Action for Microsoft Windows Preference	Key or Mouse Action for Emacs Preference	Key or Mouse Action for Apple Macintosh Preference	Operation
Shift+End	**Shift+End**	**Cmd+Shift+→**	Select to end of line.
Shift+Page Up	**Shift+Page Up** or **Ctrl+Shift+V**	**Shift+Page Up**	Select one screen up.
Shift+Page Down	**Shift+Page Down** or **Alt+Shift+V**	**Shift+Page Down**	Select one screen down.
Ctrl+Shift+Home	**Ctrl+Shift+Home**	**Cmd+Shift+Home**	Select to top of file.
Ctrl+Shift+End	**Ctrl+Shift+End**	**Cmd+Shift+End**	Select to end of file.
Shift+Enter	**Shift+Enter**	**Shift+Enter**	Add a new line that is not indented.
Insert	**Insert**	None	Toggle between insert mode and overwrite mode. In insert mode: • The status bar displays OVR as dimmed text. • The cursor is a vertical bar. In overwrite mode: • The status bar displays OVR as undimmed text. • The cursor is a black block (box).

Key or Mouse Action for Microsoft Windows Preference	Key or Mouse Action for Emacs Preference	Key or Mouse Action for Apple Macintosh Preference	Operation
			The Insert key is not supported on Macintosh platforms.
Shift+F5	**Shift+F5**	**Shift+F5**	Exit debug mode. Equivalent to typing dbquit. The Command Window displays the
Esc	**Esc**	None	If the function hints pop-up window is open, closes the window. In the Function Browser, press **Esc** up to three times: first to dismiss the search history, then to clear the search field, and lastly, to close the Function Browser.

Saving, Printing, and Closing Files in the Editor

In this section...

Saving M-Files

After making changes to a file, an asterisk (*) follows the file name in the title bar of the Editor. This asterisk indicates there are unsaved changes to the file.

To save the changes, use one of the **Save** commands in the **File** menu:

- **Save** — Saves the file using its existing name. If the file is newly created, the **Save file as** dialog box opens, where you assign a name to the file before saving it. Another way to save is by clicking the Save button ⊟ on the toolbar. If the file has not been changed, **Save** appears dimmed, but you can instead use **Save As** from the **File** menu to save to a different file name.

- **Save As** — The **Save file as** dialog box opens, where you assign a name to the file and save it. By default, if you do not type an extension, MATLAB software automatically assigns the `.m` extension to the file name. If you do not want an extension, type a `.` (period) after the file name.

- **Save All** — Saves all open files to their existing file names. For all newly created files, the **Save file as** dialog box opens, where you assign a name to each untitled file and save it. Another way to save all open files is by clicking the Save All button ⊟. This button is not on the toolbar by default, however. For information on adding it, see "Setting Toolbars Preferences for Desktop Tools" on page 2-138.

You cannot save an M-file while in debug mode. If you try to, MATLAB desktop displays a dialog box asking if you want to exit debug mode and then save the file. While debugging, you can execute sections of an M-file even though there are unsaved changes—see "Running Sections in M-Files That Have Unsaved Changes" on page 6-177.

Recommendations on Saving M-Files

The MathWorks™ recommends that you save M-files you create and M-files from The MathWorks that you edit to a directory that is not in the *matlabroot*/toolbox directory tree. If you keep your files in *matlabroot*/toolbox directories, they can be overwritten when you install a new version of MATLAB software.

Be aware that locations of files in the *matlabroot*/toolbox directory tree are loaded and cached in memory at the beginning of each MATLAB session to improve performance. Therefore, if you save files to *matlabroot*/toolbox directories using an external editor, or add or remove files from these directories using file system operations, run rehash toolbox before you use the files in the current session. If you make changes to existing files in *matlabroot*/toolbox directories using an external editor, run clear functionname before you use these files in the current session. For more information, see rehash or "Toolbox Path Caching in the MATLAB Program" on page 1-22.

Autosave

As you make changes to a file in the Editor, every 5 minutes the Editor automatically saves a copy of the file to a file of the same name but with an .asv extension. The autosave copy is useful if you have system problems and lose changes made to your file. In that event, you can open the autosave version, filename.asv, and then save it as filename.m to use the last good version of filename. For example, if you edit filename.m and do not save it for five minutes, MATLAB saves the file including the unsaved changes, to filename.asv.

Use autosave preferences to turn the autosave feature off or on, to specify the number of minutes between automatic saves, and to specify the file extension and location for autosave files. For details, select **File > Preferences > Editor/Debugger > Autosave**, and then click **Help**.

If the file you are editing is in a read-only directory and the autosave preference for location is the source file directory, an autosave copy of the file is *not* made.

Deleting Autosave Files. By default, autosave files are not automatically deleted when you delete the source file. To keep autosave to M-file relationships clear and current, it is a good practice when you rename or remove an M-file to delete or rename its corresponding autosave file.

There is a preference to **Automatically delete autosave files**. With this preference selected, when you close an M-file in the Editor, MATLAB automatically deletes the corresponding autosave file.

Accessing Your Source Control System

If you use a source control system for M-files, you can access it from within the Editor using **File > Source Control**. For more information, see Chapter 10, "Source Control Interface".

Printing M-Files

To print an entire M-file, select **File > Print**, or click the Print button 🖨 on the toolbar. To print the current selection, select **File > Print Selection**. Complete the standard print dialog box that appears.

Specify printing options for the Editor by selecting **File > Page Setup**. For example, you can specify printing with a header. For more information, see "Printing and Page Setup Options for Desktop Tools" on page 2-89.

Closing M-Files

To close the current M-file, select **Close *file name*** from the **File** menu, or click the Close box in the Editor menu bar. This is different from the Close box in the titlebar of the Editor, which closes all open files in that Editor window.

Close box for
Editor/Debugger.

Close box for
current file.

Closes all open files in this
Editor/Debugger window.
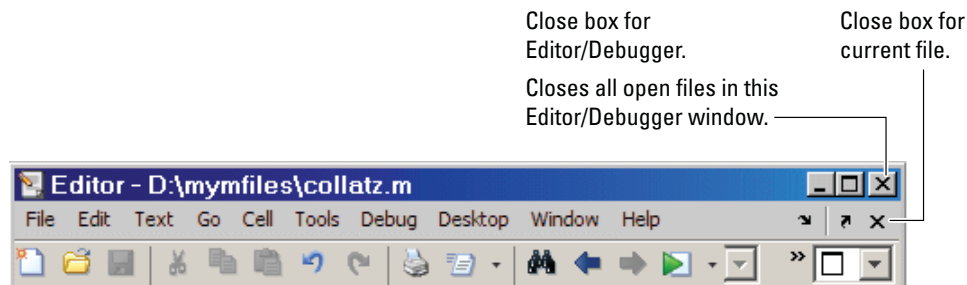
To close all files within the Editor, select **Window > Close Editor Documents**. This does not close any files undocked from the Editor. The Editor remains open with no files in it.

If each file is open in a separate window, close all the files at once using the **Close All Documents** item in the **Window** menu. Note that this also closes desktop documents of all types, including Variable Editor documents.

When you close a file that has unsaved changes, you are prompted to save the file. If you do not want to be prompted, hold **Ctrl** and click the Close box. The prompt will not appear and the document will close without saving any unsaved changes.

Running M-Files in the Editor

Running M-Files with No Input Arguments in the Editor

In the Editor, to run a script M-file, or a function M-file that requires no input arguments, click the Run button ▶ on the toolbar. The button's ToolTip includes the name of the file to be run, which is useful when you have multiple files open. Alternatively, select **Debug > Run *file name***.

If the file is neither in a directory on the search path nor in the current directory, a dialog box appears with options that allow you to run the file. You can either change the current directory to the directory containing the file, or you can add the directory containing the file to the search path.

If the file has unsaved changes, running it from the Editor automatically saves the changes before running. In that event, the **Debug** menu item is **Save File and Run *file name***.

If the M-file is a script, you can view the value of a variable in the file, which is called a *data tip* (like a ToolTip for data). You need to set the preference to show data tips in edit mode—select

File > Preferences > Editor/Debugger > Display, and for **General Display Options**, select the check box for **Enable datatips in edit mode**.

Using Run Configurations to Run M-Files with Input Arguments in the Editor

In the Editor, you can provide values for a function's input arguments using a run configuration, and then run that configuration to use the assigned values. When you are editing a function M-file, use a run configuration as an alternative to running the function in the MATLAB Command Window. You can associate multiple run configurations with an M-file to assign different input values. MATLAB saves the run configurations between sessions to a file named run_configurations.m. (For details, see "About the run_configurations.m File" on page 6-112.)

Consider the function collatzplot_new.m, which computes and plots the Collatz sequence for any given positive integer. This function requires you to specify the integer as an input value. You cannot simply run collatplot_new.m in the Editor because the input value is not defined. One way to specify the input value is to run the M-file in the Command Window. Run configurations allow you to run collatzplot_new(*specific value*) in the Editor.

You can also use run configurations to provide preparatory or setup information before running an M-file, whether it takes input arguments or not.

Note M-File run configurations use the base MATLAB workspace. Therefore, a value that you assign to a variable in an M-file run configuration overwrites the value for that variable (assuming it currently exists) in the base workspace.

Create and Use a Run Configuration for an M-File

Follow these steps to create and use a run configuration for an M-file in the Editor. These steps specify Editor toolbar buttons, but you can also use equivalent options in the **Debug** menu.

1 Open the file you want to run in the Editor. For example, open `collatzplot_new.m` by running the following command:

```
edit(fullfile(matlabroot,'help','techdoc',...
'matlab_env','examples','collatzplot_new.m'))
```

To work with `collatzplot_new.m` on your system, save the file to a directory for which you have write permission. In the example, the file is saved to `I:\my_matlab_files\my_mfiles\collatzplot_new.m`.

2 Click the down arrow on the Run button in the Editor toolbar,
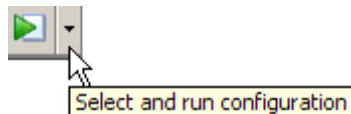
and then select **Edit Run Configurations for *file-name***, where *file-name* in this example is `collatzplot_new.m`.
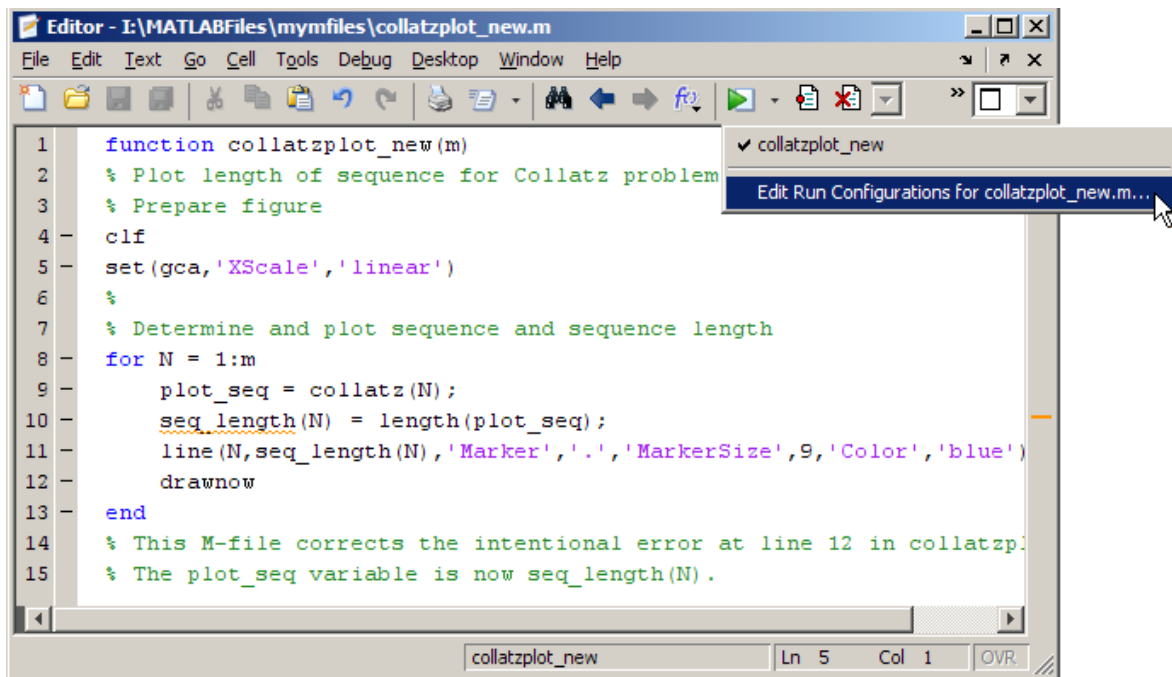
The Edit M-File Configurations dialog box opens, with a default run configuration template for collatzplot_new.m.
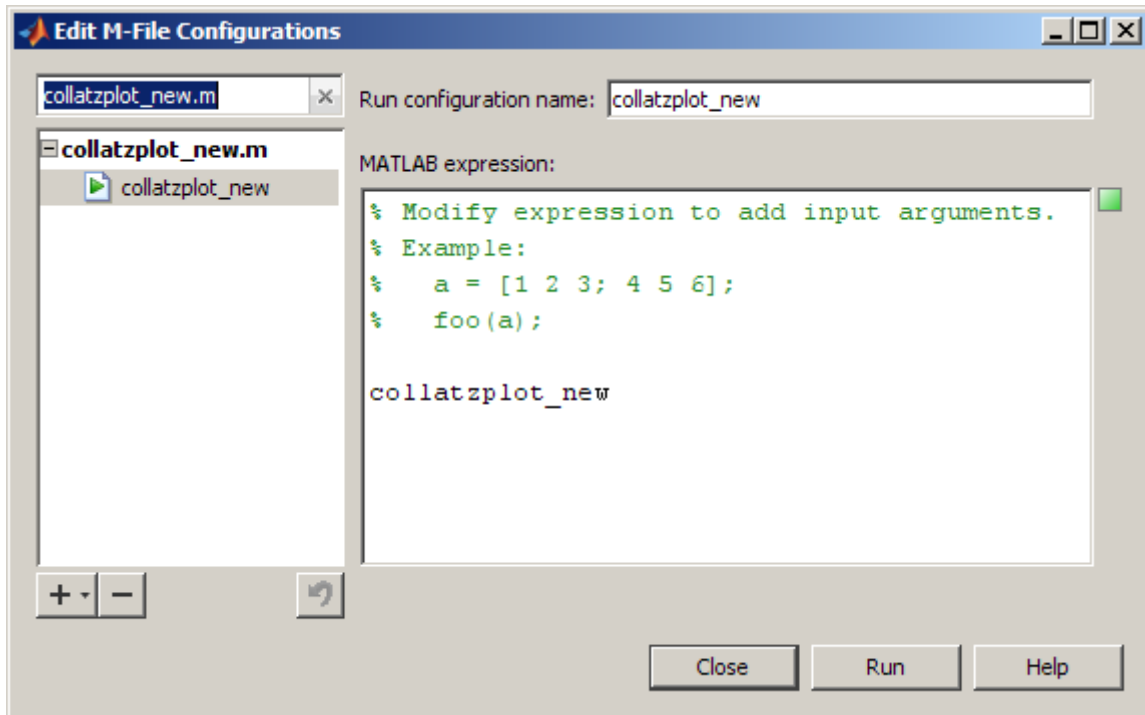
3 In the **MATLAB expression** area of the dialog box, enter MATLAB statements that you want to run. Delete the existing comments or replace them with comments relevant to your run configuration. To undo and redo, use the keyboard shortcuts for your platform, such as **Ctrl+Z** and **Ctrl+Y** for Microsoft Windows platforms.

In this example, set m equal to 3, which is a small value useful for debugging purposes. Complete the statement to run collatzplot_new(m).

```
MATLAB expression:

% For debugging purposes
m=3;
collatzplot_new(m)
```

The **MATLAB expression** area provides syntax highlighting and shows M-Lint messages, similar to the Editor.

4 To ensure your run configuration executes as expected, click **Run** to execute the statements in the **MATLAB expression** field. In this example, collatzplot_new(3) runs, and a Figure window displays the plot.
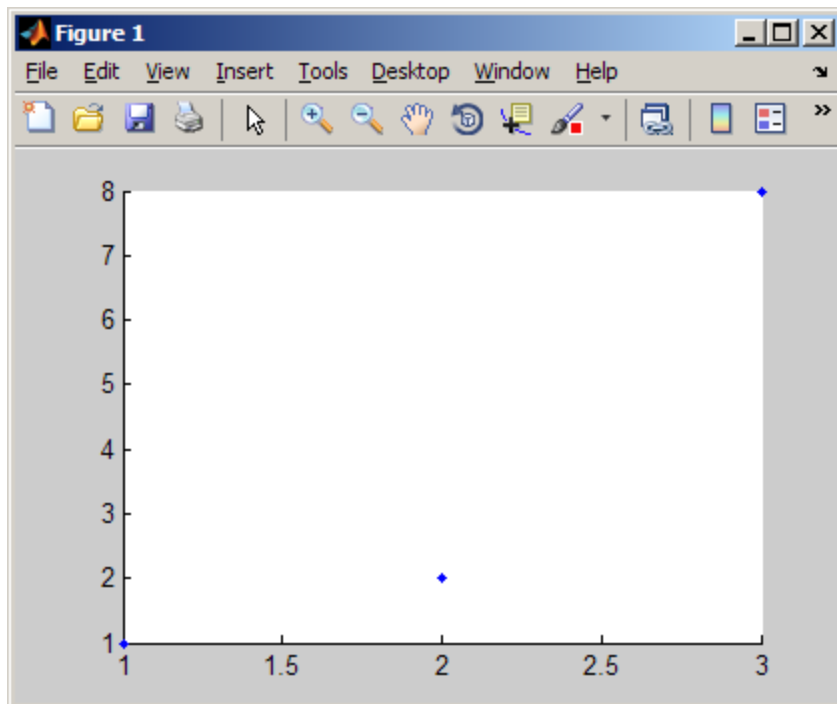
5 You can modify the statements in the **MATLAB expression** area of the dialog box and click **Run** to see the results of the changes. You can also modify the M-file and save the changes while the Edit M-File Configurations dialog box is open, and then click **Run** to see the results of the M-file changes.

6 You can assign a name using the **Run configuration name** field in the Edit M-File Configurations dialog box. By default, the run configuration name is the M-file name. If you expect to create multiple run configurations for an M-file, assign each a name that helps you identify the configuration. In this example, name the run configuration `collatzplot_new_test`.

MATLAB automatically saves the run configuration and its association with the M-file in the `run_configurations.m` file in your preferences directory.

For more information, see "About the run_configurations.m File" on page 6-112.

7 To close the Edit M-File Configurations dialog box, click **Close**.

8 After creating a run configuration, you can view and use the configuration without opening the Edit M-File Configurations dialog box.

In the Editor toolbar, click the down arrow on the Run button and position the mouse pointer on a run configuration name. The MATLAB desktop displays a ToolTip showing the run configuration's **MATLAB Expression** so you can see what will run.
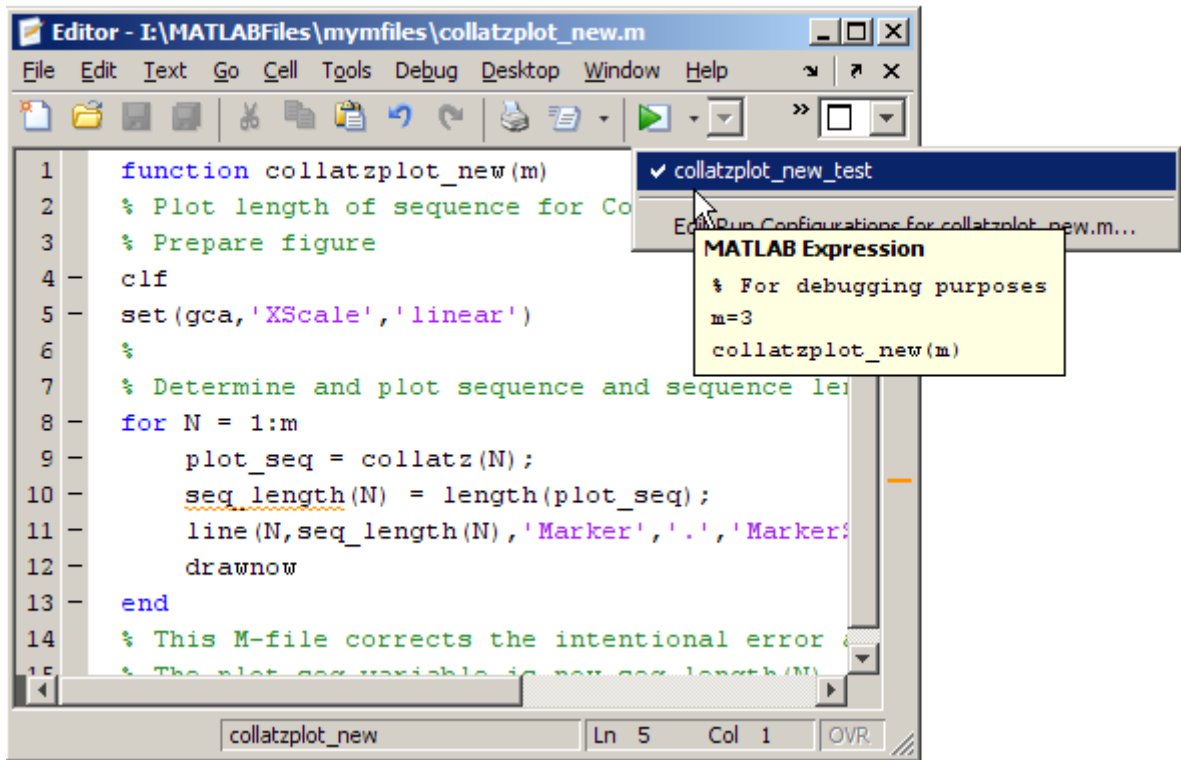
9 To use the run configuration, select the run configuration name. MATLAB runs the expression you specified in the run configuration. For example, select `collatzplot_new_test`, and MATLAB runs `collatzplot_new(3)`, as specified in step 3. You can modify the M-file, save it, and execute the run configuration from the toolbar to see the effects of the M-file changes.

Create and Execute Multiple Run Configurations for an M-File

You can create multiple run configurations for a given M-file, allowing you to run with different values for input arguments, each for a different purpose. Create a named run configuration for each purpose, all associated with the M-file. Then any time you open the M-file, choose and execute the run configuration you want. For example, for `collatzplot_new(m)` you might use three values for `m` and have three run configurations:

- Small value, for example, 3, for debugging and testing

- Realistic value, for example, 200 or more, for a specific project

- Random value to observe changes

1 Open the Edit M-File Configurations dialog box, and then do the following:

    a Select the M-file to which you want to add a run configuration, or select a configuration associated with that M-file.

    b Click the Add button **+** (under the list of M-files and configurations), and then click **Run Configuration**.

MATLAB creates a new default run configuration template, in this example, `collatzplot_new`.

The example shows `collatzplot_new` and its default expression, as well as one previously created run configuration associated with `collatzplot_new.m`, `collatzplot_new_test`.
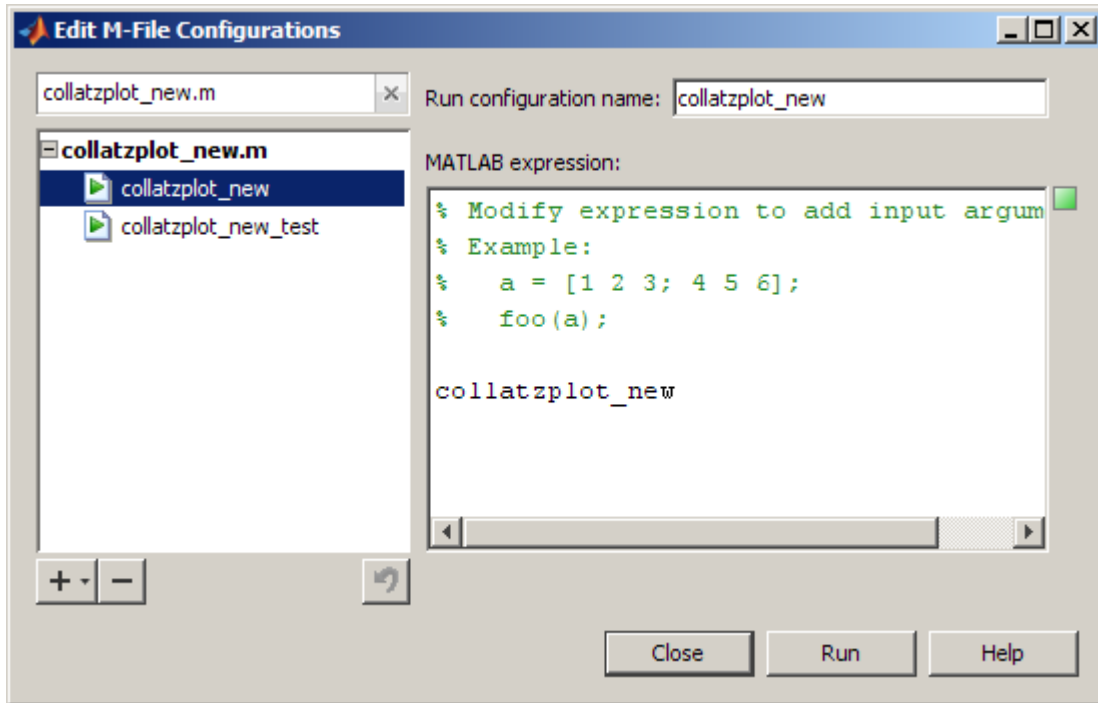
2 In the Edit M-File Configurations dialog box, modify, run, and name the new run configurations as you did for the initial run configuration, collatplot_new_test, as described in "Create and Use a Run Configuration for an M-File" on page 6-103.

For example, rename collatzplot_new to collatzplot_new_largevalue, and replace the default template expression with:

```
m=200;
collatzplot_new(m)
```

To create another run configuration, click the down arrow next to the Add button ✚▾ again, and then click **Run Configuration**. Rename collatzplot_new_2 to collatzplot_new_random and replace the default template expression with:

```
% Random value
```

```
m=int16(rand*50);
collatzplot_new(m)
clear all
```

3 Select a run configuration in the listing to see and modify its expression, or to rename the configuration. Click the expanders next to an M-file name (plus **+** and minus **-** signs on Windows platforms) to see or hide all the configurations associated with that M-file.

4 To get a quick view of the expression in a configuration, position the mouse pointer on the name of a configuration without selecting it. In this example, collatzplot_new_largevalue is selected and you can edit its expression or name. The pointer is positioned on collatzplot_new_test and you can see the statements in it.
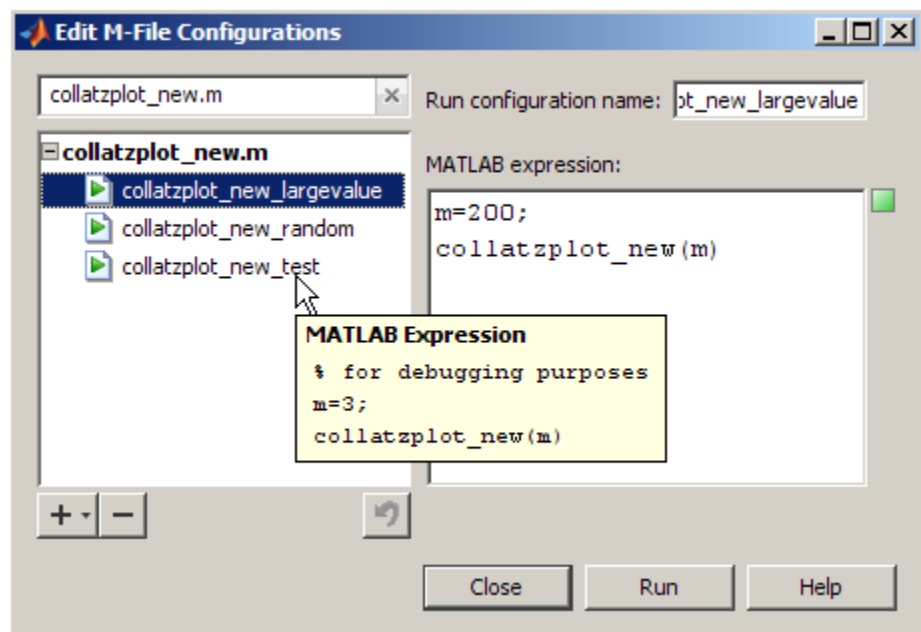
5 To close the Edit M-File Configurations dialog box, click **Close**. MATLAB saves the configurations and their associations with the M-file in the run_configurations.m file in your preferences directory.

6-111

For more information, see "About the run_configurations.m File" on page 6-112.

About the run_configurations.m File

When you create one or more run configurations using the Edit M-File Configurations dialog box, the Editor creates or updates the `run_configurations.m` file in your preferences directory (the directory MATLAB returns when you run `prefdir`). This is a text file that you can view and use to evaluate M-files.

Although you can port this file from the preferences directory on one system to another, there can only be one `run_configurations.m` file on a system. Therefore, you should only do this if you have not already created configurations on the second system. In addition, because this file may contain references to file paths, you need to be sure the specified M-files and paths exist on the second system.

The MathWorks recommends that you do not update this file in the Editor or a text editor. Changes you make using tools other than the Edit M-File Configurations dialog box may be overwritten.

Each time you change a run configuration using the Edit M-File Configurations dialog box, MATLAB updates the `run_configurations.m` file as well as the `publish_configurations.m` file. See "About the publish_configurations.m File" on page 8-108 for more information about that file.

Find Configurations

Follow these steps to find run or publish configurations. (For information on publish configurations, see "Producing Published Output from M-Files" on page 8-68.)

1 Open any M-file in the Editor. For example, open the MATLAB function `sin`.

2 Open the Edit M-File Configurations dialog box. MATLAB automatically creates a default configuration for `sin.m`, if none exists.

In the left pane, MATLAB lists all configurations currently defined for `sin.m`.

3 Click the X icon to clear the filter field.

| sin.m | × |

4 In the left pane, MATLAB lists all M-files containing configurations.
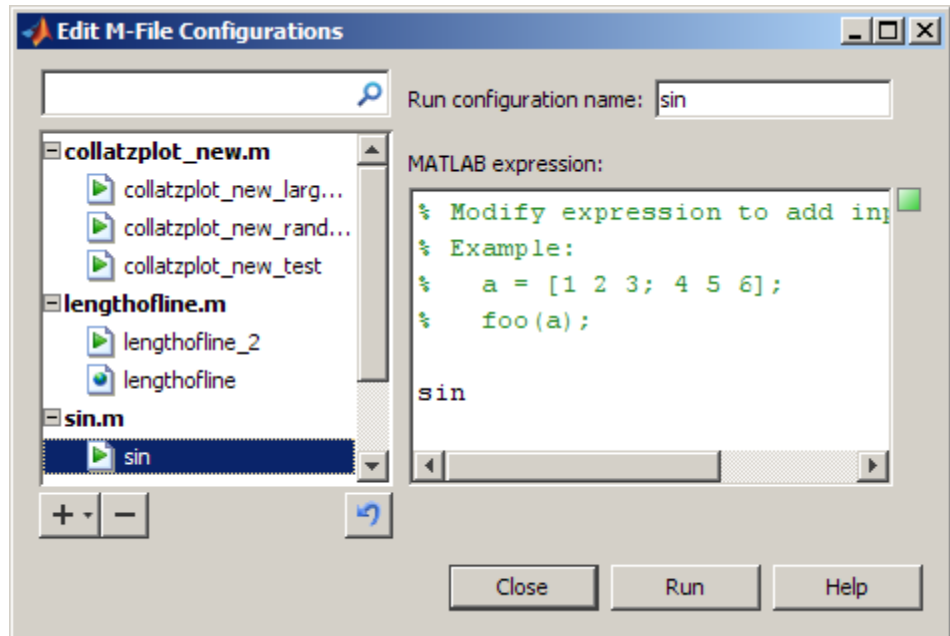
5 Type a term in the filter field to find an M-file or configuration by name.

MATLAB displays only those M-files whose names contain the term, or whose associated configurations contain the term in their name. As you type, MATLAB filters out files and configurations that do not contain the term.

For example, type `rand`. In this example, only one M-file, `collatzplot_new.m`, has a configuration that contains the term `rand`.
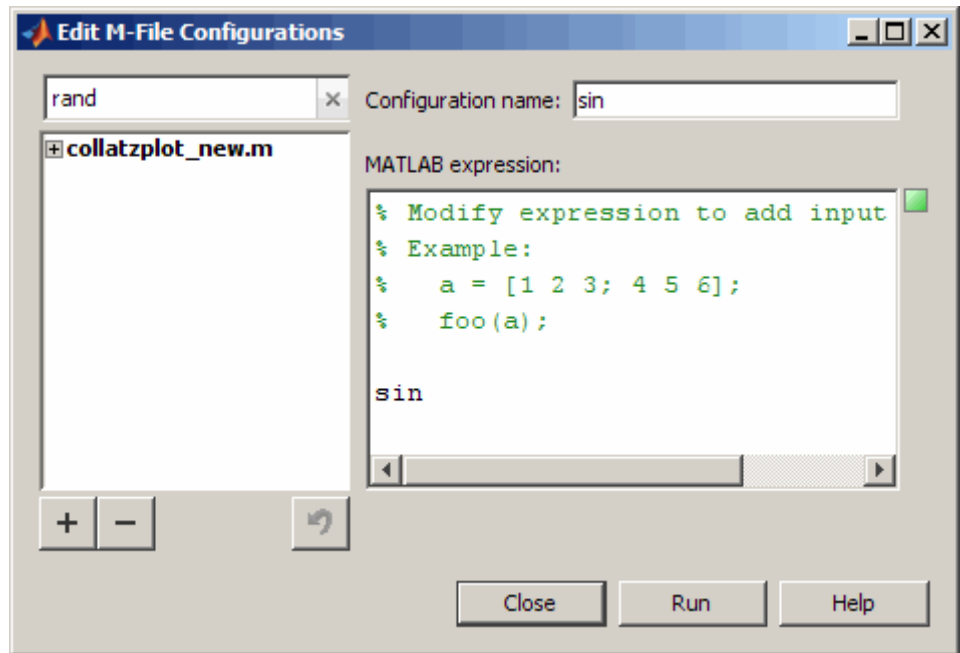
6 If you cannot view the entire name of a configuration, drag the separator bar to the right of the list, making the left pane wider.

7 To see the expression in that configuration, select the configuration, or position the mouse pointer over the name.

8 As you type additional letters in the filter field, fewer M-files remain in the list of results. Use the backspace key to modify the term. If there are no M-files or configurations containing the term, the list is empty.

Remove Configurations

If you no longer need a run or publish configuration because you do not use it or because you deleted the M-file with which it is associated, it is a good practice to delete the configuration. (For information on publish configurations, see "Producing Published Output from M-Files" on page 8-68.)

1 Open any M-file in the Editor.

2 Open the Edit M-File Configurations dialog box.

3 Do one of the following in the panel on the left:

- If you want to remove a single configuration, select that configuration.

- If you want to remove all the run and publish configurations for an M-file, select the M-file

4 Click the Remove button ▬.

5 To undo the last deletion, click the Undo button ↺. You cannot undo the last deletion after you close this dialog box.

Reassociate and Rename Configurations

Each run and publish configuration is associated with a specific M-file. If you move or rename an M-file that has configurations, you need to redefine the association. If you delete an M-file, you might want to delete the associated configurations, or associate them with a different M-file. You might also need to modify the statements in the configurations so they will run.

When MATLAB cannot associate a configuration with an M-file, the Edit M-File Configurations dialog box displays the M-file name in red, displays a **File Not Found** message, and allows you to find the M-file to which you want to associate the configuration. In this example, MATLAB cannot find the file collatzplot_new.m, which has three configurations associated with it. For this example, collatzplot_new.m had been renamed to collatzplot_fixed.m, so the configurations associated with collatzplot_new.m need to be reassociated with collatzplot_fixed.m.
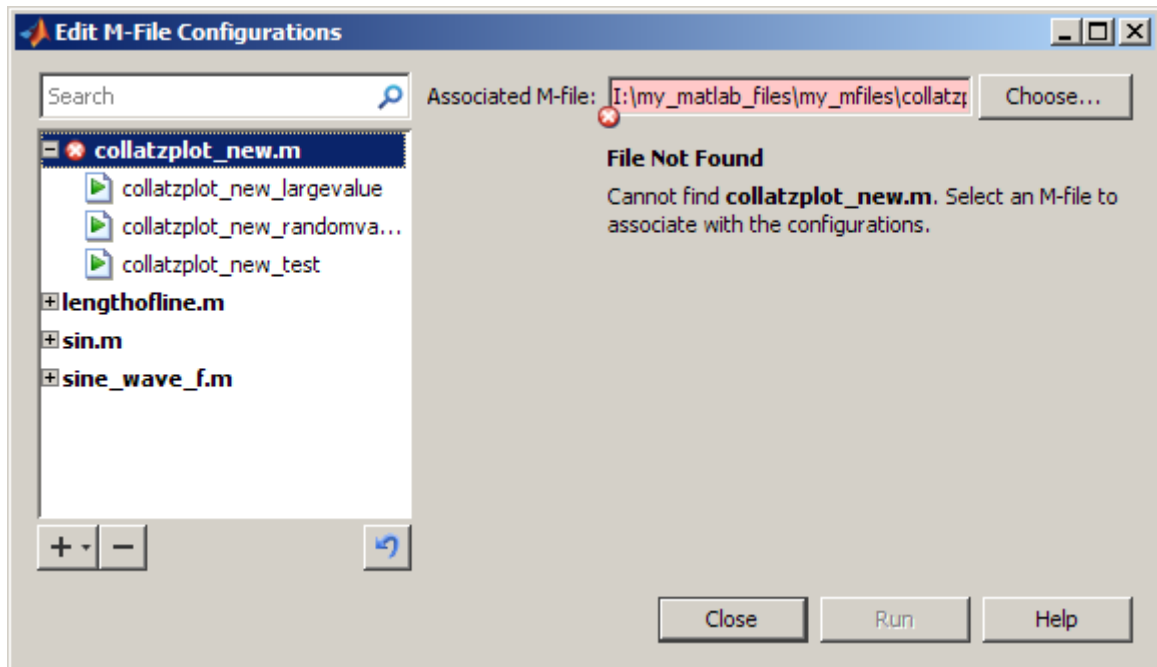
To reassociate a configuration:

1 In the list of configurations (left pane), select the M-file. The **Associated M-file** displays the full path to the M-file that was associated with the configurations. Click **Choose**.

2 In the resulting Open dialog box, navigate to and select the M-file with which you now want to reassociate the configurations. Click **Open**.

In this example, you want to reassociate the configurations with `collatzplot_fixed.m`; select `collatzplot_fixed.m`, and then click **Open**.

In the Edit M-File Configurations dialog box, the **Associated M-file** value reflects the change you made and the **File Not Found** message no longer appears.
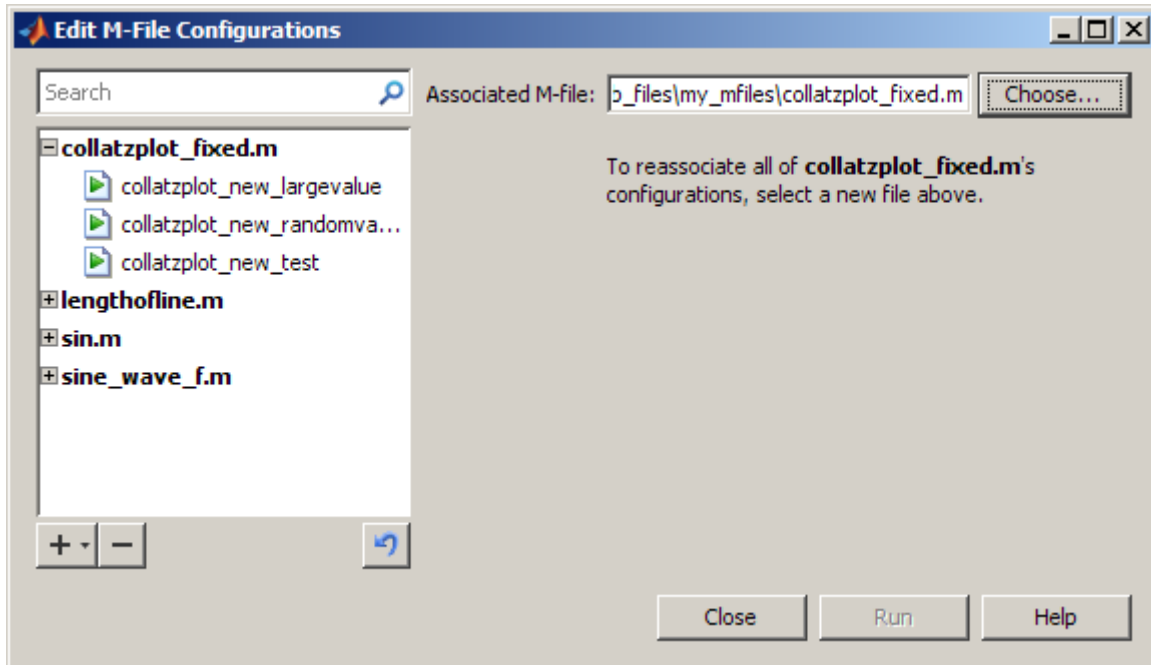
3 You might want to rename the configurations to be consistent with the new M-file name, or at least to not reflect the former M-file name. This is not required, but it is a good practice. To do so, select a configuration from the list in the left pane. In the right pane, edit the value for the configuration name. Depending on the type of configuration that you are renaming, the field is labeled either **Run configuration name** or **Publish configuration name**. Repeat this step for all run and publish configurations associated with the M-file.

In this example, remove `collatzplot_new` from the start of each run configuration name.
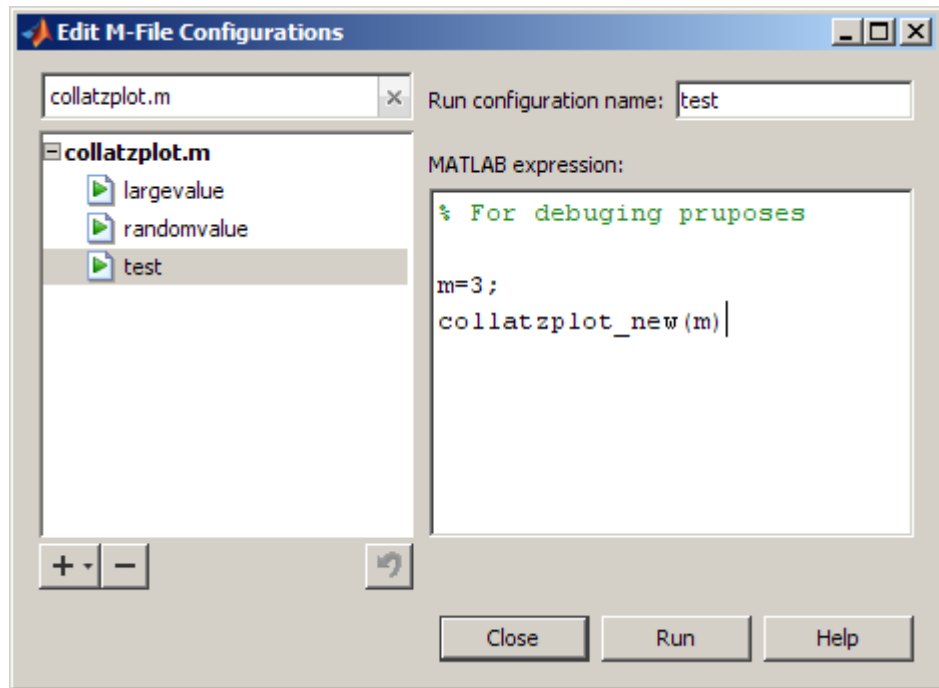
4 For an M-file name change, you might need to modify the configuration statements to run correctly. For this example, modify the collatzplot_new(m) statement in each configuration to use collatzplot(m).
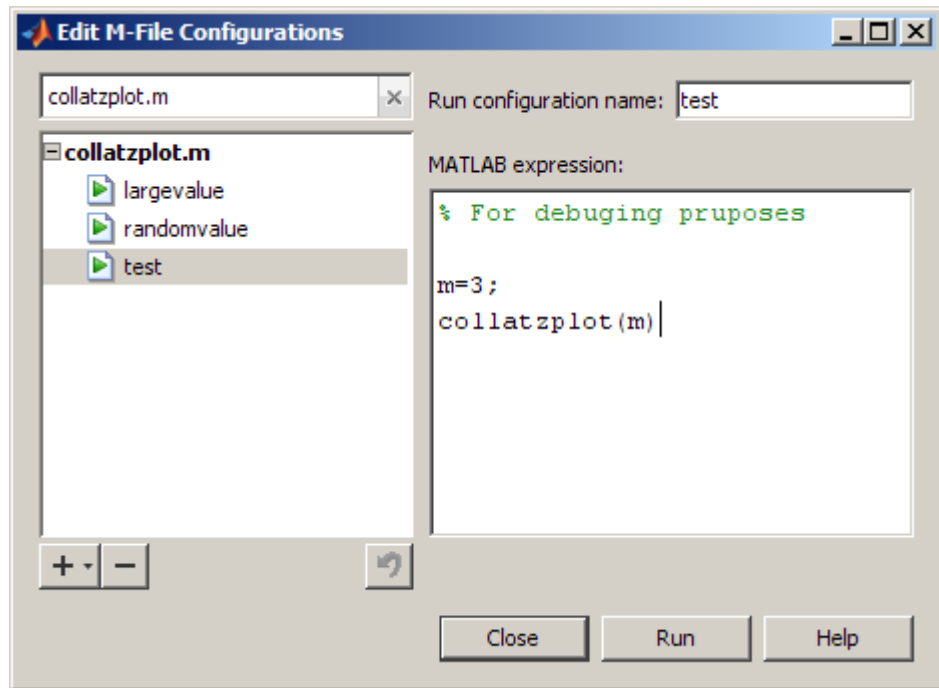
Other Ways to Run M-Files from the Editor

- See "Running an M-File with Breakpoints" on page 6-162 for additional information about running M-files while debugging.

- While debugging, you can execute sections of an M-file even though there are changes. See "Running Sections in M-Files That Have Unsaved Changes" on page 6-177.

- You can execute M-files one section at a time and quickly modify values incrementally using the toolbar. For more information, see "Using Cells for Rapid Code Iteration and Publishing Results" on page 6-185.

Finding Errors, Debugging, and Correcting M-Files

This section introduces general techniques for finding errors and using the M-Lint automatic code analyzer to detect possible areas for improvement in M-files. It then illustrates the MATLAB debugger features in the Editor, as well as equivalent Command Window debugging functions, using a simple example.

There are two kinds of errors:

- Syntax errors — For example, misspelling a function name or omitting a parenthesis.

- Run-time errors — These errors are usually algorithmic in nature. For example, you might modify the wrong variable or code a calculation incorrectly. Run-time errors are usually apparent when an M-file produces unexpected results. Run-time errors are difficult to track down because the function's local workspace is lost when the error forces a return to the MATLAB base workspace. The process of isolating and fixing these run-time problems is referred to as *debugging*.

In addition to finding and fixing problems with your M-files, you might want to improve the performance and make other enhancements using MATLAB tools.

Use the following techniques to isolate the causes of errors and improve your M-files.

Technique or Tool	Description	For More Information
Syntax highlighting and Delimiter matching	Syntax highlighting helps you identify unterminated strings in an M-file before you run the file. Delimiter matching helps you correctly match pairs of parentheses, brackets, braces, and keywords.	"Syntax Highlighting" on page 6-50 "Matching Delimiters (Parentheses)" on page 3-25

Technique or Tool	Description	For More Information
Error Messages	When you run an M-file with a syntax error, MATLAB software will most likely detect it and display an error message in the Command Window describing the error and showing its line number in the M-file. Click the underlined portion of the error message, or position the cursor within the message and press **Ctrl+Enter**. The offending M-file opens in the Editor, scrolled to the line containing the error. To check for syntax errors in an M-file without running the M-file, use the `pcode` function.	None
M-Lint	Use the M-Lint code analyzer to help you verify the integrity of your code and learn about potential improvements. Access M-Lint messages automatically while you work in a file in the Editor, or run an M-Lint report for an existing file. To evaluate the McCabe complexity (also known as the cyclomatic complexity) of an M-File, use the `mlint` function with the `-cyc` option.	"Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 6-123 and the reference page for the `mlint` function
Editor, Graphical Debugger, and MATLAB Debugging Functions	The MATLAB Editor, graphical debugger, and MATLAB debugging functions are useful for correcting run-time problems because you can access function workspaces and examine or change the values they contain. You can set and clear *breakpoints*, indicators that temporarily halt execution in an M-file. While stopped at a breakpoint, you can change workspace contexts, view the function call stack, and execute the lines in an M-file one by one.	"Debugging Process and Features" on page 6-154

Technique or Tool	Description	For More Information
Other Debugging Techniques	• Add `keyboard` statements to the M-file—`keyboard` statements stop M-file execution at the point where they appear and allow you to examine and change the function's local workspace. This mode is indicated by a special `K>>`prompt. Resume function execution by typing `return` and pressing the **Enter** key. For more information, see the `keyboard` reference page. • Remove selected semicolons from the statements in your M-file—semicolons disable the display of output in the M-file. By removing the semicolons, you instruct MATLAB to display these results on your screen as the M-file executes. • List dependent functions—use the `depfun` function to see the dependent functions.	Reference pages for `keyboard` and `depfun` function
Cells	In the Editor, isolate sections of an M-file, called cells, so you can easily make changes to and run a single section.	"Using Cells for Rapid Code Iteration and Publishing Results" on page 6-185
Profiler	Use the Profiler to help you improve performance and detect problems in your M-files. Access the Profiler from the Editor by selecting **Tools > Open Profiler**.	"Profiling for Improving Performance" on page 7-33
Reports	The M-File reports helps you polish and package M-files before providing them to others to use. Access all of the reports from the Current Directory browser.	"Using M-File Reports" on page 7-2

Checking M-File Code for Problems Using the M-Lint Code Analyzer

In this section...
"What Is the M-Lint Code Analyzer?" on page 6-123
"Ways to Use the M-Lint Code Analyzer" on page 6-123
"Using M-Lint Automatic Code Analyzer in the Editor" on page 6-124
"Suppressing M-Lint Indicators and Messages" on page 6-136
"Setting Preferences for M-Lint" on page 6-145

What Is the M-Lint Code Analyzer?

The M-Lint code analyzer, also referred to as just M-Lint, checks your code for problems and recommends modifications to maximize performance and maintainability.

Ways to Use the M-Lint Code Analyzer

You can use M-Lint in three different ways, all of which report the same messages:

- Continuously check code in the Editor while you work.

  View M-Lint messages and make changes to your file based on the messages. The code analyzer updates automatically and continuously so you can see if your changes addressed the issues noted in the M-Lint messages. For some messages, M-Lint offers extended messages, automatic code correction, or both. Details about using the continuous checking and correction interface in the Editor is explained in "Using M-Lint Automatic Code Analyzer in the Editor" on page 6-124

- Run a report for an existing M-file:

  1 From an M-file in the Editor, select **Tools > M-Lint > Show M-Lint Report**.

  2 Make any changes to your file based on the M-Lint messages in the report.

3 Save the file.

4 Rerun the report to see if your changes addressed the issues noted in the M-Lint messages.

- Run a report for all files in a directory:

  1 In the Current Directory browser, click the **Actions** button ⚙.

  2 Select **Reports > M-Lint Code Check Report**.

  3 Make any changes to your files based on the M-Lint messages in the report.

    For details, see "M-Lint Code Check Report" on page 7-21.

  4 Save the file.

  5 Rerun the report to see if your changes addressed the issues noted in the M-Lint messages.

Using M-Lint Automatic Code Analyzer in the Editor

To use the M-Lint continuous code checking in an M-file in the Editor, follow these steps:

1 Ensure the M-Lint messaging preference is enabled: Select **File > Preferences > M-Lint**, and then select the **Enable integrated M-Lint warning and error messages** check box. To follow these instructions, be sure the **Underlining** option is set to Underline warnings and errors.
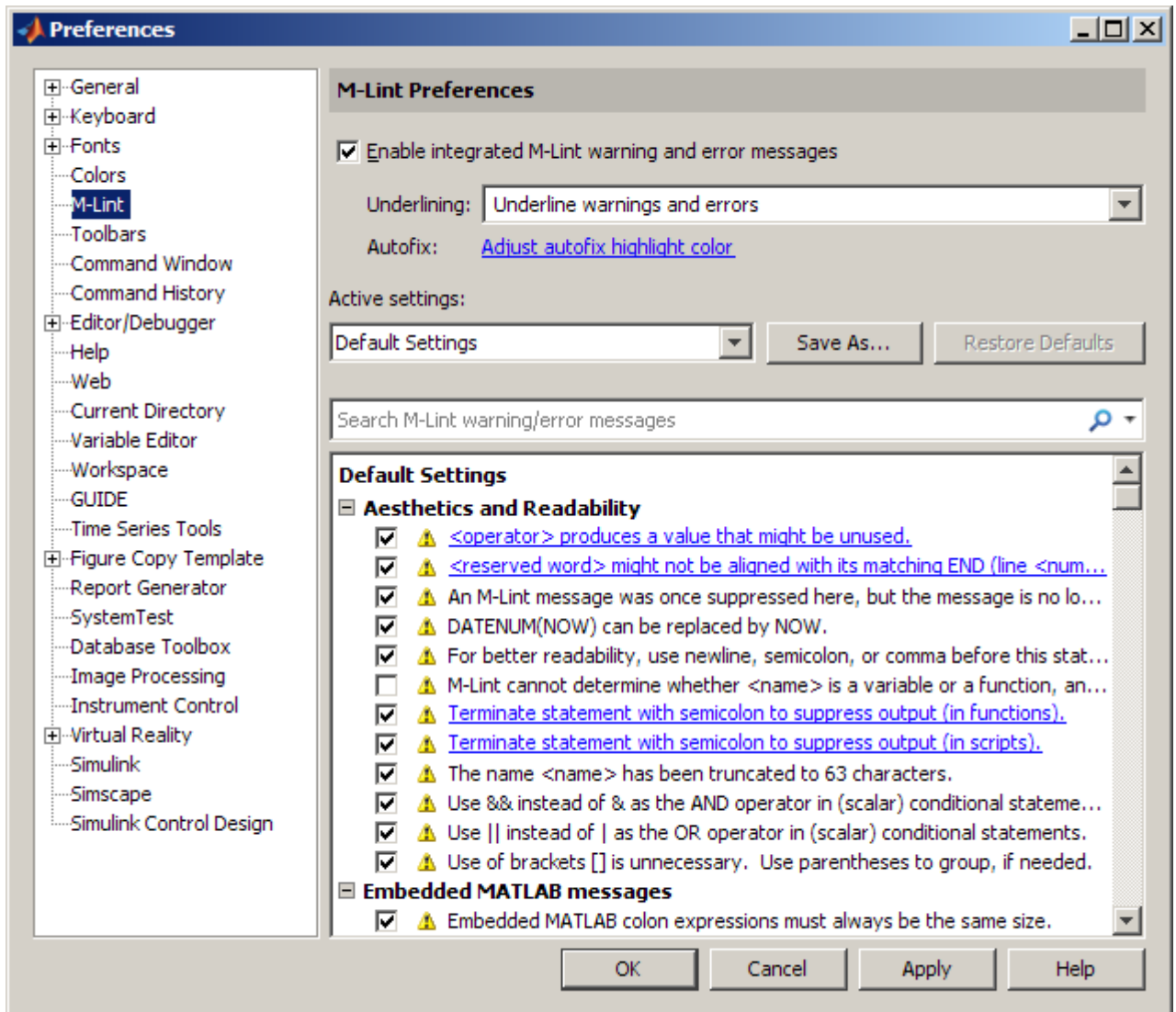
2 Click **OK**.

3 Open an M-file in the Editor. This example uses the sample file lengthofline.m that ships with the MATLAB software:

a Open the example file:

```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...
        'examples','lengthofline.m'))
```

b Save the example file to a directory to which you have write access. For the example, lengthofline.m is saved to I:\MATLABFiles\mymfiles.

4 The M-Lint message indicator at the top right edge of the window conveys the M-Lint messages reported for the file:

- **Red** indicates syntax errors were detected. Another way to detect some of these errors is using syntax highlighting to identify unterminated strings, and delimiter matching to identify unmatched keywords, parentheses, braces, and brackets.

- **Orange** indicates warnings or opportunities for improvement, but no errors, were detected.

- **Green** indicates no errors, warnings, or opportunities for improvement were detected.

For the example, the indicator is red, meaning there is at least one error in the file.

M-Lint message indicator for all messages in file:
Red: Errors detected
Orange: Warnings or improvement opportunities detected
Green: No errors, warnings or improvement opprotunities detected
Click indicator to go to next line that has an associated M-Lint message. ⌐

Current cursor position

5 Click the M-Lint message indicator to go to the next code fragment containing an M-Lint message. The next code fragment is relative to the current cursor position, viewable in the status bar.

In the `lengthofline` example, the first message is at line 22. The cursor moves to the beginning of line 22.

The code fragment for which there is an M-Lint message is underlined in either red for errors or orange for warnings and improvement opportunities.

6 View the M-Lint message by moving the mouse pointer within the underlined fragment. The message appears with a yellow highlighted background, similar to data tips (see "Viewing Values as Data Tips in the Editor" on page 6-166).

Position cursor within orange underlined code fragment and the Editor displays the related M-Lint message.
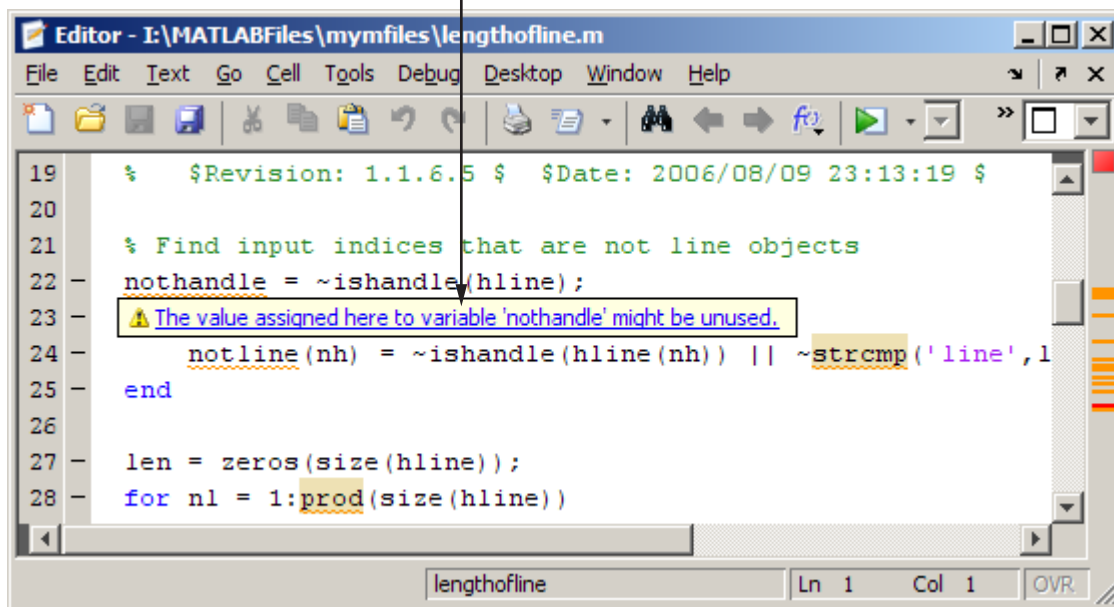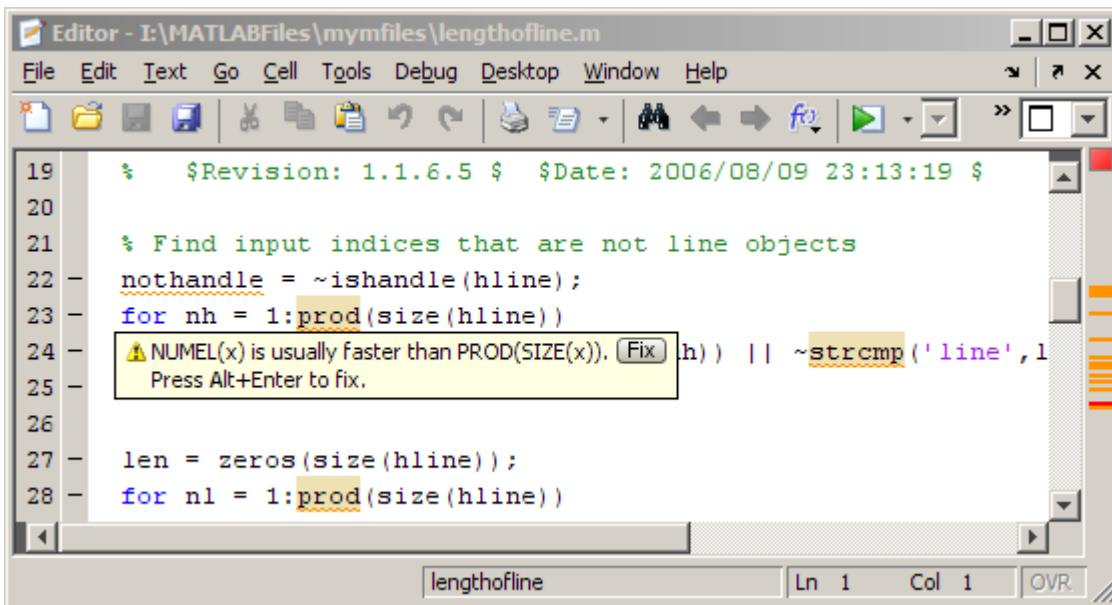
This message means that in line 22, `nothandle` is assigned a value, but is probably not used anywhere after that in the file. The line might be extraneous and you could delete it. But it might be that you actually

intended to use the variable, as shown in step 7 of this example. Notice that the message is a link to additional information. M-Lint message links are discussed in steps 9 and 10.

7 Make changes to your code as needed. The M-Lint indicator and underlining automatically update to reflect the changes you make, even if you do not save the file.

In this example, the intention was to use `nothandle` as a performance improvement by determining the value prior to the loop. Changing `~ishandle(hline(nh))` in line 24 to `nothandle(nh)` means there is no longer an M-Lint message associated with line 22. For more information about what the warning and improvement messages in this example mean and actions you can take to address them, see "Messages and Resulting Changes for the lengthofline Example" on page 7-27.

8 In `lengthofline`, line 23, `prod` is underlined because there is an M-Lint warning, and it is highlighted because an automatic fix is available. When you view the M-Lint message, it provides a button to apply the automatic fix. It also indicates the keyboard controls (**Alt+Enter**) to apply it.

To view what the automated fix is, right-click the highlighted code (for a single-button mouse, use **Ctrl**+click). The first item in the context menu indicates the automatic fix that M-Lint can perform. Select it and M-Lint automatically corrects the code. In this example, M-Lint replaces `prod(size(hline))` with `numel(hline)`.
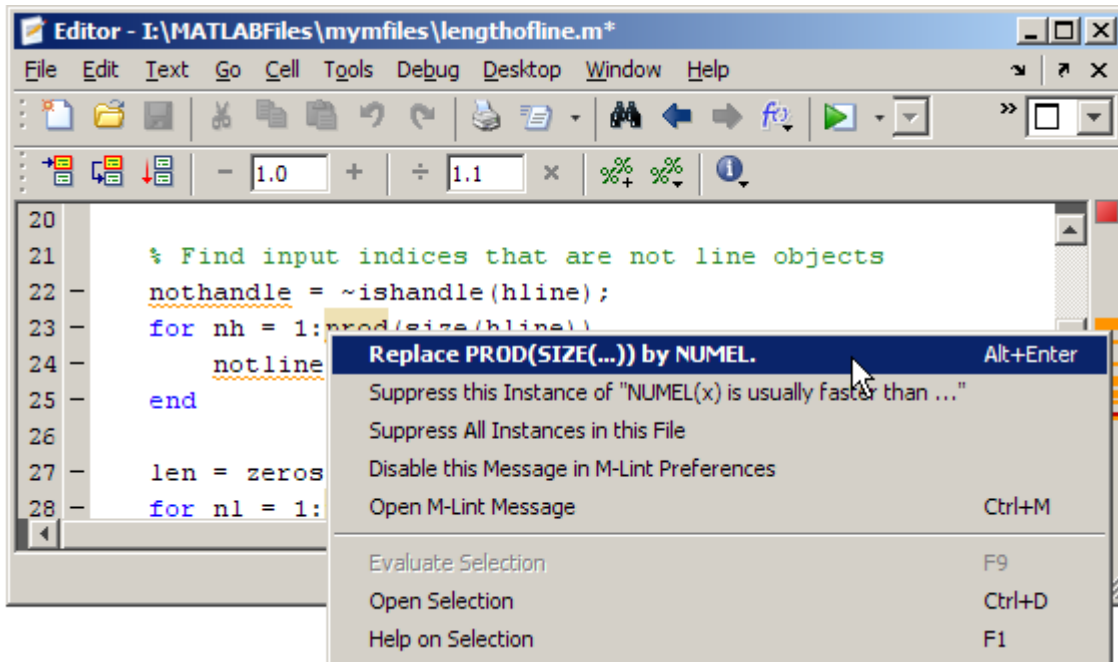
After you apply the fix, M-Lint removes the underline from `prod` in line 23.

9 Click the M-Lint message indicator to go to the next message, on line 24. When you view this M-Lint message, it appears as a link to indicate that there is more information about the message. Not all messages have additional information.
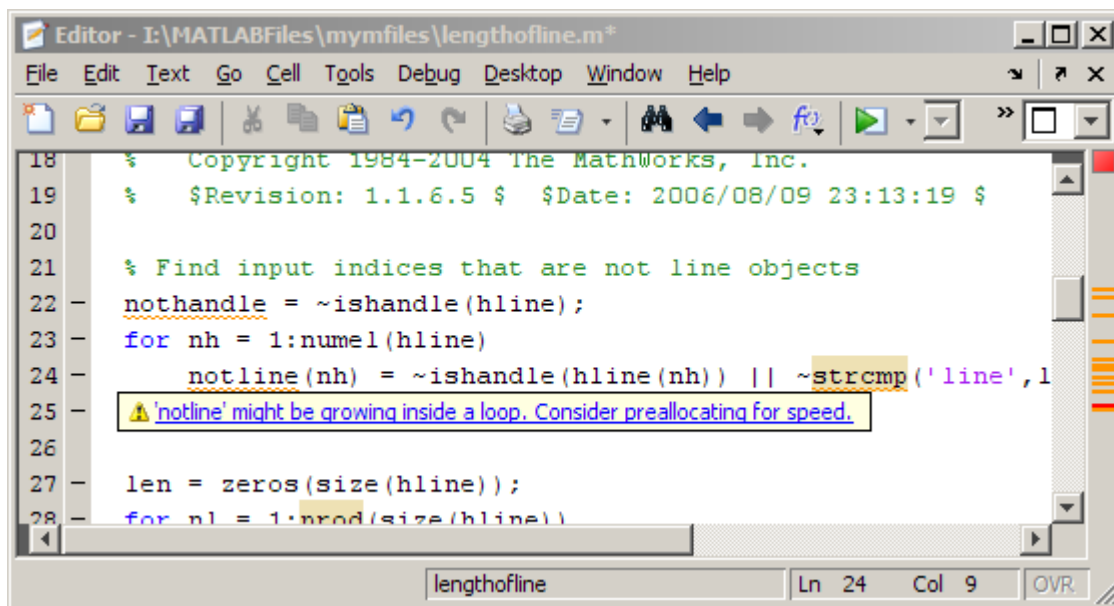
10 Click the link in the M-Lint Message. The window expands to display an explanation and user action.

11 Click the M-Lint message indicator to go to the next message, or use the M-Lint message bar.

Each marker in the bar represents one or more lines that have associated M-Lint messages.

a Position the mouse pointer at a marker in the message bar to view the message. For example, to see an error in `lengthofline`, position the pointer at a red (error) marker in the message bar. There is only one

error in the file and with the pointer positioned over it, the associated M-Lint messages appears. (There can be multiple messages per line.) Click the marker to go to the first code fragment in the line that resulted in an M-Lint message. For the example, click the red marker, which takes you to the first suspect code fragment in line 48.

```
temp = diff([data{1}(:) data{2}(:) data{3}(:)]);
```

Multiple messages can represent a single problem or multiple problems. Addressing one might address all of them, or after addressing one, the other messages might change or what you need to do might become clearer.
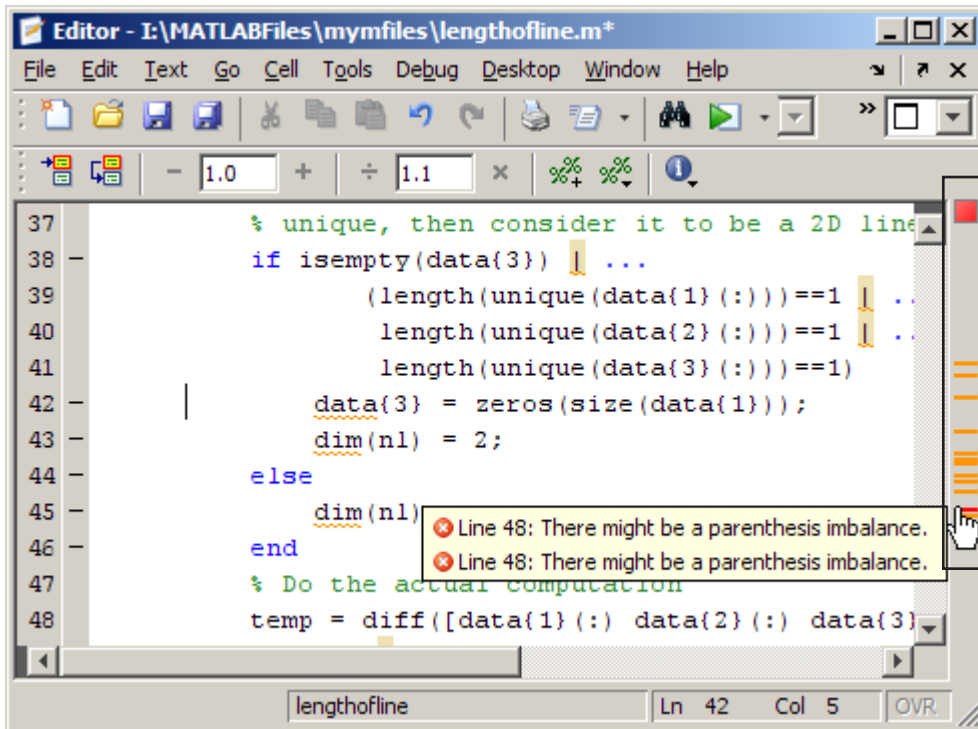
Each marker represents an M-Lint message.

To view a message, position the pointer over a marker.

To go to the code fragment that resulted in the message, click a marker.

b Make changes to address the problem noted in the M-Lint message—the M-Lint indicators update automatically.

In the example, the M-Lint message suggest a delimiter imbalance. You can check that by following these steps:

i Choose **File > Preferences > Keyboard > Delimiter Matching**, and then select **Match on arrow key**, if it is not already selected.

ii Move the arrow key over each of the delimiters to see if MATLAB indicates a mismatch.

In the example, it may appear that there are no mismatched delimiters. However, M-Lint detects the semicolon in parentheses: `data{3}(;)`, and interprets it as the end of a statement. M-Lint reports that the two statements on line 48 each have a delimiter imbalance.

iii In line 48, change `data{3}(;)` to `data{3}(:)`.

When you make the change, the underline no longer appears in line 48. M-Lint now interprets the code at line 48 as a single statement. The single change addresses the issues in both of the M-Lint messages for line 48.

Because the change you made removed the only error in the file, the M-Lint message indicator at the top of the bar changes from red to orange, indicating that only warnings and potential improvements remain.

If there are multiple messages associated with a line, there might be multiple underlined code fragments that are adjacent, as in the previous step, making it difficult to display the message of interest. In these cases, it is easier to view the messages through the marker on the message bar than moving the arrow over each delimiter.

After making changes to address all M-Lint messages, or disabling designated messages, the M-Lint message indicator becomes green. The example file with all M-Lint messages addressed has been saved as `lengthofline2.m`. Open the example file with the command:

```
open(fullfile(matlabroot,'help','techdoc',...
    'matlab_env', 'examples','lengthofline2.m'))
```
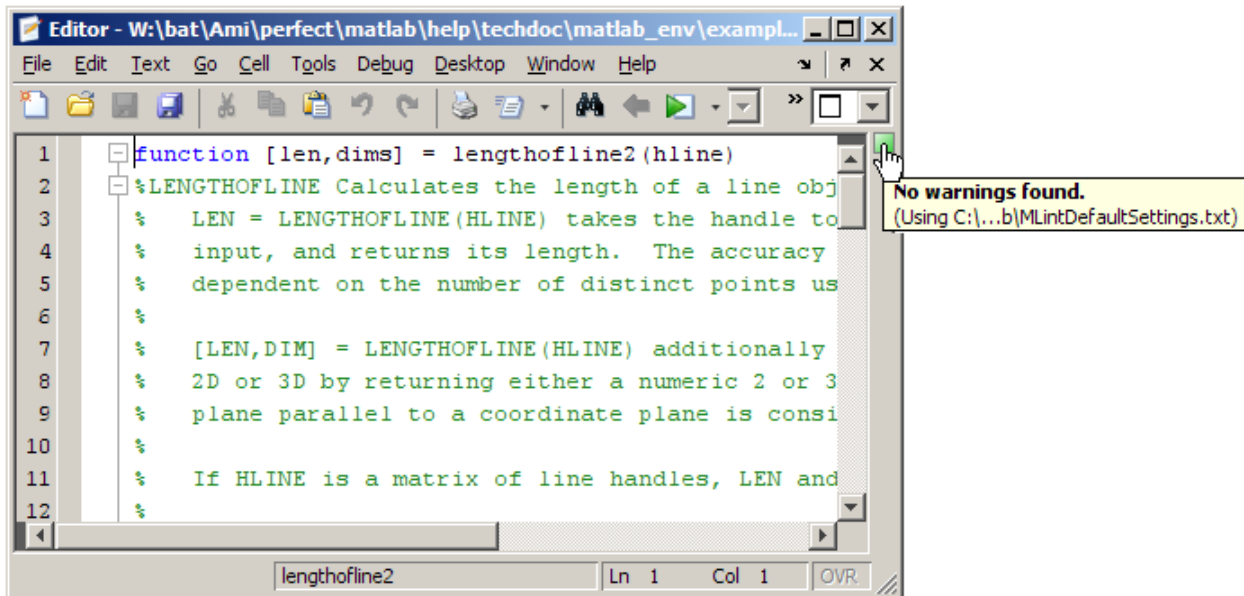
Suppressing M-Lint Indicators and Messages

Depending on the stage at which you are in completing the M-file, you might want to restrict the M-Lint underlining. You can do this by using the M-Lint preference referred to in step 1, in "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 6-123. For example, when first coding, you might prefer to underline only errors because warnings would be distracting. For details, click the **Help** button in the Preferences dialog box.

M-Lint does not provide perfect information about every situation and in some cases, you might not want to make any changes based on an M-Lint message. In the event you do not want to change the code, and you also do not want to see the M-Lint indicator and message for that line, instruct M-Lint to suppress them. For the `lengthofline` example, in line 49, the first M-Lint message is `Terminate statement with semicolon to suppress output (in functions)`. Adding a semicolon to the end of a statement suppresses output and is a common practice. M-Lint alerts you to lines that produce output, but lack the terminating semicolon. If you want to view output from line 49, do not add the semicolon as M-Lint suggests.

There are a few different ways to suppress (turn off) the M-Lint indicators for warning and error messages:

- "Suppress an Instance of an M-Lint Message in the Current File" on page 6-137

- "Suppress All Instances of an M-Lint Message in the Current File" on page 6-139

- "Suppress All Instances of an M-Lint Message in All Files" on page 6-140

- "Specify Default M-File Message Settings" on page 6-140

- "Specify Nondefault M-File Message Settings" on page 6-141

- "Understanding M-File Code Containing Suppressed M-Lint Messages" on page 6-143

You cannot suppress M-Lint error messages such as syntax errors. Therefore, instructions on suppressing M-Lint messages do not apply to those types of messages.

Suppress an Instance of an M-Lint Message in the Current File

You can instruct M-Lint to suppress a specific instance of a message in the current file. For example, using the code presented in "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 6-123, follow these steps:

1 In line 49, right-click at the first M-Lint underline (for a single-button mouse, use **Ctrl**+click) .

2 From the context menu, select `Suppress this Instance of "Terminate statement with semicolon...".`

M-Lint adds a `%#ok<NOPRT>` to the end of the line, which instructs MATLAB not to check for a terminating semicolon at that line. M-Lint removes the underline and mark in the M-Lint indicator bar for that message.

If there are two messages on a line that you do not want M-Lint to display, right-click separately at each underline and select the appropriate entry from the context menu. M-Lint expands the `%#ok` syntax. For the example, in the code presented in "Checking M-File Code for Problems Using the

M-Lint Code Analyzer" on page 6-123, ignoring both messages for line 49 adds `%#ok<NBRAK,NOPRT>`.

Even if M-Lint preferences are set to enable this message, the specific instance of the message suppressed in this way does not appear because the `%#ok` takes precedence over the preference setting. If you later decide you want M-Lint to check for a terminating semicolon at that line, delete the `%#ok<NOPRT>` string from the line.

For more information about `%#ok`, see the `mlint` function reference page.
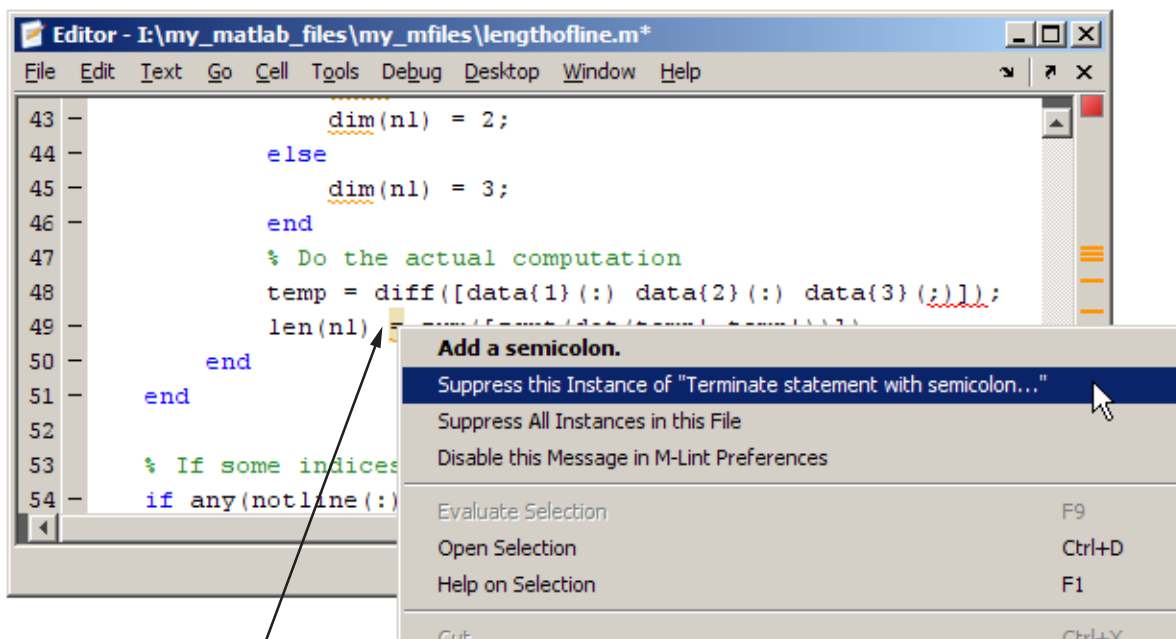
Right-click at an M-Lint underline and select
the option instructing M-Lint to suppress only
this instance of the message.

M-Lint adds %#ok for a specific message to the
end of the line for which you specified that the M-Lint
message be suppressed.

Suppress All Instances of an M-Lint Message in the Current File

You can instruct M-Lint to suppress all instances of a specific message in
the current file. For example, using the code presented in "Checking M-File
Code for Problems Using the M-Lint Code Analyzer" on page 6-123, follow
these steps:

1 In line 49, right-click at the first M-Lint underline (for a single-button
mouse, use **Ctrl**+click).

2 From the context menu, select `Suppress all Instances in this File`.

M-Lint adds a `%#ok<*NOPRT>` to the end of the line, which instructs MATLAB
to not check for a terminating semicolon throughout the file. M-Lint removes
all underlines from the code, as well as marks in the M-Lint indicator bar that
correspond to this message.

If there are two messages on a line that you do not want M-Lint to display any instances of in the current file, right-click separately at each underline, and then select the appropriate entry from the context menu. M-Lint expands the %#ok syntax. For the example, in the code presented in "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 6-123, ignoring both messages for line 49 adds %#ok<*NBRAK,*NOPRT>.

Even if M-Lint preferences are set to enable this message, the message does not appear because the %#ok takes precedence over the preference setting. If you later decide you want M-Lint to check for a terminating semicolon in the file, delete the %#ok<*NOPRT> string from the line.

For more information about %#ok, see the mlint function reference page.

Suppress All Instances of an M-Lint Message in All Files

You can instruct M-Lint to disable all instances of an M-Lint message in all files. For example, using the code presented in "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 6-123, follow these steps:

1 In line 49, right-click at the first M-Lint underline (for a single-button mouse, use **Ctrl**+click).

2 From the context menu, select Disable this Message in M-Lint Preferences.

This modifies the M-Lint preference setting. For more information about M-Lint preferences, including how to restore MATLAB default settings, select **File > Preferences > M-Lint**, and then click **Help**.

Specify Default M-File Message Settings

You can specify that you want certain messages to be disabled by default when you open any M-file. Typically, you do this if you find that you do not want certain messages or categories of messages to be enabled for all or most of your M-files.

Follow these steps:

1 Select **File > Preferences > M-Lint**

The Preferences dialog box opens and displays the M-Lint Preferences panel.

2 Disable specific messages, or categories of messages.

The **Active Settings** field now contains the value `Default Settings (modified)`.

3 Click Apply.

Now, each M-file you open uses the modified default settings. If you want to restore the factory-installed default settings, decide if you want to save the current settings to a file, as described in "Specify Nondefault M-File Message Settings" on page 6-141, and then click **Restore Defaults**.

For more information about M-Lint settings and preferences, click **Help** in the M-Lint preferences panel.

Specify Nondefault M-File Message Settings

You can specify that you want certain messages to be enabled or disabled, and then save those settings to a file. When you want to use a settings file with a particular M-file, you select it from the M-Lint preferences panel. That setting file remains in effect until you select another settings file. Typically, you change the settings file when you have a subset of M-files for which you want to use a particular settings file.

Follow these steps:

1 Select **File > Preferences > M-Lint**

The Preferences dialog box opens and displays the M-Lint Preferences panel.

2 Enable or disable specific messages, or categories of messages.

3 Click **Save as** and save the settings to a `txt` file.

4 Click **OK**.

You can reuse these settings for any M-file, or provide the settings file to another user.

To use the saved settings:

1 In the Editor, select **Tools > M-Lint**.

The currently-selected setting choice displays, preceded by a bullet point.

2 Choose from any of the settings files, such as the `MLintNoSemis` example, as shown here.

The settings you choose are in effect for all M-files until you select another set of M-Lint settings.

M-Lint default settings are currently selected (as indicated by the bullet preceding that menu item).

Select any settings file to use its M-Lint settings.
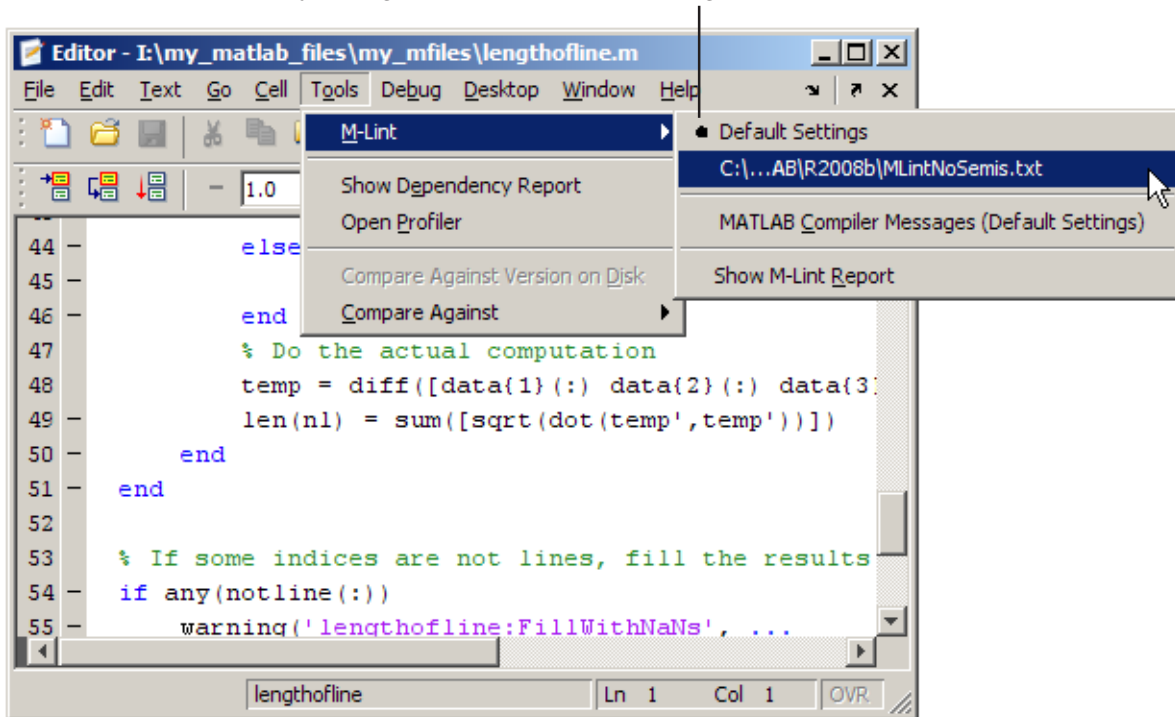
Understanding M-File Code Containing Suppressed M-Lint Messages

If you are given code that contains suppressed M-Lint messages, you might want to review those messages without the need to unsuppress them first. A messages might be in a suppressed state for any of the following reasons:

- One or more `%#ok<message-ID>` pragmas are on a line of code that elicits an M-Lint message specified by `<message-ID>`.

- One or more `%#ok<*message-ID>` pragmas are in a file that elicits the M-Lint message specified by `<message-ID>`.

- It is deselected in the M-Lint Preferences panel.

- It is disabled by default.

To determine the reasons why some M-Lint messages might be suppressed, follow these steps:

1 Search the M-file for the `%#ok` pragma and create a list of all the message-IDs associated with that pragma.

2 Open the M-Lint Preferences dialog box by choosing **File > Preferences > M-Lint**.

3 In the filter field, enter one of the message IDs, if any, you found in step 1.

The message list now contains only the message that corresponds to that ID. If the message is a hyperlink, you can click it to see an explanation and suggested action for the message. This might provide insight into why the message is suppressed or disabled. The following image shows how the Preferences dialog box appears when you enter `CPROP` in the filter field, for example.
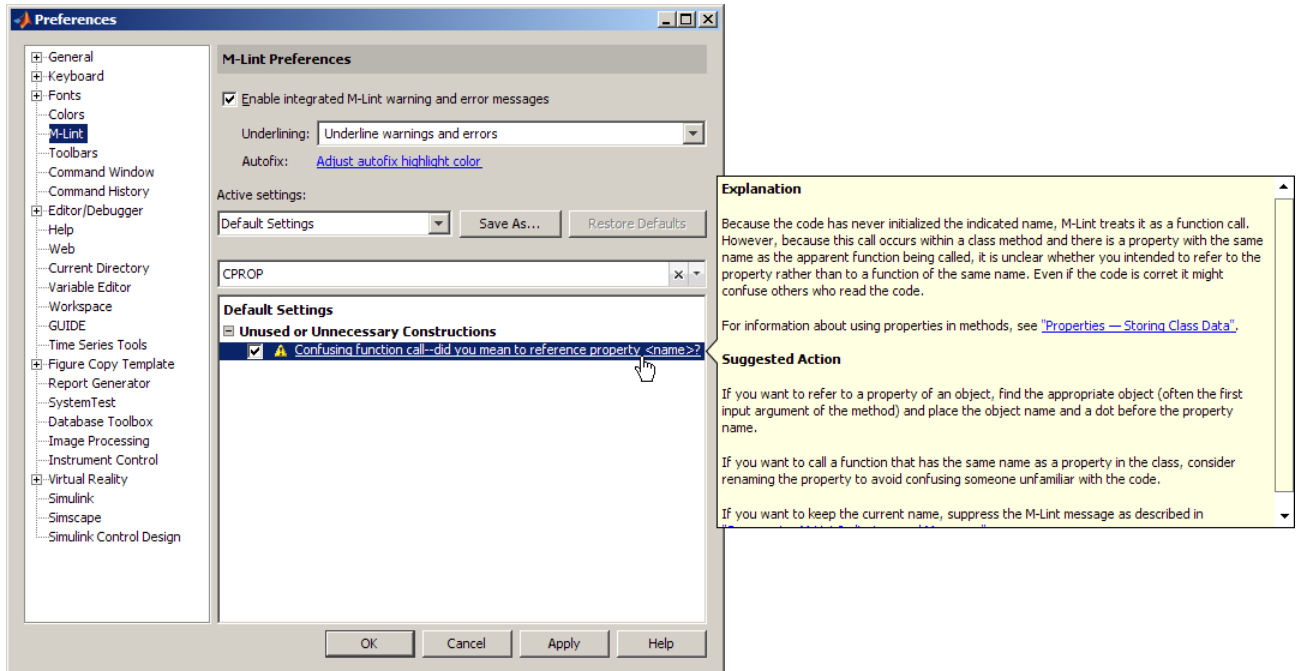
4 Click the ⊠ button to clear the filter field, and then repeat step 3 for each message ID you found in step 1.

5 Click the ⊠ button to clear the filter field.

6 Filter the messages to display those that are disabled by default and disabled via the Preferences panel by clicking the down arrow to the right of the filter field, and then clicking **Show Disabled Messages**.

The message list now contains all the messages that are disabled in the Preferences panel.

7 Review the message associated with each message ID to understand why it might be suppressed in the code or disabled in Preferences.

Setting Preferences for M-Lint

Use M-Lint preferences to adjust how the M-Lint code analyzer operates. The preferences apply to M-Lint usage with the Editor, the Embedded MATLAB™

Editor (if you have products which use that tool), and the M-Lint Code Check Report, with a few exceptions that are noted. For more information about using M-Lint, see "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 6-123.

This section contains information about the following topics:

- Enabling integrated M-Lint warning and error messages in the Editor
- Restricting M-Lint Underlining in the Editor
- Changing the Color M-Lint Uses to Indicate an Automatic Fix Is Available
- Choosing the Messages to Display in Your Code
- Filtering the Messages Displayed in the Preferences Panel
- Saving Your Preferred Settings
- Using Your Saved Settings
- Using Default Settings
- "Enabling MATLAB Compiler Deployment Messages" on page 6-153

Enable integrated M-Lint warning and error messages

Select this check box if you want the Editor to show M-Lint messages in the M-file. This preference does *not* apply to the M-Lint Code Check Report. When you select this preference, the Editor provides visual cues that alert you to potential errors and problems, as well as opportunities for improvement in your code. These visual cues take the form of underlines and a message indicator bar. From these cues, you can view a message for each line of an M-file that M-Lint indicates might be improved. For example, a common M-Lint message is that a variable is defined but never used in the M-file.

Underlining. Restrict the M-Lint underlining notification using the associated drop-down list. The list is available only when **Enable integrated M-Lint warning and error messages** is selected. Options are

- Underline warnings and errors
- Underline errors only
- No underlines

Errors are underlined in red and warnings are underlined in orange. For all of the underlining options, the Editor provides the marks for errors and warnings in the indicator bar.

You might choose a different option at different stages in your workflow. For example, when first coding, you might prefer no underlines because they would appear as you enter a statement and might be distracting. Later, you might choose to underline only errors to help you debug your file. Finally, when tweaking an existing M-file, you might want to underline warnings and errors because the file is in a state that you can fix any issues you introduce at the time.

Autofix. Click **Adjust autofix highlight color** to open the Colors Preferences dialog box so that you can adjust the color that M-Lint uses to highlight errors and warnings that it can fix automatically. By default, this color is pale orange.

In the Color Preferences dialog box, in the **Other colors** group, use the M-Lint autofix highlight palette to select a color. For more information, see "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 6-123.

Active settings

Use M-Lint settings to show or hide specific messages or categories of messages from displaying in your code when you have M-Lint warnings and error messages enabled. If you are new to using M-Lint or MATLAB, use the default settings. Once you are familiar with M-Lint and MATLAB, you might want to suppress the display of certain M-Lint messages.

To suppress an M-Lint message on a line-by-line or file-by-file basis, see "Suppressing M-Lint Indicators and Messages" on page 6-136, or the `mlint` function. To suppress messages in more than one file, it is more convenient to disable the M-Lint preference settings, as described here:

1 Optionally, use the filter field to filter the messages displayed in the Active settings table. For details, see "Filtering M-Lint Messages" on page 6-148.

2 Optionally, click the link for a given message (if available), to get more information about that message, including an explanation and suggested action.

3 In the Active settings table, select ☑ or clear ☐ the check boxes to enable or disable messages, respectively. If you want to enable or disable a set of messages simultaneously, highlight the messages in the Active settings table, right-click, and then choose **Enable** or **Disable**.
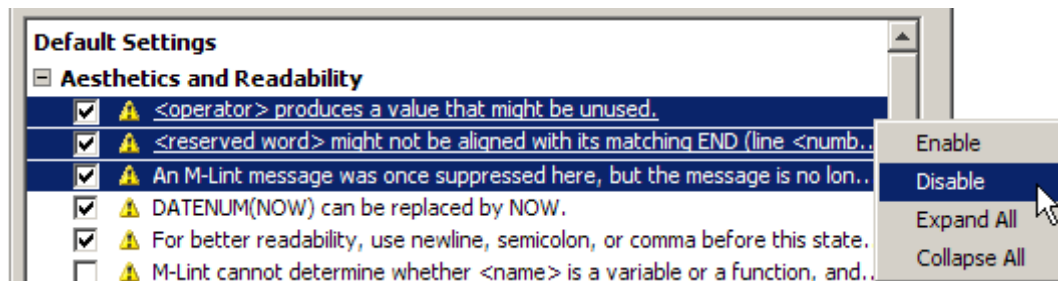
M-Lint shows enabled messages and hides disabled messages in the Editor.

4 Click **Apply** or **OK** to save the changes.

If you click one of these buttons, and then try to choose a different option from the **Active settings** drop-down list, MATLAB prompts you to save changes—see "Saving Settings" on page 6-152. Like all preferences, the settings are retained for you next MATLAB session.

Note The **MATLAB Compiler (deployment) messages** category is the only one that you can enable or disable by category. The M-Lint preferences panel does not display the **MATLAB Compiler (deployment) messages** category if the MATLAB® Compiler™ is not installed.

Filtering M-Lint Messages. To filter the list of messages in the Preferences dialog box to display only those that are currently of interest to you, use any combination of the methods presented in the table that follows:

To see a list of messages that:	Do this:
Contain a given string in the short message, extended message, message category, or the message ID. You might do this for example because you recall seeing a message containing a certain string that you want to review, but you can't remember the exact message text.	Type the string in the filter field. For example, type com in the filter field to display those messages that contain that string in the short message, extended message, or message ID.
Correspond to a given message ID. You might do this, for example, if you are reviewing someone else's code and you want to see the message that corresponds to one that is suppressed using the %#ok<message-ID> pragma.	Type msgid: followed by the message ID in the filter field. For example: msgid:agrow. If a link is provided, click it to get an explanation of what might cause M-Lint to display this message in your code.
You can set using M-Lint Preferences. That is, to see the complete list of messages after you have filtered the messages on a given string or filter menu option.	Click the down arrow to the right of the search field, and then click **Show All** .
Are different from the default setting (of enabled or disabled) .	Click the down arrow to the right of the search field, and then click **Show Messages Modified from Default**. A grey dot precedes a message with a setting different from the default. For example: ● ☐ ⚠ DATENUM(NOW) can be replaced by NOW.
Are in a given category.	Click the down arrow to the right of the filter field, click **Show Messages in Category**, and then click the category you want.
Are warnings.	Click the down arrow to the right of the filter field, and choose **Show All Warnings**. An exclamation point in a yellow triangle ⚠ indicates a warning message.

To see a list of messages that:	Do this:
Are errors.	Click the down arrow to the right of the filter field, and choose **Show All Errors**. By default, an X in a red dot indicates an error message, ⊗. See "M-File Syntax Highlighting Colors" on page 2-135 for information on changing the color for errors.
Are disabled.	Click the down arrow to the right of the filter field, and then choose **Show Disabled Messages**.

Note When you filter messages, the state of a category is not considered. Therefore, if the **MATLAB Compiler (deployment) messages** category is disabled, but all the messages within it are enabled, the M-Lint preferences panel does not display the MATLAB Compiler (deployment) messages when you select **Show Disabled Messages**. The M-Lint preferences panel does not display the **MATLAB Compiler (deployment) messages** category if the MATLAB Compiler is not installed.

Example of Filtering Messages. To display error messages that contain the string comma and are disabled, do the following:

1 Click the arrow next to the filter field and select **Show All Errors**.

The filter field contains the string severity:error.
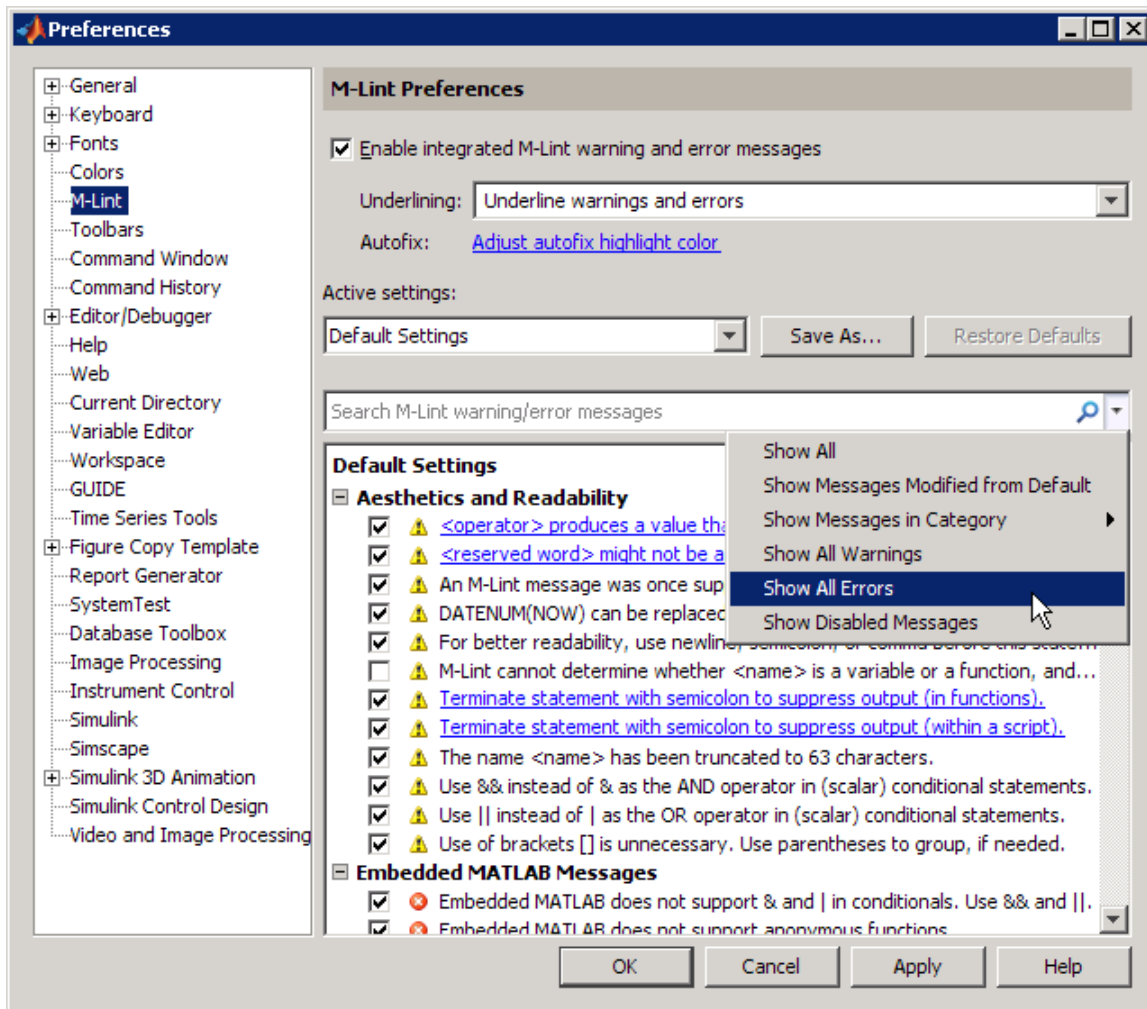
2 At the end of the string `severity:error`, press the **Space** key and then type `comma`.

3 Click the arrow next to the filter field and select **Show Disabled Messages**.

The filter field now contains the string, `severity:error comma enabled:false` and only the M-Lint messages that fulfill those requirements display in the Preferences panel.

To restore the list of all M-Lint messages, click the ⊠ button.

Saving Settings. If you are likely to use different settings at different times, or if you want to make these settings available to other users, click **Save As**. MATLAB prompts you to provide the name of the `txt` file in which the settings are saved. The default location for settings is the MATLAB preferences directory (the directory returned when you run `prefdir`), although you can choose a different directory when saving.

Using Saved Settings. To use settings previously saved, select the settings `txt` file from the **Active settings** drop-down list. Choose from the recently-used settings files, which are shown, or select `Browse` to locate the settings file. Then click **Apply** or **OK** to make the settings take effect. You can also access saved settings from within the Editor using **Tools > M-Lint**, or the M-Lint message bar.

After selecting a settings file, you can modify the settings, but your changes automatically modify the `txt` file. If you want to retain the settings in the `txt` file, you need to create a copy of the settings file in which you can make changes. To do so, click **Save As** and save the file to a different name. Then make changes to the newly named file.

Default Settings. The **Active settings** indicator shows `Default Settings` when you are using the default settings rather than settings from a `txt` file. The term `(modified)` appears when you have made changes to default settings but have not saved the changes to a file. To undo any unsaved changes and return to the default settings, click **Restore Defaults**. If you think you will use the modified default settings in a future session, save the settings as described in "Saving Settings" on page 6-152.

When a `txt` file is selected, return to default settings by using the drop-down list to select `Default Settings` or `Default Settings (modified)`.

Enabling MATLAB Compiler Deployment Messages

You can switch between showing or hiding Compiler deployment messages when you work on a file by changing the M-Lint preference for this category of messages. Your choice likely depends on whether or not you are working on a file to be deployed. When you change the preference, it also changes the setting in the Editor. The converse is also true—when you change the setting from the Editor, it effectively changes this preference, however you will not see the changes reflected in the **M-Lint Preferences** if Preferences is open at the time you make the change. Whether or not you change the setting from the Editor or from **M-Lint Preferences**, it applies to the Editor and to the M-Lint Code Check Report.

To enable MATLAB Compiler deployment messages, follow these steps:

1 Choose **File > Preferences > M-Lint**.

2 Click the down arrow next to the filter field, and then choose **Messages in Category > MATLAB Compiler (deployment) messages.**

3 Click the **Enable Category** button.

4 Deselect individual message that you do not want M-Lint to display for your code (if any).

The settings `txt` file, which you can create as described in "Saving Settings" on page 6-152, includes the status of this setting.

Debugging Process and Features

Ways to Debug M-Files

You can debug the M-files using the Editor, which is a graphical user interface, as well as by using debugging functions from the Command Window. You can use both methods interchangeably. These topics and the example describe both methods.

Preparing for Debugging

Do the following to prepare for debugging:

- Open the file — To use the Editor for debugging, open it with the file to run.

- Save changes — If you are editing the file, save the changes before you begin debugging. If you try to run a file with unsaved changes from within the Editor, the file is automatically saved before it runs. If you run a file with unsaved changes from the Command Window, MATLAB software runs the saved version of the file, so you will not see the results of your changes.

- Add the files to a directory on the search path or put them in the current directory — Be sure the file you run and any files it calls are in directories

that are on the search path. If all files to be used are in the same directory, you can instead make that directory be the current directory.

Debugging Example — The Collatz Problem

The debugging process and features are best described using an example. To prepare to use the example, create two M-files, `collatz.m` and `collatzplot.m`, that produce data for the Collatz problem.

For any given positive integer, `n`, the `Collatz` function produces a sequence of numbers that always resolves to 1. If `n` is even, divide it by 2 to get the next integer in the sequence. If `n` is odd, multiply it by 3 and add 1 to get the next integer in the sequence. Repeat the steps until the next integer is 1. The number of integers in the sequence varies, depending on the starting value, `n`.

The Collatz problem is to prove that the Collatz function will resolve to 1 for all positive integers. The M-files for this example are useful for studying the Collatz problem. The file `collatz.m` generates the sequence of integers for any given `n`. The file `collatzplot.m` calculates the number of integers in the sequence for all integers from 1 through `m`, and plots the results. The plot shows patterns that can be further studied.

Following are the results when `n` is 1, 2, or 3.

n	Sequence	Number of Integers in the Sequence
1	1	1
2	2 1	2
3	3 10 5 16 8 4 2 1	8

M-Files for the Collatz Problem. Following are the two M-files you use for the debugging example. To create these files on your system, open two new M-files. Select and copy the following code from the Help browser and paste it into the M-files. Save and name the files `collatz.m` and `collatzplot.m`. Save them to your current directory or add the directory where you save them to the search path. One of the files has an embedded error to illustrate the debugging features.

Code for collatz.m.

```
function sequence=collatz(n)
% Collatz problem. Generate a sequence of integers resolving to 1
% For any positive integer, n:
% Divide n by 2 if n is even
% Multiply n by 3 and add 1 if n is odd
% Repeat for the result
% Continue until the result is 1

sequence = n;
next_value = n;
while next_value > 1
 if rem(next_value,2)==0
  next_value = next_value/2;
 else
  next_value = 3*next_value+1;
 end
 sequence = [sequence, next_value];
end
```

Code for collatzplot.m.

```
function collatzplot(m)
% Plot length of sequence for Collatz problem
% Prepare figure
clf
set(gca,'XScale','linear')
%
% Determine and plot sequence and sequence length
for N = 1:m
 plot_seq = collatz(N);
 seq_length(N) = length(plot_seq);
 line(N,plot_seq,'Marker','.','MarkerSize',9,'Color','blue')
 drawnow
end
```

Alternatively, you can open the files by issuing the following commands, and then save each file to a local directory:

```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...
```

```
 'examples','collatz.m'))

open(fullfile(matlabroot,'help','techdoc','matlab_env',...
 'examples','collatzplot.m'))
```

Trial Run for Example. Open the file collatzplot.m. Make sure the current directory is the directory in which you saved collatzplot.

Try out collatzplot to see if it works correctly. Use a simple input value, for example, 3, and compare the results to those shown in the preceding table. Typing

```
collatzplot(3)
```

produces the plot shown in the following figure.
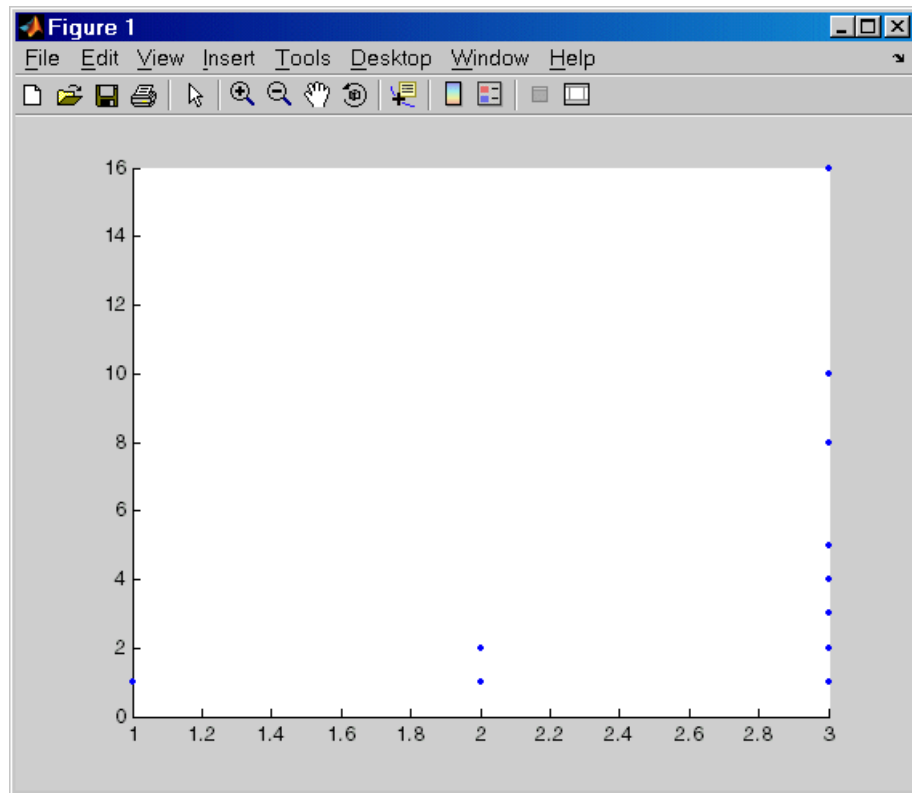
The plot for n = 1 appears to be correct—for 1, the Collatz series is 1, and contains one integer. But for n = 2 and n = 3, it is wrong because there should be only one value plotted for each integer, the number of integers in the sequence, which the preceding table shows to be 2 (for n = 2) and 8 (for n = 3). Instead, multiple values are plotted. Use MATLAB debugging features to isolate the problem.

Setting Breakpoints

Set breakpoints to pause execution of the M-file so you can examine values where you think the problem might be. You can set breakpoints in the Editor, using functions in the Command Window, or both.

There are three basic types of breakpoints you can set in M-files:

- A standard breakpoint, which stops at a specified line in an M-file. For details, see "Setting Standard Breakpoints" on page 6-160.

- A conditional breakpoint, which stops at a specified line in an M-file only under specified conditions. For details, see "Conditional Breakpoints" on page 6-178.

- An error breakpoint that stops in any M-file when it produces the specified type of warning, error, or NaN or infinite value. For details, see "Error Breakpoints" on page 6-181.

You can disable standard and conditional breakpoints so that MATLAB temporarily ignores them, or you can remove them. For details, see "Disabling and Clearing Breakpoints" on page 6-172. Breakpoints are not maintained after you exit the MATLAB session.

You can only set valid standard and conditional breakpoints at executable lines in saved files that are in the current directory or in directories on the search path. When you add or remove a breakpoint in a file that is not in a directory on the search path or in the current directory, a dialog box appears, presenting you with options that allow you to add or remove the breakpoint. You can either change the current directory to the directory containing the file, or you can add the directory containing the file to the search path.

Do not set a breakpoint at a `for` statement if you want to examine values at increments in the loop. For example, in

```
for n = 1:10
    m = n+1;
end
```

MATLAB executes the `for` statement only once, which is efficient. Therefore, when you set a breakpoint at the `for` statement and step through the file, you only stop at the `for` statement once. Instead place the breakpoint at the next line, `m=n+1` to stop at each pass through the loop.

You cannot set breakpoints while MATLAB is busy, for example, running an M-file, unless that M-file is paused at a breakpoint.

Setting Standard Breakpoints

To set a standard breakpoint using the Editor, click in the breakpoint alley at the line where you want to set the breakpoint. The breakpoint alley is the narrow column on the left side of the Editor, just right of the line number. Set breakpoints at lines that are preceded by a - (dash). Lines not preceded by a dash, such as comments or blank lines, are not executable—if you try to set a breakpoint there, it is actually set at the next executable line. Other ways to set a breakpoint are to position the cursor in the line and then click the Set/clear breakpoint button 🔲 on the toolbar, or select **Set/Clear Breakpoint** from the **Debug** menu or the context menu. A breakpoint icon appears.

Set Breakpoints for the Example. It is unclear whether the problem in the example is in collatzplot or collatz. To start, set breakpoints in collatzplot.m at lines 9, 10, and 11. The breakpoint at line 9 enables you to step into collatz to see if the problem might be there. The breakpoints at lines 10 and 11 stop the program where you can examine the interim results.

Click where there is a dash (-) to set a breakpoint at that line.

A red icon indicates a valid breakpoint is set.
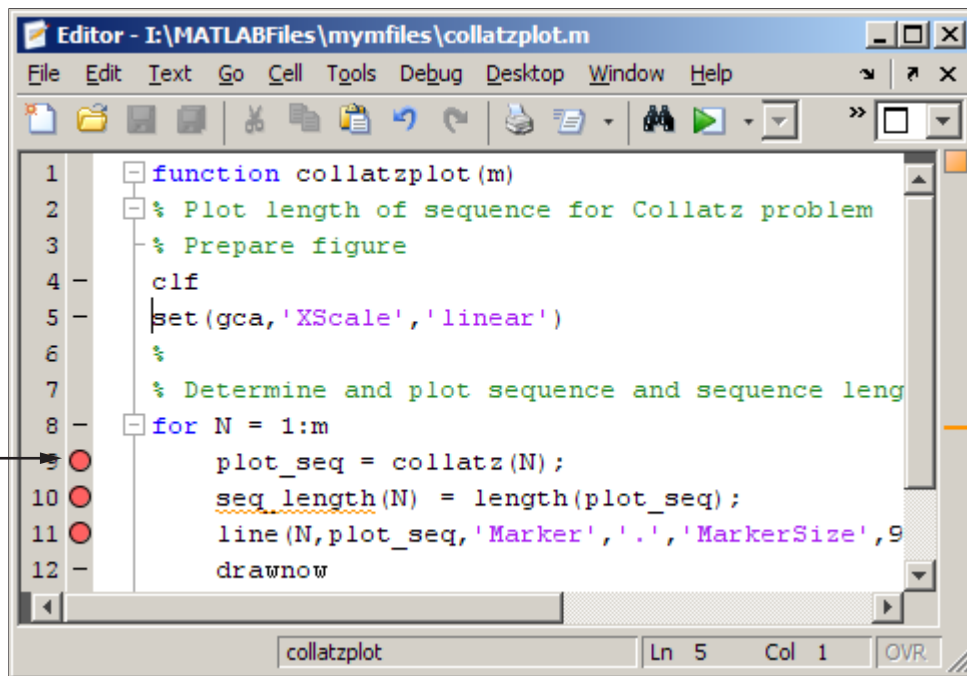
Valid (Red) and Invalid (Gray) Breakpoints. Red breakpoints are valid standard breakpoints. If breakpoints are gray, they are not valid.

When breakpoints are gray they are not valid.

In this example, it is because the file has not been saved since changes were made to it.

Save the file to make the breakpoints valid (red)

Breakpoints are gray for either of these reasons:

- The file has not been saved since changes were made to it. Save the file to make breakpoints valid. The gray breakpoints become red, indicating they are now valid. Any gray breakpoints that were entered at invalid breakpoint lines automatically move to the next valid breakpoint line with the successful file save.

- There is a syntax error in the file. When you set a breakpoint, an error message appears indicating where the syntax error is. Fix the syntax error and save the file to make breakpoints valid.

Function Alternative for Setting Breakpoints

To set a breakpoint using the debugging functions, use `dbstop`. For the example, type:

```
dbstop in collatzplot at 9
dbstop in collatzplot at 10
dbstop in collatzplot at 11
```

Some useful related functions are

- `dbtype` — Lists the M-file with line numbers in the Command Window.

- `dbstatus` — Lists breakpoints.

Running an M-File with Breakpoints

After setting breakpoints, run the M-file from the Command Window or the Editor.

Running the Example

For the example, run `collatzplot` for the simple input value, 3, by typing in the Command Window

```
collatzplot(3)
```

The example, `collatzplot`, requires an input argument and therefore runs only from the Command Window or from a run configuration with a value specified.

Results of Running an M-File Containing Breakpoints

Running the M-file results in the following:

- The prompt in the Command Window changes to

   ```
   K>>
   ```

   indicating that MATLAB is in debug mode.

- The program pauses at the first breakpoint. This means that line will be executed when you continue. The pause is indicated in the Editor by the

green arrow just to the right of the breakpoint, which in the example, is line 9 of `collatzplot` as shown here.

```
9  plot_seq = collatz(N);
```

If you use debugging functions from the Command Window, the line at which you are paused is displayed in the Command Window. For the example, it would show

```
9        plot_seq = collatz(N);
```

- The function displayed in the **Stack** field on the toolbar changes to reflect the current function (sometimes referred to as the caller or calling workspace). The call stack includes subfunctions as well as called functions. If you use debugging functions from the Command Window, use `dbstack` to view the current call stack.

- If the file you are running is not in the current directory or a directory on the search path, you are prompted to either add the directory to the path or change the current directory.

In debug mode, you can set breakpoints, step through programs, examine variables, and run other functions.

Note that MATLAB software could become nonresponsive if it stops at a breakpoint while displaying a modal dialog box or figure that your M-file creates. In that event, use **Ctrl+C** to go the MATLAB prompt.

Stepping Through an M-File

While debugging, you can step through an M-file, pausing at points where you want to examine values.

Use the step buttons or the step items in the **Debug** menu of the Editor or desktop, or use the equivalent functions.

6-163

Toolbar Button	Debug Menu Item	Description	Function Alternative
▶	**Run** *m-file* or **Run Configuration for** *m-file*	Commence execution of M-file and run until completion or until a breakpoint is encountered. The **Run Configurations for** *m-file* menu item provides a submenu that enables you to select a particular run configuration or to edit the run configurations for the M-file. If you choose **Run** *m-file*, MATLAB uses the default run configuration.	None
None	**Go Until Cursor**	Continue execution of M-file until the line where the cursor is positioned. Also available on the context menu.	None
▤	**Step**	Execute the current line of the M-file.	dbstep
▤	**Step In**	Execute the current line of the M-file and, if the line is a call to another function, step into that function.	dbstep in
▤	**Continue**	Resume execution of M-file until completion or until another breakpoint is encountered.	dbcont
▤	**Step Out**	After stepping in, run the rest of the called function or subfunction, leave the called function, and pause.	dbstep out

Continue Running in the Example

In the example, collatzplot is paused at line 9. Because the problem results are correct for N/n = 1, continue running until N/n = 2. Press the Continue

button ⬇️ three times to move through the breakpoints at lines 9, 10, and 11. Now the program is again paused at the breakpoint at line 9.

Stepping In to Called Function in the Example

Now that collatzplot is paused at line 9 during the second iteration, use the Step In button 🔼 or type dbstep in in the Command Window to step into collatz and walk through that M-file. Stepping into line 9 of collatzplot goes to line 9 of collatz. If collatz is not open in the Editor, it automatically opens if you have selected **Debug > Open M-Files When Debugging**.

The pause indicator at line 9 of collatzplot changes to a hollow arrow ⇨, indicating that MATLAB control is now in a subfunction called from the main program. The call stack shows that the current function is now collatz.

In the called function, collatz in the example, you can do the same things you can do in the main (calling) function—set breakpoints, run, step through, and examine values.

Examining Values

While the program is paused, you can view the value of any variable currently in the workspace. Examine values when you want to see whether a line of code has produced the expected result or not. If the result is as expected, continue running or step to the next line. If the result is not as expected, then that line, or a previous line, contains an error. Use the following methods to examine values:

- "Selecting the Workspace" on page 6-166
- "Viewing Values as Data Tips in the Editor" on page 6-166
- "Viewing Values in the Command Window" on page 6-167
- "Viewing Values in the Workspace Browser and Variable Editor" on page 6-168
- "Evaluating a Selection" on page 6-169
- "Examining Values in the Example" on page 6-169
- "Problems Viewing Variable Values from Parent Workspace" on page 6-170

Many of these methods are used in "Examining Values in the Example" on page 6-169.

Selecting the Workspace

Variables assigned through the Command Window and created using scripts are considered to be in the base workspace. Variables created in a function belong to their own function workspace. To examine a variable, you must first select its workspace. When you run a program, the current workspace is shown in the **Stack** field. To examine values that are part of another workspace for a currently running function or for the base workspace, first select that workspace from the list in the **Stack** field.

If you use debugging functions from the Command Window:

- To display the call stack, use `dbstack`.

- To change to a different workspace, use `dbup` and `dbdown`.

- To list the variables in the current workspace, use `who` or `whos`.

Workspace in the Example. At line 9 of `collatzplot`, you stepped in, and the current line is 9 in `collatz`. The **Stack** shows that `collatz` is the current workspace.

Viewing Values as Data Tips in the Editor

In the Editor, position the mouse pointer to the left of a variable. Its current value appears—this is called a data tip, which is like a ToolTip for data. The data tip stays in view until you move the pointer. If you have trouble getting the data tip to appear, click in the line containing the variable and then move the pointer next to the variable.

A related function is `datatipinfo`.

Data Tips in the Example. Position the mouse pointer over `n` in line 9 of `collatz`. The data tip shows that `n = 2`, as expected.

Hold the mouse pointer over a variable.

Its current value temporarily displays as a data tip.

Viewing Values in the Command Window

You can examine values while in debug mode at the K>> prompt. To see the variables currently in the workspace, use who. Type a variable name in the Command Window and it displays the variable's current value. For the example, to see the value of n, type

```
n
```

The Command Window displays the expected result

```
n =
  2
```

and displays the debug prompt, K>>.

Viewing Values in the Workspace Browser and Variable Editor

You can view the value of variables in the **Value** column of the Workspace browser. The Workspace browser displays all variables in the current workspace. Use the **Stack** in the Workspace browser to change to another workspace and view its variables.
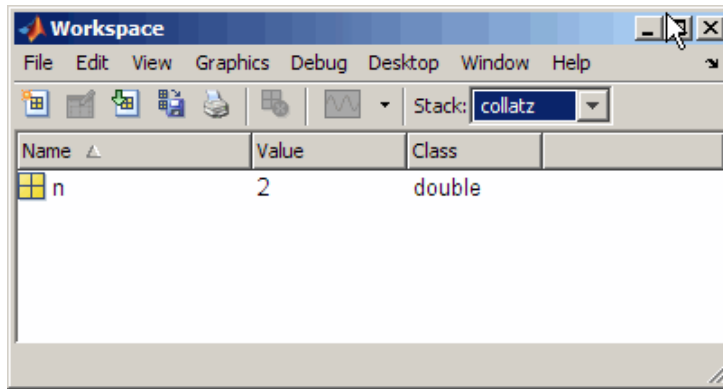
The **Value** column does not show all details for all variables. To see details, double-click a variable in the Workspace browser. The Variable Editor opens, displaying the content for that variable. You can open the Variable Editor directly for a variable using openvar.

To see the value of n in the Variable Editor for the example, type

```
openvar n
```

and the Variable Editor opens, showing that n = 2 as expected.
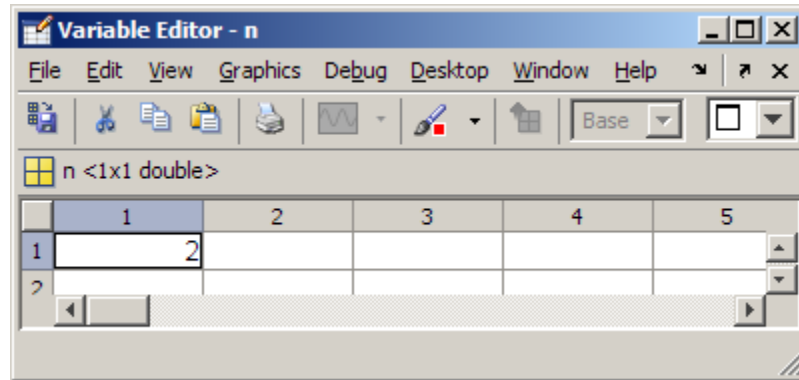
Evaluating a Selection

Select a variable or equation in an M-file in the Editor. Right-click and select **Evaluate Selection** from the context menu (for a single-button mouse, use **Ctrl**+ click). The Command Window displays the value of the variable or equation. You cannot evaluate a selection while MATLAB is busy, for example, running an M-file.

Examining Values in the Example

Step from line 9 through line 13 in `collatz`. Step again, and the pause indicator jumps to line 17, just after the `if` loop, as expected. Step again, to line 18, check the value of `sequence` in line 17 and see that the array is

```
2   1
```

as expected for `n = 2`. Step again, which moves the pause indicator from line 18 to line 11. At line 11, step again. Because `next_value` is now 1, the `while` loop ends. The pause indicator is at line 11 and appears as a green down arrow ⬇. This indicates that processing in the called function is complete and program control will return to the calling program. Step again from line 11 in `collatz` and execution is now paused at line 9 in `collatzplot`.

Note that instead of stepping through `collatz`, the called function, as was just done in this example, you can step out from a called function back to the calling function, which automatically runs the rest of the called function and returns to the next line in the calling function. To step out, use the Step Out button or type `dbstep out` in the Command Window.

In `collatzplot`, step again to advance to line 10, and then line 11. The variable `seq_length` in line 10 is a vector with the elements:

```
1   2
```

which is correct.

Finally, step again to advance to line 12. Examining the values in line 11, N = 2 as expected, but the second variable, `plot_seq`, has two values, where only one value is expected. While the value for `plot_seq` is as expected:

```
2   1
```

it is the incorrect variable for plotting. Instead, `seq_length(N)` should be plotted.

Problems Viewing Variable Values from Parent Workspace

Sometimes, if you set a breakpoint in a function, and then attempt to view the value of a variable in the parent workspace using the `dbup` command, the value of the variable is currently under construction. Therefore, the value is not available. This is true whether you view the value by specifying the `dbup` command in the Command Window or by using the **Stack** field on the Editor toolbar.

In such cases, MATLAB returns the following message, where *x* is the variable for which you are trying to examine the value:

```
K>> x
??? Reference to a called function result under construction x.
```

For example, suppose you have code such as the following:

```
x = collatz(x);
```

MATLAB detects that the evaluation of `collatz(x)` replaces the input variable, *x*. To optimize memory use, MATLAB overwrites the memory that *x* currently occupies to hold a new value for *x*. When you request the value of *x*, and it is under construction, its value is not available, and MATLAB displays the error message.

Correcting Problems and Ending Debugging

These are some of the ways to correct problems and end the debugging session:

- "Changing Values and Checking Results" on page 6-171
- "Ending Debugging" on page 6-171
- "Disabling and Clearing Breakpoints" on page 6-172
- "Saving Breakpoints" on page 6-174
- "Correcting an M-File" on page 6-174
- "Completing the Example" on page 6-174
- "Running Sections in M-Files That Have Unsaved Changes" on page 6-177

Many of these features are used in "Completing the Example" on page 6-174.

Changing Values and Checking Results

While debugging, you can change the value of a variable in the current workspace to see if the new value produces expected results. While the program is paused, assign a new value to the variable in the Command Window, Workspace browser, or Variable Editor. Then continue running or stepping through the program. If the new value does not produce the expected results, the program has a different problem.

Ending Debugging

After identifying a problem, end the debugging session. You must end a debugging session if you want to change and save an M-file to correct a problem, or if you want to run other functions in MATLAB.

> **Note** It is recommended that you quit debug mode before editing an M-file. If you edit an M-file while in debug mode, you can get unexpected results when you run the file. If you do edit an M-file while in debug mode, breakpoints turn gray, indicating that results might not be reliable. See "Valid (Red) and Invalid (Gray) Breakpoints" on page 6-161 for details.
>
> If you attempt to save an edited M-file while in debug mode, a dialog box opens allowing you to exit debug mode and save the file.

To end debugging, click the Exit debug mode button ⧉ or select **Exit Debug Mode** from the **Debug** menu.

You can instead use the function dbquit or the **Shift+F5** keyboard shortcut to end debugging.

After quitting debugging, pause indicators in the Editor display no longer appear, and the normal prompt >> appears in the Command Window instead of the debugging prompt, K>>. You can no longer access the call stack.

Disabling and Clearing Breakpoints

Disable a breakpoint to temporarily ignore it. Clear a breakpoint to remove it.

Disabling and Enabling Breakpoints. You can disable selected breakpoints so the program temporarily ignores them and runs uninterrupted, for example, after you think you identified and corrected a problem. This is especially useful for conditional breakpoints—see "Conditional Breakpoints" on page 6-178.

To disable a breakpoint, right-click the breakpoint icon and select **Disable Breakpoint** from the context menu, or click anywhere in a line and select **Enable/Disable Breakpoint** from the **Debug** or context menu. You can also disable a conditional breakpoint by clicking the breakpoint icon. This puts an X through the breakpoint icon as shown here.

```
9 🗵         plot_seq = collatz(N);
```

When you run dbstatus, the resulting message for a disabled breakpoint is

```
Breakpoint on line 9 has conditional expression 'false'.
```

After disabling a breakpoint, you can enable it to make it active again, or clear it. To enable it, right-click the breakpoint icon and select **Enable Breakpoint** from the context menu, or click anywhere in a line and select **Enable/Disable Breakpoint** from the **Breakpoints** or context menu. The X no longer appears on the breakpoint icon and program execution will pause at that line.

Clearing (Removing) Breakpoints. All breakpoints remain in a file until you clear (remove) them or until they are automatically cleared. Clear a breakpoint after determining that a line of code is not causing a problem.

To clear a breakpoint in the Editor:

- Click anywhere in a line that has a breakpoint and select **Set/Clear Breakpoint** from the **Debug** or context menu.

- Click a standard breakpoint icon, or a disabled conditional breakpoint icon.

- Use the `dbclear in m-file at lineno` command in the Command Window. For the example, clear the breakpoint at line 9 in `collatzplot` by typing:

  ```
  dbclear in collatzplot at 9
  ```

To clear all breakpoints in all files:

- Select **Debug > Clear Breakpoints in All Files** on the toolbar.

- Click the Clear breakpoints in all files button

- Use `dbclear all` in the Command Window.

  For the example, clear all of the breakpoints in `collatzplot` by typing

  ```
  dbclear all in collatzplot
  ```

Breakpoints are automatically cleared when you

- End the MATLAB session

- Clear the M-file using `clear name` or `clear all`

6-173

Note When `clear name` or `clear all` is in a statement in an M-file that you are debugging, it clears the breakpoints.

Saving Breakpoints

You can use the `s=dbstatus` syntax and then save `s` to save the current breakpoints to a MAT-file. At a later time, you can load `s` and restore the breakpoints using `dbstop(s)`. For more information, including an example, see the `dbstatus` reference page.

Correcting an M-File

To correct a problem in an M-file,

1 Quit debugging.

Do not make changes to an M-file while MATLAB is in debug mode. If you do edit an M-file while in debug mode, breakpoints turn gray, indicating that results might not be reliable. See "Valid (Red) and Invalid (Gray) Breakpoints" on page 6-161 for details.

2 Make changes to the M-file.

3 Save the M-file.

4 Set, disable, or clear breakpoints, as appropriate.

5 Run the M-file again to be sure it produces the expected results.

Completing the Example

To correct the problem in the example, do the following:

1 End the debugging session. One way to do this is to select **Exit Debug Mode** from the **Debug** menu.

2 In `collatzplot.m` line 11, change the string `plot_seq` to `seq_length(N)` and save the file.

3 Clear the breakpoints in `collatzplot.m`. One way to do this is by typing

```
dbclear all in collatzplot
```

in the Command Window.

4 Run `collatzplot` for m = 3 by typing

```
collatzplot(3)
```

in the Command Window.

5 Verify the result. The figure shows that the length of the Collatz series is 1 when n = 1, 2 when n = 2, and 8 when n = 3, as expected.
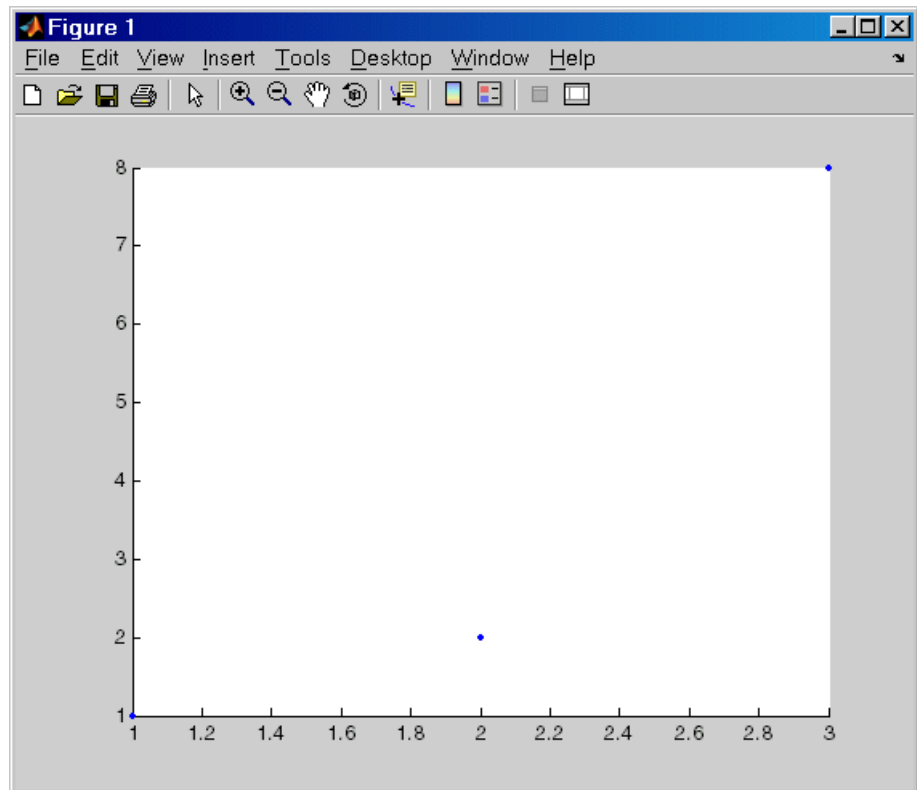
6 Test the function for a slightly larger value of m, such as 6, to be sure the results are still accurate. To make it easier to verify `collatzplot` for m = 6 as well as the results for `collatz`, add this line at the end of `collatz.m`

```
sequence
```

which displays the series in the Command Window. The results for when n = 6 are

```
sequence =

     6     3    10     5    16     8     4     2     1
```

Then run `collatzplot` for m = 6 by typing

```
collatzplot(6)
```

7 To make debugging easier, you ran `collatzplot` for a small value of m. Now that you know it works correctly, run `collatzplot` for a larger value to produce more interesting results. Before doing so, you might want to disable output for the line you just added in step 6, line 19 of `collatz.m`, by adding a semicolon to the end of the line so it appears as

```
sequence;
```

Then run

```
collatzplot(500)
```

The following figure shows the lengths of the Collatz series for n = 1 through n = 500.
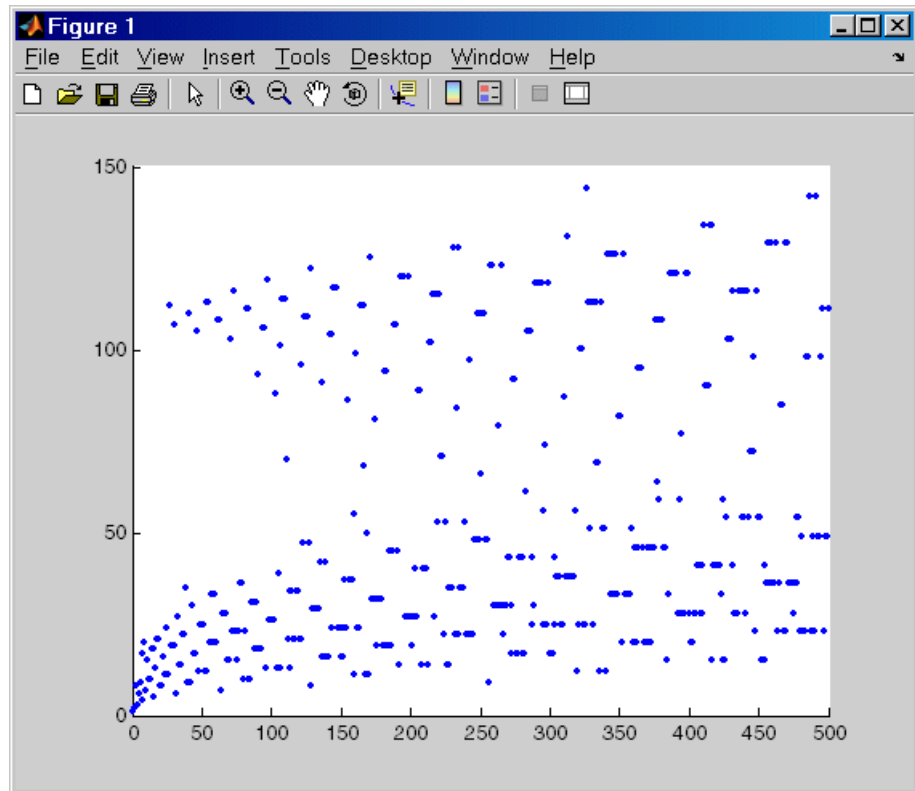
Running Sections in M-Files That Have Unsaved Changes

It is a good practice to make changes to an M-file after you quit debugging, and to save the changes and then run the file. Otherwise, you might get unexpected results. But there are situations where you might want to experiment during debugging, to make a change to a part of the file that has not yet run, and then run the remainder of the file without saving the change.

To do this, while stopped at a breakpoint, make a change to a part of the file that has not yet run. Breakpoints will turn gray, indicating they are invalid. Then select all of the code after the breakpoint, right-click, and

choose **Evaluate Selection** from the context menu. You can also use cell mode to do this.

Conditional Breakpoints

Set conditional breakpoints to cause MATLAB to stop at a specified line in a file only when the specified condition is met. One particularly good use for conditional breakpoints is when you want to examine results after a certain number of iterations in a loop. For example, set a breakpoint at line 10 in collatzplot, specifying that MATLAB should stop only if N is greater than or equal to 2. This section covers the following topics:

- "Setting Conditional Breakpoints" on page 6-178
- "Copying, Modifying, Disabling, and Clearing Conditional Breakpoints" on page 6-180
- "Function Alternative for Conditional Breakpoints" on page 6-180

Setting Conditional Breakpoints

To set a conditional breakpoint, follow these steps:

1 Click in the line where you want to set the conditional breakpoint. Then select **Set/Modify Conditional Breakpoint** from the **Debug** or context menu. If a standard breakpoint already exists at that line, use this same method to make it conditional.

The **MATLAB Editor** conditional breakpoint dialog box opens as shown in this example.
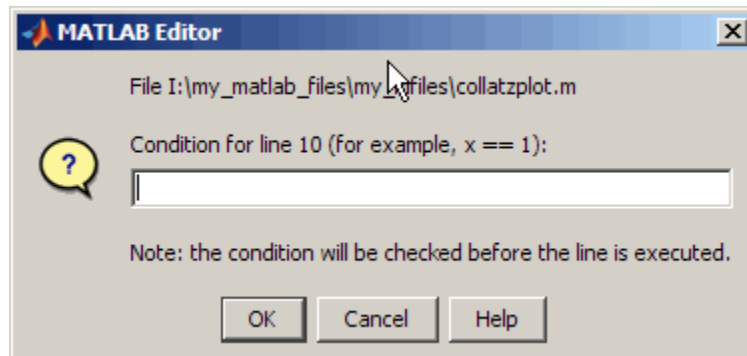
2 Type a condition in the dialog box, where a condition is any legal MATLAB expression that returns a logical scalar value. Click **OK**. As noted in the dialog box, the condition is evaluated before running the line. For the example, at line 9 in collatzplot, enter

```
N>=2
```

as the condition. A yellow breakpoint icon (indicating the breakpoint is conditional) appears in the breakpoint alley at that line.

Conditional breakpoint (yellow)
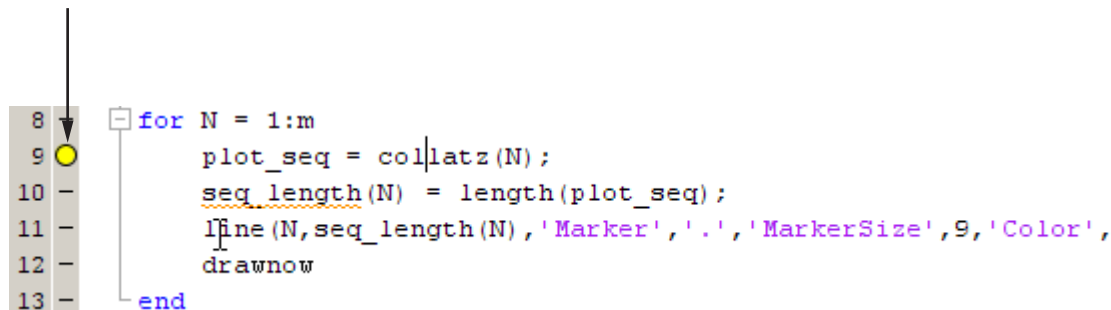
When you run the file, MATLAB software enters debug mode and pauses at the line only when the condition is met. In the collatzplot example, MATLAB runs through the for loop once and pauses on the second iteration

at line 9 when N is 2. If you continue executing, MATLAB pauses again at line 9 on the third iteration when N is 3.

Copying, Modifying, Disabling, and Clearing Conditional Breakpoints

To copy a conditional breakpoint, right-click the icon in the breakpoint alley and select **Copy** from the context menu. Then right-click in the breakpoint alley at the line where you want to paste the conditional breakpoint and select **Paste** from the context menu.

Modify the condition for the breakpoint in the current line by selecting **Set/Modify Conditional Breakpoint** from the **Debug** or context menu.

Click a conditional breakpoint icon to disable it. Click a disabled conditional breakpoint to clear it.

Function Alternative for Conditional Breakpoints

Use the dbstop function with appropriate arguments to set conditional breakpoints from the Command Window, and use dbclear to clear them. Use dbstatus to view the breakpoints currently set, including any conditions, which are listed in the expression field. If no condition exists, the value in the expression field is [   ] (empty). For details, see the function reference pages: dbstop, dbclear, and dbstatus.

Breakpoints in Anonymous Functions

There can be multiple breakpoints in an M-file line that contains anonymous functions. There can be a breakpoint for the line itself (MATLAB software stops at the start of the line), as well as a breakpoint for each anonymous function in that line. When you add a breakpoint to a line containing an anonymous function, the Editor asks exactly where in the line you want to add the breakpoint. If there is more than one breakpoint in a line, the breakpoint icon is blue ●.

When there are multiple breakpoints set on a line, the icon is always blue, regardless of the status of any of the breakpoints on the line. Position the mouse on the icon and a ToolTip displays information about all breakpoints in that line.

To perform a breakpoint action for a line that can contain multiple breakpoints, such as **Clear Breakpoint**, right-click the breakpoint alley at that line and select the action. MATLAB prompts you to specify the exact breakpoint on which to act in that line.

When you set a breakpoint in an anonymous function, MATLAB stops when the anonymous function is called. The following illustration shows the Editor when you set a breakpoint in the anonymous function sqr in line 2, and then run the file. MATLAB stops when it runs sqr in line 4. After you continue execution, MATLAB stops again when it runs sqr the second time in line 4. Note that the **Stack** display shows the anonymous function.

Breakpoint set in anonymous function sqr.

MATLAB stops when it runs sqr.
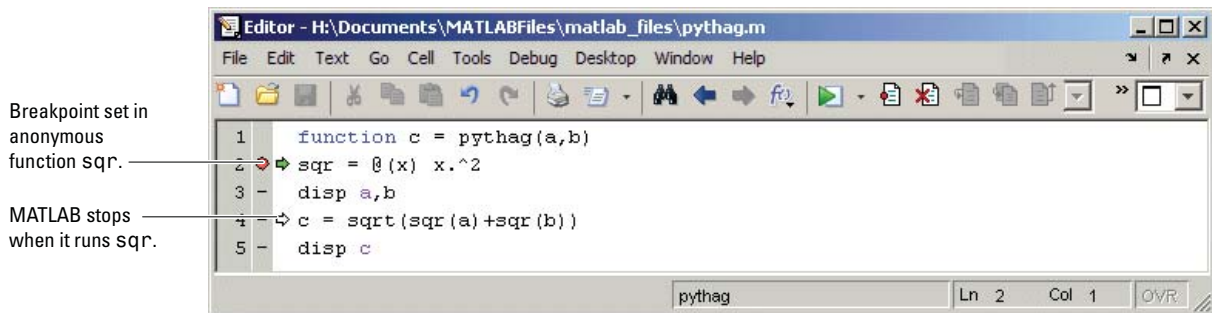
Error Breakpoints

Set error breakpoints to stop program execution and enter debug mode when MATLAB encounters a problem. Unlike standard and conditional breakpoints, you do not set these breakpoints at a specific line in a specific file. Rather, once set, MATLAB stops at any line in any file when the error condition specified using the error breakpoint occurs. MATLAB then enters debug mode and opens the file containing the error, with the pause indicator at the line containing the error. Files open only when the you select **Debug > Open M-Files** . Error breakpoints remain in effect until you clear them or until you end the MATLAB session. You can set error breakpoints from the **Debug** menu in any desktop tool. This section covers the following topics:

- "Setting Error Breakpoints" on page 6-182
- "Error Breakpoint Types and Options" on page 6-182

• "Function Alternative for Error Breakpoints" on page 6-184

Setting Error Breakpoints

To set error breakpoints, select **Debug > Stop if Errors/Warnings**. In the resulting **Stop if Errors/Warnings for All Files** dialog box, specify error breakpoints on all appropriate tabs and click **OK**. To clear error breakpoints, select the **Never stop if ...** option for all appropriate tabs and click **OK**.
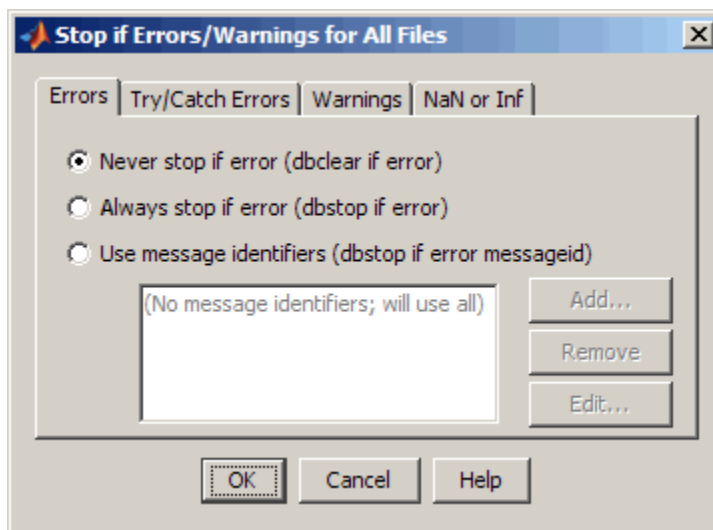
For example, to pause execution when a warning occurs, select the **Warnings** tab, and from it select **Always stop if warning**, then click **OK**. When you run an M-file and MATLAB produces a warning, execution pauses, MATLAB enters debug mode, and the file opens in the Editor at the line that produced the warning. To remove the warning breakpoint, select **Never stop if warning** in the **Warnings** tab and click **OK**.

Error Breakpoint Types and Options

The four basic types of error breakpoints you can set are **Errors**, **Try/Catch Errors**, **Warnings**, and **NaN or Inf**. Select the **Always stop if ...** option for each tab to set that type of breakpoint. Select the **Use message identifiers ...** option to limit each type of error breakpoint (except NaN or Inf) so that execution stops only for specific errors.

Errors. When an error occurs, execution stops, unless the error is in a `try...catch` block. MATLAB enters debug mode and opens the M-file to the line in the `try` portion of the block that produced the error. You cannot resume execution.

Try/Catch Errors. When an error occurs in a `try...catch` block, execution pauses. MATLAB enters debug mode and opens the M-file to the line that produced the error. You can resume execution or use debugging features.

Warnings. When a warning is produced, MATLAB pauses, enters debug mode, and opens the M-file, paused at the line that produced the warning. You can resume execution or use debugging features.

NaN or Inf. When an operator, function call, or scalar assignment produces a `NaN` (not-a-number) or `Inf` (infinite) value, MATLAB pauses, enters debug mode, and opens the M-file, paused immediately after the line that encountered the value. You can resume execution or use debugging features.

Use Message Identifiers. Execution stops only when MATLAB encounters one of the specified errors. This option is not available for the **NaN or Inf** type of error breakpoint. To use this feature:

1 Select the **Errors**, **Try/Catch Errors,** or **Warnings** tab.

2 Select the **Use Message Identifiers** option.

3 Click **Add**.

4 In the resulting **Add Message Identifier** dialog box, supply the message identifier of the specific error you want to stop for, where the identifier is of the form `component:message`, and click **OK**.

5 The message identifier you added appears in the **Stop If Errors/Warnings for All Files** dialog box, where you click **OK**.

   You can add multiple message identifiers, and edit or remove them.

One way to obtain an error message identifier generated by a MATLAB function for example, is to produce the error, and then run the `lasterror` function. MATLAB returns the error message and identifier. Copy the identifier from the Command Window output and paste it into the **Add**

Message Identifier dialog box. An example of an error message identifier is MATLAB:UndefinedFunction. Similarly, to obtain a warning message identifier, produce the warning and then run [m,id] = lastwarn; MATLAB returns the last warning identifier to id. An example of a warning message identifier is MATLAB:divideByZero.

Function Alternative for Error Breakpoints

The function equivalent for each option appears in the **Stop if Errors/Warnings for All Files** dialog box. For example, the function equivalent for **Always stop if error** is dbstop if error. Use the dbstop function with appropriate arguments to set error breakpoints from the Command Window, and use dbclear to clear them. Use dbstatus to view the error breakpoints currently set. Error breakpoints are listed in the cond field and message identifiers for breakpoints are listed in the identifier field of the dbstatus output.

Using Cells for Rapid Code Iteration and Publishing Results

In this section...
"What Are Cells?" on page 6-185
"Rapid Code Iteration Overview" on page 6-186
"Defining Cells" on page 6-188
"Understanding Nested Cells" on page 6-197
"Evaluating M-File Cells" on page 6-208

What Are Cells?

M-files often have a natural structure consisting of multiple sections. Especially for larger files, you typically focus efforts on a single section at a time, working with the code in just that section. Similarly, when conveying information about your M-files to others, often you describe the sections of the code. To facilitate these processes, use M-file *cells*, where *cell* refers to a section of code. A cell contains the contiguous lines of code that you want to evaluate as a whole in an M-file script. A cell has boundaries to define its start and end. Because cell features operate on cells, it is important to understand how you define boundaries explicitly, how MATLAB defines boundaries implicitly, and how implicitly and explicitly defined cell boundaries interact to create cells, as described in "Defining Cells" on page 6-188

Specifically, MATLAB software uses cells for:

- Rapid code iteration in the Editor — This makes the experimental phase of your work with M-file scripts easier. The next section, "Rapid Code Iteration Overview" on page 6-186, outlines the process, and is followed by details for defining, evaluating, and modifying values in cells.

- Publishing M-files — This allows you to include code and results in a presentation format such as HTML. Publishing using cells also requires you to define cells. You can make use of the cell navigation and evaluation you specify for rapid code iteration or define and use cells explicitly for publishing. See Chapter 8, "Publishing M-Files" for complete details.

Rapid Code Iteration Overview

When working with an M-file, you often experiment with your code—modifying it, testing it, and updating it—until you have an M-file that does what you want. For example:

- Suppose you have code that plots data. You might break the code into two cells: the code in the first cell creates the basic results, while the code in the second cell labels the plot. The two cells allow you to experiment with the plot of the data first, and then when that is final, change the plot properties to affect the style of presentation. This scenario is presented in "Example of Evaluating Cells" on page 6-211.

- Suppose you have an two images that you want to add together, and then display the results. As part of this algorithm, you want to adjust the brightness of the second image before adding it to the first image. You can read in that images, tweak the second image's brightness, read it in again, adjust its brightness, and so on using the cell mode toolbar buttons until you see the brightness you want. There is no need to save the M-file between adjustments. If you have an active Internet connection, you can watch the Rapid Code Iteration Using Cells video demo that illustrates this example and more, including an overview of the major rapid code iteration features.

Use the MATLAB Editor cell features with M-file scripts to facilitate this process. You also can use cell features with function M-files, but there are some restrictions—see "Using Cells in Function M-Files" on page 6-210.

Note Cell mode is supported for use with M-files only. It is not intended for use with plain text files. When used with plain text files, results are unpredictable.

This is the overall process of using cells for rapid code iteration:

1 In the MATLAB Editor, select **Cell > Enable Cell Mode**. Items in the **Cell** menu become selectable. The cell toolbar appears, unless you had previously hidden it. With cell mode enabled, hide or show the toolbar by right-clicking in the Editor menu bar or toolbars and selecting **Cell Toolbar** from the context menu.

2 Define the boundaries of the cells in an M-file script using cell features. Cells are denoted by a specialized comment syntax, %%. For details, see "Defining Cells" on page 6-188.

3 After you define the cells, use cell features to navigate quickly from cell to cell in your file, evaluate the code in a cell in the base workspace, and view the results. To facilitate experimentation, use cell features to modify values in cells, and then reevaluate them to see how different values impact the result. For details, see "Evaluating M-File Cells" on page 6-208.

Cell features.

Cell toolbar.

Current cell is highlighted.

Use display preference to show horizontal lines between cells.
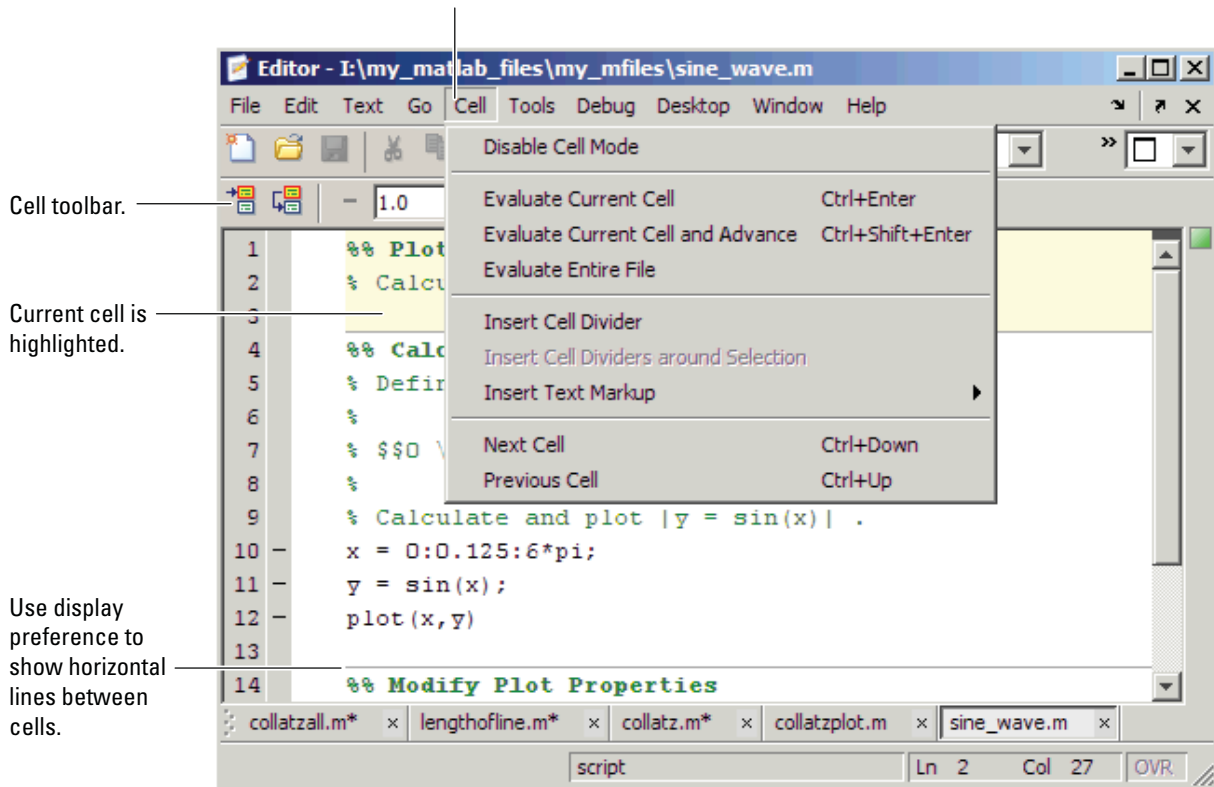
Defining Cells

You define cell boundaries explicitly by inserting a line that begins with a cell break (also referred to as a cell divider), which is two percent sign characters (%%). White space can precede these two characters, and text can follow them, as long as there is white space between the %% characters and the text. For details, see "Defining Cell Boundaries Explicitly" on page 6-189.

MATLAB defines implicit cell boundaries in a code block only when you specify one or more explicit cell breaks within that code block. MATLAB defines implicit cell breaks as follows:

- MATLAB defines implicit cell breaks at the top and bottom of the file, to create an implicit cell that contains the entire file. However, the Editor does not highlight the resulting cell, which encloses the entire file, unless you add one or more explicit cell breaks to the file.

- If you define an explicit cell break in a function, MATLAB defines implicit cell breaks at the function declaration and at the function end statement.

  The resulting cells are nested within the full file cell. Note that if you do not end the function with an explicit end statement, MATLAB behaves as though the end of the function occurs immediately before the start of the next function.

- If you define an explicit cell break within a language construct (such as an if statement, a while statement, and so on), MATLAB defines implicit cell breaks at the lines containing the start and end of the language construct.

  The resulting cells are nested within the full file cell, and the function in which the code block occurs, if any.

If an implicit cell break and an explicit cell break occur on the same line, they collapse into one explicit cell break. For more information on nested cells, see "Understanding Nested Cells" on page 6-197.

This section includes the following topics:

- "Defining Cell Boundaries Explicitly" on page 6-189

- "Creating Titles for Cells" on page 6-190

- "Highlighting Cells" on page 6-190

Defining Cell Boundaries Explicitly

To define cell boundaries explicitly, you must insert cell breaks. Follow these steps:

1 Ensure that cell mode is enabled. (See "Rapid Code Iteration Overview" on page 6-186.)

2 Optionally, to help you distinguish cells from each other, do one or both of the following:

- Include a faint gray horizontal line (rule) above each cell to help you distinguish the cells from each other.

  Select **File > Preferences > Editor/Debugger > Display**, and then in **Cell display options**, select **Show lines between cells**.

  The horizontal lines do not appear in the M-file when you print it.

- Set a color to indicate the current cell.

  Select **File > Preferences > Editor/Debugger > Display**. Then, in **Cell display options**, select **Highlight cells**, and then select the color that you want. By default, MATLAB highlights the current cell in yellow.

  The current cell is the cell where you have placed the cursor. Like the lines between cells, highlighting helps you distinguish the cells from each other.

3 Do one of the following to insert the cell breaks:

- Position the cursor just before the line at which you want to start the cell and select **Cell > Insert Cell Break** .

- Click the Insert cell break button ⅍.

- Enter two percent signs (%%) at the start of the line where you want to begin the new cell.

- Select the lines of code you want in a cell, and then select **Cell > Insert Cell Breaks Around Selection**

Note Program control statements, such as if ... end, must be contained within a single cell. You cannot insert a cell break between the if and the end statements.

You can define a cell at the start of a new empty file, enter code for the cell, define the start of the next cell, enter its code, and so on. Redefine cells by defining new cells, removing existing cell boundaries, and moving lines of code.

Creating Titles for Cells

The Editor emphasizes the special meaning of the start of a cell by making any text following the percent signs appear bold. The text on the %% line is called the *cell title*. Including text in cell titles is optional, however, it improves the readability of the file and is used for cell publishing features.

To create a cell title, after the %% characters that specify a cell break, type a space, followed by a description of the cell.

Highlighting Cells

When the cursor is positioned in any line within a cell, the Editor highlights the entire cell that contains that line with a yellow background, by default. This identifies it as the *current cell*. For example, it is used when you select the **Evaluate Current Cell** option on the **Cell** menu.

Turning Off Cell Highlighting.

1 Select **File > Preferences > Editor/Debugger > Display**.

2 Under **Cell display options**, deselect **Highlight cells**.

Setting a Color for Cell Highlighting.

1 Select **File > Preferences > Editor/Debugger > Display**.

2 Under **Cell display options**, select **Highlight cells**.

3 Click the down arrow next to the color block beside **Highlight cells**, and then select the color with which you want to highlight cells.
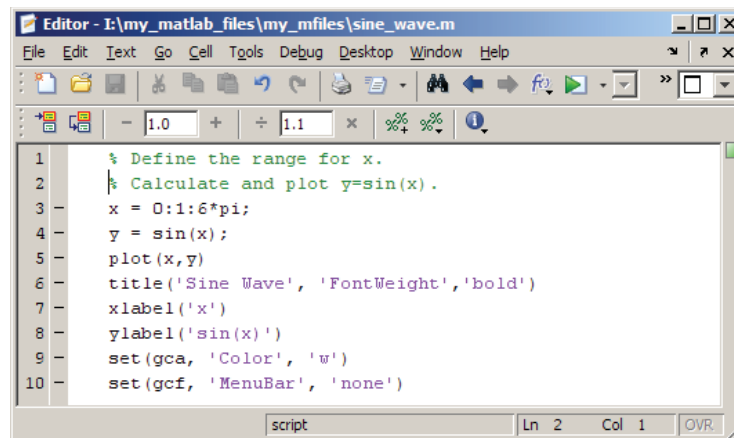
Example of Defining Cells

This example defines two cells for a simple M-file called `sine_wave.m`, shown in the following code and figure.

The steps that follow insert a cell breaks into the code to create two cells. The code in the first cell creates the basic results, while the second labels the plot. The two cells allow you to experiment with the plot of the data first, and then when that is final, change the plot properties to affect the style of presentation.

```
% Define the range for x.
% Calculate and plot y = sin(x).
x = 0:1:6*pi;
y = sin(x);
plot(x,y)
title('Sine Wave','FontWeight','bold')
xlabel('x')
ylabel('sin(x)')
set(gca,'Color','w')
set(gcf, 'MenuBar', 'none')
```

M-file before defining cells.

1 Select **Cell > Enable Cell Mode**, if it is not already enabled.

2 Position the cursor at the start of the first line. Select **Cell > Insert Cell Break**.

The Editor inserts %% as the first line and moves the rest of the file down one line. All lines appear highlighted in yellow, indicating that the entire file is a single cell, assuming you have that display preference for cells selected.

3 After the %%, type a space, and then enter a cell title.

```
%% Calculate and Plot Sine Wave
```

4 Position the cursor at the start of line 7, `title...`, and then select **Cell > Insert Cell Break**.

The Editor inserts a line containing only %% at line 7 and moves the remaining lines down one line. A horizontal line that helps you distinguish the two cells appears above the %% line, assuming you have that display preference for cells selected. Lines 7 through 12 appear highlighted in yellow, indicating they comprise the current cell.

5 On line 7, type a space after the %%, and then enter a cell title for the new cell.

```
%% Modify Plot Properties
```
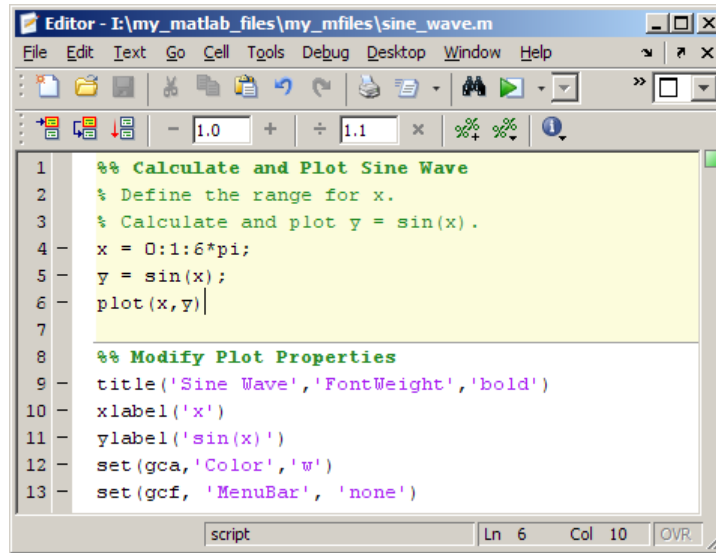
Save the file. The file appears as shown in this figure.

Use preferences to show highlighting of current cell and horizontal lines between cells
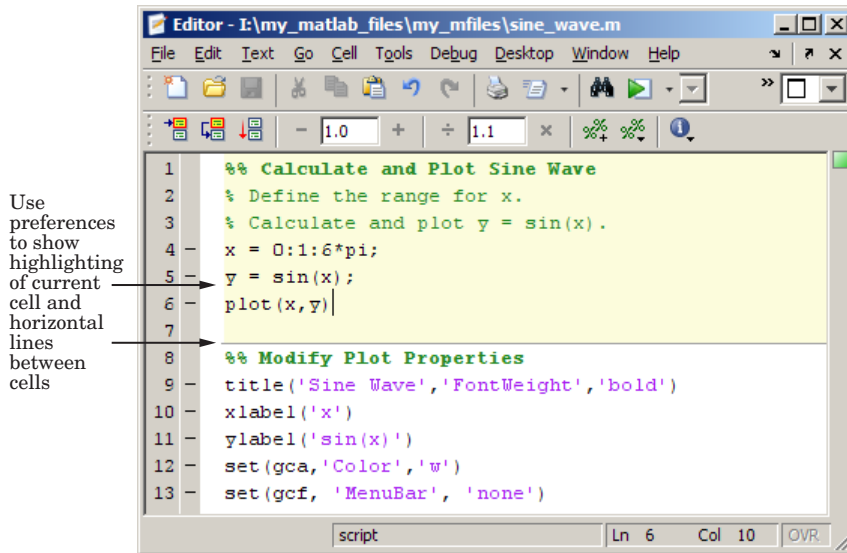
Fixing Cell Highlighting Problems

If you introduce an error into a file, such as a syntax error, cell highlighting and dividers may not appear as you expect. Although dividers and highlighting for existing cells remain in place, cells you insert after you have introduced the syntax error do not appear highlighted. In addition, if you close and reopen the file, then all cell dividers are gone and none of the cells appears highlighted.

For example, suppose your code currently appears as specified in the "Example of Defining Cells" on page 6-191, and as shown here.
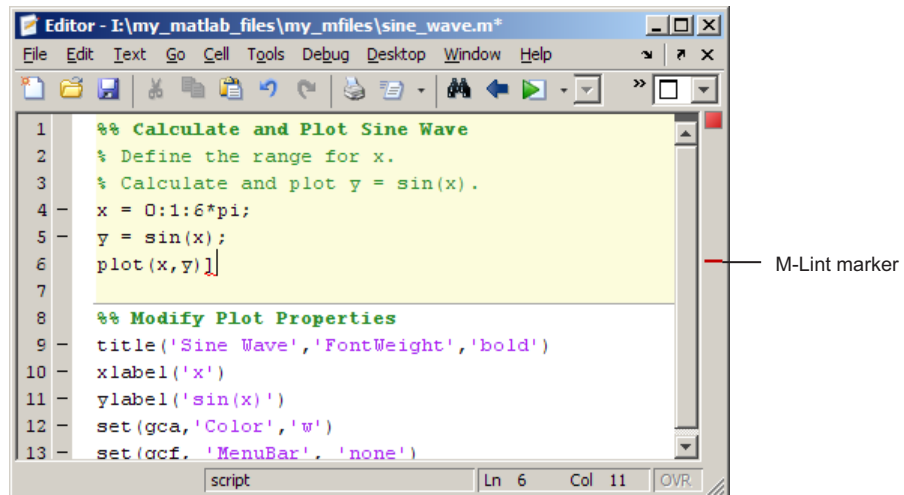
If you accidentally insert a syntax error (a closing bracket at the end of line 6), the error is evident by the M-Lint marker. The cell highlighting remains as-is.
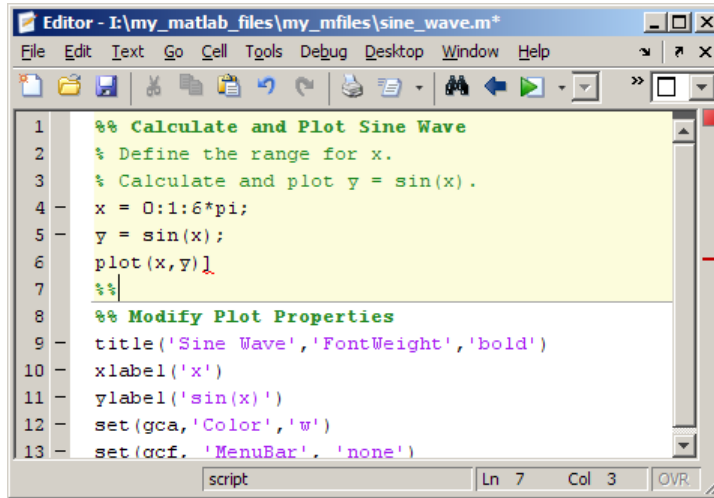
M-Lint marker

However, if you attempt to introduce a new cell at line 7, a cell divider does not appear above line 7 and the highlighting does not indicate a new cell, as you might expect.

In addition, if you save, close, and then reopen the file, all cell dividers and highlighting are gone.

To fix the problem, correct the syntax error. The cell dividers and highlighting reappear.

Removing Cells

To remove a cell, do one of the following:

- Delete one of the percent signs (%) from the line that starts the cell.

  This changes the line from a cell break to a standard comment.

- Delete the entire line that contains the %% characters.

In both cases, because you remove the cell break, MATLAB merges the two cells that were previously separated by the cell break.

Summary of Cell Mode and Cell Requirements

The following list summarizes facts to keep in mind when using cell mode and defining cells:

- Cell mode is supported for use with M-files only. It is not intended for use with plain text files.

- MATLAB does not execute the code in lines beginning with cell break characters, %%.

- MATLAB considers the entire file to be a single cell; therefore, the first line in a file does not have to begin with %%.

- For program control statements, such as if ... end, a cell must contain both the opening and closing statements, that is, it must contain both the if and the end statements.

- You can set preferences for cell display options by selecting **File > Preferences > Editor/Debugger > Display**, and then making choices for **Cell Display options**.

- If M-Lint finds errors in a file, cell dividers and highlighting may not appear as you expect. For details, see "Fixing Cell Highlighting Problems" on page 6-193.

For more information on M-Lint, see "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 6-123.

Understanding Nested Cells

You can insert cells within nested code, which results in nested cells. The following sections illustrate how inserting explicit cell breaks interacts with the implicit cell breaks that MATLAB inserts within an M-file.

- "M-File Without Explicit Cell Breaks" on page 6-197

- "How Nesting Cell Breaks Result in Cells" on page 6-199

- "Example M-File with Nested Cell Breaks" on page 6-200

- "Associating Cell Breaks with Subfunctions" on page 6-205

M-File Without Explicit Cell Breaks

The following code when viewed in an M-file displays no cells or highlighting. It is a single, implicit cell, defined by MATLAB.

```
function fourier
    t = 0:.1:pi*4;
    y = sin(t);
    updatePlot(1,t,y);
```

```
        for k = 3:2:9
            y = y + sin(k*t)/k;
            display(sprintf('When k = %.1f',k));
        end
    end

    function updatePlot(k,t,x)
        cla
        plot(t,x)

    end
```

This code appears as shown in the following image when you view it in the Editor.

How Nesting Cell Breaks Result in Cells

Suppose you insert two cell breaks into `fourier.m` as follows:

1 One within the `fourier` function, at line 5.

2 One within the `for` loop, at line 7.

This results in the following cells, which are illustrated in "Example M-File with Nested Cell Breaks" on page 6-200:

• One cell at the outermost level, from the top to the bottom of the file.

- Two cells at the second level, within the `fourier` function:

    - One from the implicit break at line 2 to the explicit break at line 5.

    - One from the explicit break at line 5 to the implicit break before line 11 (end of the function).

- One cell at the third level, within the `for` loop from the explicit line break at line 7 to the implicit line break before line 10.

Example M-File with Nested Cell Breaks

The following images illustrate how inserting explicit cell breaks, as described in"How Nesting Cell Breaks Result in Cells" on page 6-199, affect the appearance of the M-file:

- **First level of nesting** — When you place the cursor outside a function, at the outermost level, the entire file appears highlighted, showing that it comprises a cell at this level of nesting.

Cursor at outermost level

MATLAB only defines implicit cell breaks in a code block if you specify an explicit cell break within that code block. Therefore, because function `updatePlot` in this example has no explicit (and therefore, no implicit) cell breaks defined for it, when you place the cursor within that function, MATLAB considers the cursor to be within the cell that encloses the whole file.

Cursor in function with no explict, and therefore no implicit, cells defined.

- **Second level of nesting** — When you place the cursor within the function (but outside the for loop), either the first or second cell at this level of nesting appears highlighted, depending on where the cursor is located.
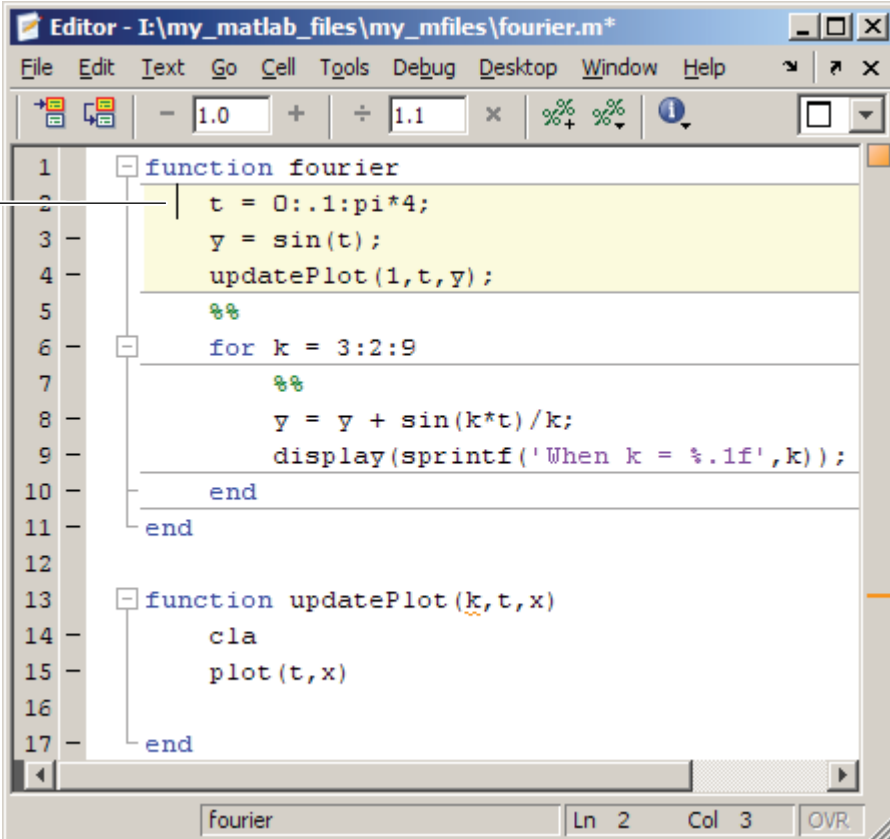
First cell within
function
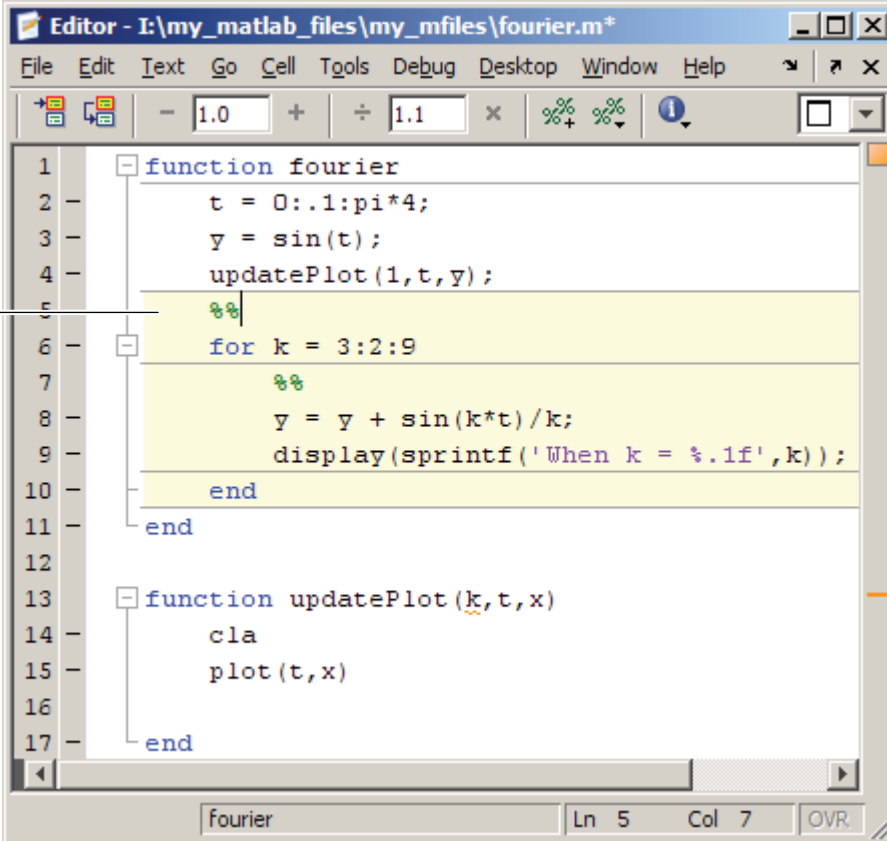
```
Editor - I:\my_matlab_files\my_mfiles\fourier.m*

File  Edit  Text  Go  Cell  Tools  Debug  Desktop  Window  Help

1    function fourier
2        t = 0:.1:pi*4;
3        y = sin(t);
4        updatePlot(1,t,y);
5        %%
6        for k = 3:2:9
7            %%
8            y = y + sin(k*t)/k;
9            display(sprintf('When k = %.1f',k));
10       end
11   end
12
13   function updatePlot(k,t,x)
14       cla
15       plot(t,x)
16
17   end

fourier                          Ln  2      Col  3      OVR
```

Second cell within function

```
Editor - I:\my_matlab_files\my_mfiles\fourier.m*
File  Edit  Text  Go  Cell  Tools  Debug  Desktop  Window  Help

1     function fourier
2         t = 0:.1:pi*4;
3         y = sin(t);
4         updatePlot(1,t,y);
5         %%
6         for k = 3:2:9
7             %%
8             y = y + sin(k*t)/k;
9             display(sprintf('When k = %.1f',k));
10        end
11    end
12
13    function updatePlot(k,t,x)
14        cla
15        plot(t,x)
16
17    end

fourier                          Ln  5      Col  7      OVR
```
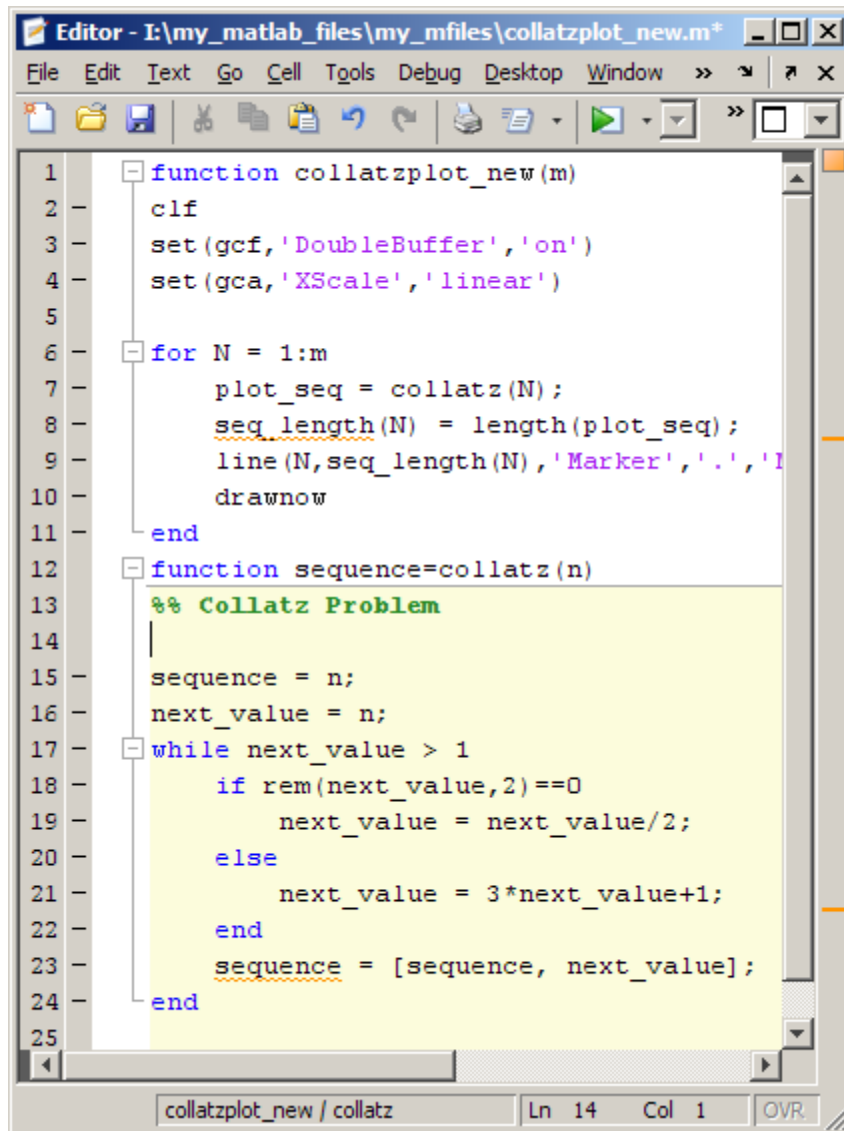
- **Third level of nesting** — When you place the cursor within the `for` loop, the cell within this loop is highlighted.

Cell within for loop
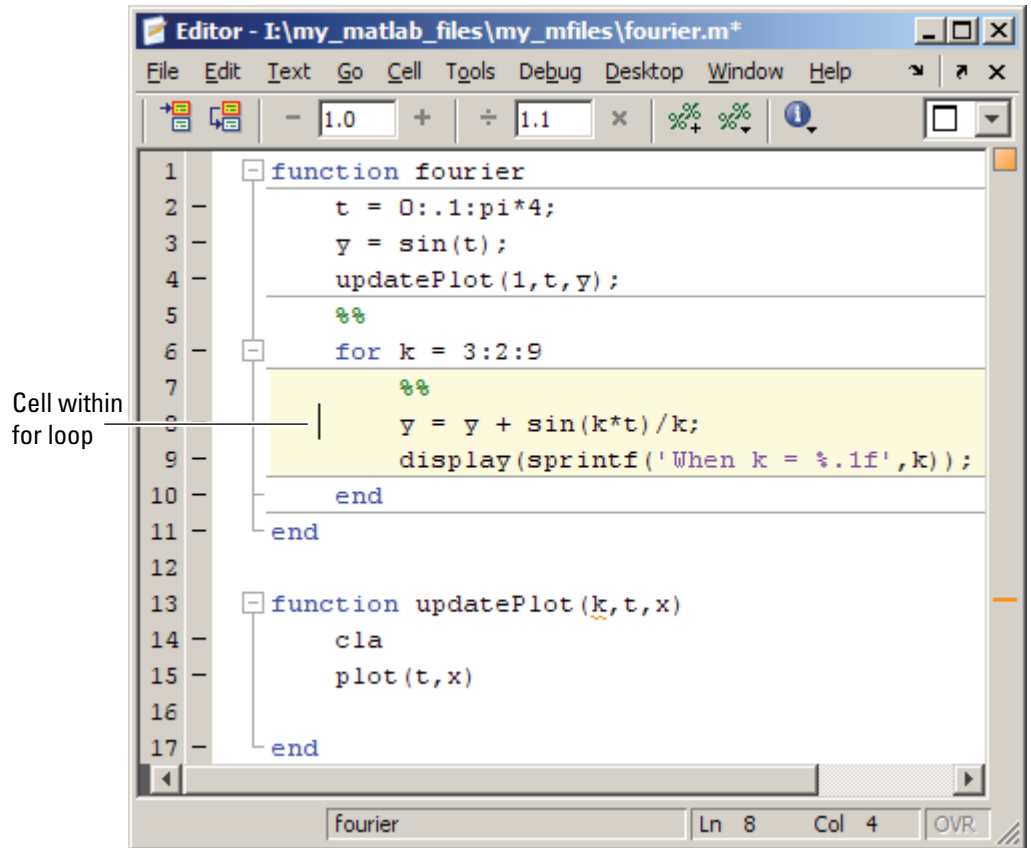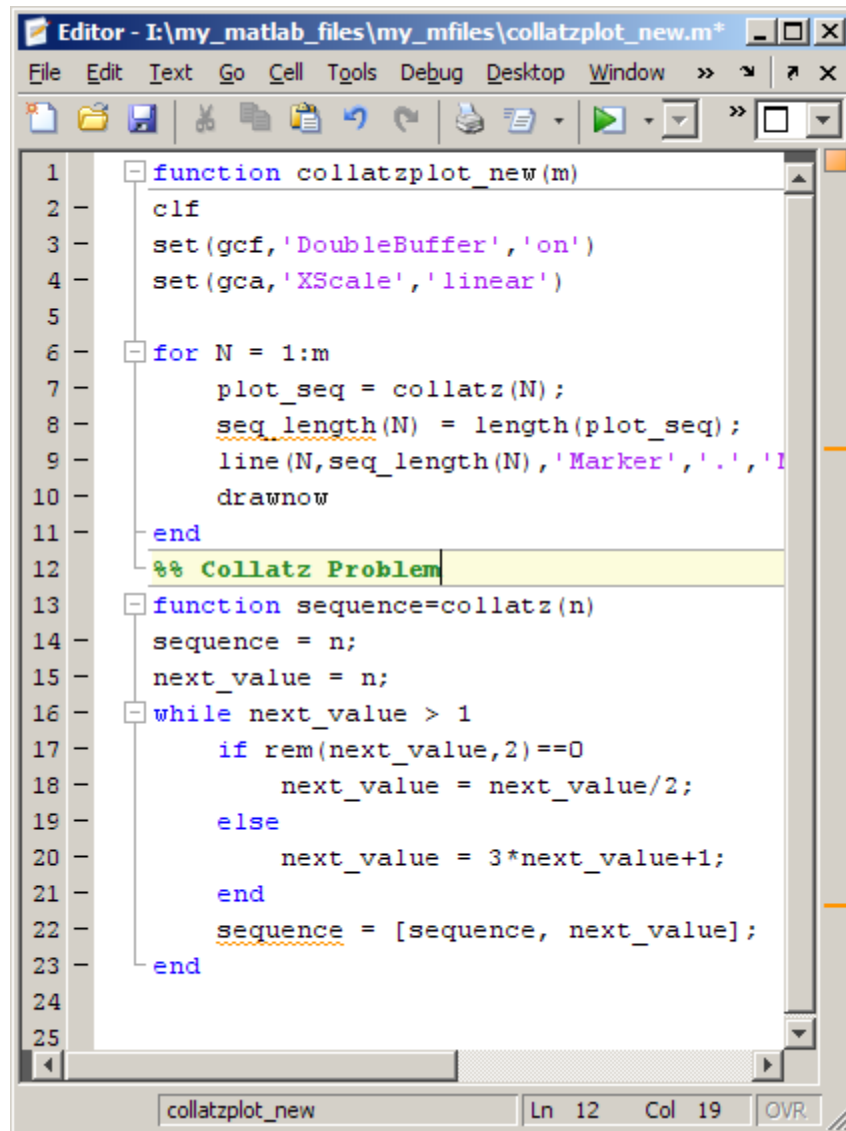
Associating Cell Breaks with Subfunctions

Be aware that if you want a cell break to be associated with a subfunction, you should place it within the subfunction, rather than above the subfunction declaration. Otherwise, it creates a single cell within the code block that precedes the subfunction. The following two images demonstrate this using `collatzplot_new.m`.

```
Editor - I:\my_matlab_files\my_mfiles\collatzplot_new.m*    _ □ ×

File   Edit   Text   Go   Cell   Tools   Debug   Desktop   Window   »   ↘   ↗ ×

1     function collatzplot_new(m)
2  -   clf
3  -   set(gcf,'DoubleBuffer','on')
4  -   set(gca,'XScale','linear')
5
6  -   for N = 1:m
7  -        plot_seq = collatz(N);
8  -        seq_length(N) = length(plot_seq);
9  -        line(N,seq_length(N),'Marker','.','I
10 -        drawnow
11 -   end
12    function sequence=collatz(n)
13    %% Collatz Problem
14    |
15 -   sequence = n;
16 -   next_value = n;
17 -   while next_value > 1
18 -        if rem(next_value,2)==0
19 -            next_value = next_value/2;
20 -        else
21 -            next_value = 3*next_value+1;
22 -        end
23 -        sequence = [sequence, next_value];
24 -   end
25

collatzplot_new / collatz          Ln  14     Col  1      OVR
```

Evaluating M-File Cells

As you develop an M-file, you can use the Editor cell features to evaluate the M-File cell-by-cell. This method helps you to experiment with, debug, and fine-tune your code. You can navigate from cell to cell, and evaluate each cell individually. See the following topics for details:

Navigating Among Cells in an M-File

Operation	Instructions
Move to the next cell.	Select **Cell > Next Cell**.
Move to previous cell.	Select **Cell > Previous Cell**.

Move to a specific cell.	Do either of the following:
	• Use the Editor **Cell Mode** toolbar, as follows:
	   1 Click the Show cell titles button ⚹.
	   2 Select the cell title to which you want to move.
	• Use the **Go** menu, as follows:
	   1 Select **Go > Go To**.
	
	     The Go To dialog box opens.
	   2 Select **Function or cell title**.
	   3 Select the cell title to which you want to move.
	   4 Click **OK**.

Evaluating Cells in an M-File

The cell evaluation features run the cell code currently shown in the Editor, even if the file contains unsaved changes. The file does not have to be on the search path. To evaluate a cell, it must contain all the values it requires, or the values must exist in the MATLAB workspace.

To run the code in a cell, use the **Cell** menu evaluation items or equivalent buttons in the cell mode toolbar. When you evaluate a cell, the results display in the Command Window, figure window, or elsewhere, depending on the code evaluated.

Operation	Instructions
Run the code in the current cell.	Select **Cell > Evaluate Current Cell** or click the Evaluate cell button 📋.
Run the code in the current cell, and then move to the next cell.	Select **Cell > Evaluate Current Cell and Advance** or click the Evaluate cell and advance button 📋.
Run all the code in the file.	Select **Cell > Evaluate Entire File** or click the Evaluate entire file button 📋. By default, the Evaluate entire file button is not on the Editor toolbar. See "Setting Toolbars Preferences for Desktop Tools" on page 2-138 for information on how to add it. **Note** A beep indicates there is an error. See the Command Window for the error message.

Processing Considerations When Evaluating Cells

This section describes processing considerations that you should take into account when you evaluate cells in M-files.

Setting Breakpoints. While you can set breakpoints and debug a file containing cells, when you evaluate a file from the **Cell** menu or cell toolbar, breakpoints are ignored. To run the file and stop at breakpoints, use **Run/Continue** in the **Debug** menu. This means you cannot debug while running a single cell.

Using Cells in Function M-Files. You can define and evaluate cells in function M-files as long as the variables referenced in the cell are in your workspace. This can be useful during debugging. If execution is stopped at a breakpoint, you can define cells and execute them without saving the file. If you are not debugging, add the necessary variables to the base workspace, and then execute the cells.

Using Function Names as Variable Names in Cells. If you use a MATLAB function name as a variable name within a cell, you might receive an unexpected error when you evaluate the cell. The precedence rules that MATLAB typically follows do not apply when it evaluates a cell. Typically, MATLAB evaluates variables before functions. However, when you evaluate cells, MATLAB parses all the cell code and loads it into memory before evaluating it. Therefore, functions might be evaluated before variables under some circumstances, as illustrated by the following example:

Suppose you create a MAT file, `mydata.mat`, using the following commands:

```
clear all
info=5;
save mydata.mat
clear all
```

When you enter the following commands in the Command Window, `b` evaluates to 5, as expected:

```
load mydata
b=info
```

However, when you evaluate the same commands in an M-File cell, `b` evaluates to the MATLAB `info` function, thus the Command Window displays the following error:

```
??? Error using ==> info
Too many output arguments.
```

For this reason, you may want to avoid using function names as variable names within M-file cells.

Modifying Values in a Cell

You can use cell features to modify numbers in a cell, which also automatically reevaluates the cell. This helps you experiment with and fine-tune your code.

To modify a number in a cell, select the number (or place the cursor near it) and use the value modification tool in the cell toolbar. Using this tool, you can specify a number and press the appropriate math operator to add (increment), subtract (decrement), multiply, or divide the number. The cell then automatically reevaluates.

Decrement/increment number

Divide/multiply number

Divide and multiply buttons

Decrement and increment buttons
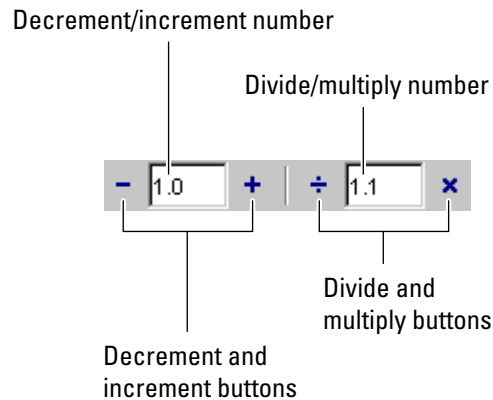
You can use the numeric keypad operator keys (-, +, /, and *) instead of the operator buttons on the toolbar.

Note MATLAB software does not automatically save changes you make to values using the cell toolbar. To save changes, select **File > Save**.

Example of Evaluating Cells

In this example, modify the values for x in sine_wave.m:

1 Run the first cell in sine_wav.m. Click somewhere in the first cell, that is, between lines 1 and 6. Select **Cell > Evaluate Current Cell**. The following figure appears.

Plot generated by running sine_wave.m.
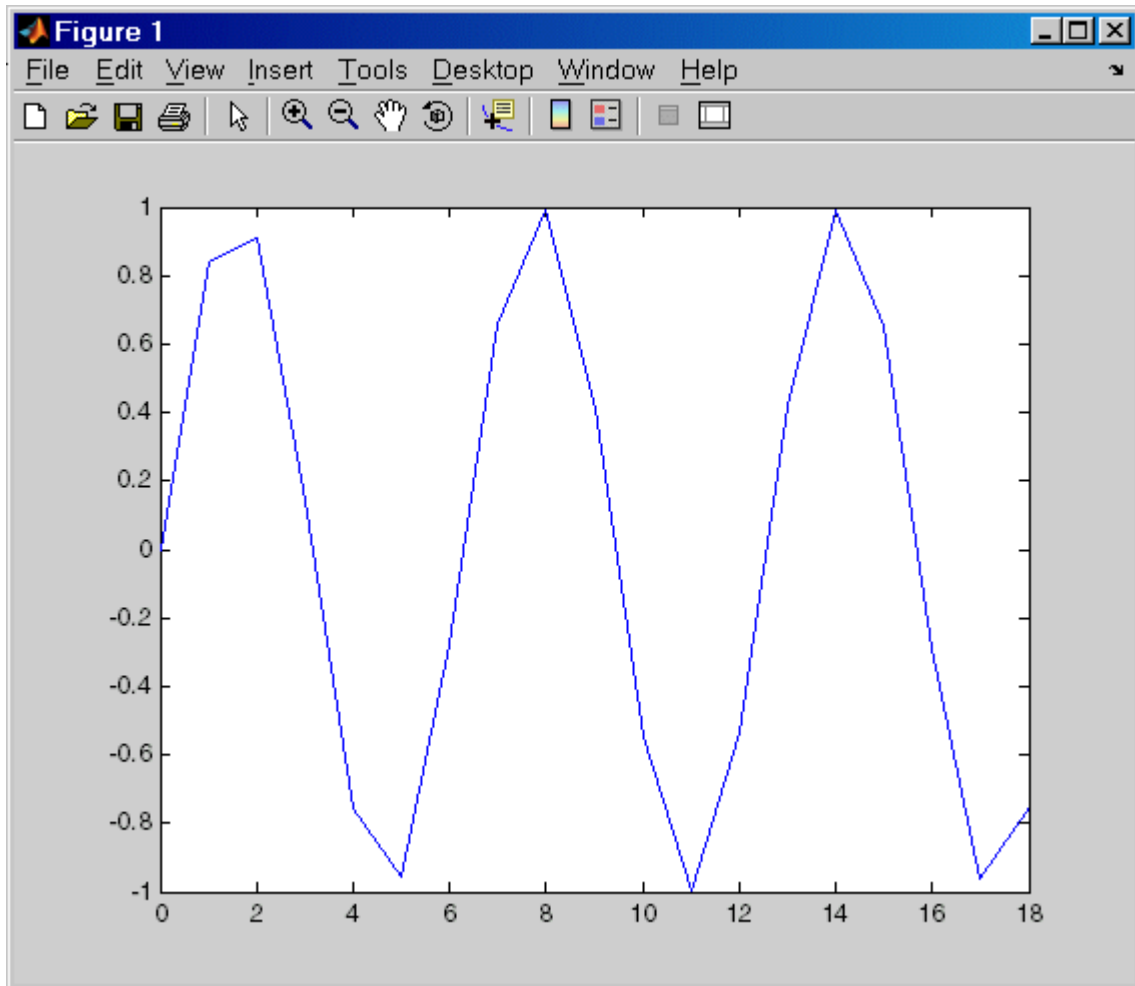
2 Assume you want to produce a smoother curve. Use more values for x in 0:1:6*pi. Position the cursor in line 4, next to the 1. In the cell toolbar,

change the `1.1` default multiply/divide by value to `2`. Click the Divide button ÷.

Line 4 becomes

```
4 -    x = 0:0.5:6*pi;
```

and the length of x doubles. The plot automatically updates. The curve still has some rough edges.

3 To add more values for x, click the Divide button three more times. Line 4 becomes

```
4 -    x = 0:0.0625:6*pi;
```

The curve is smooth, but because there are more values, processing time is slower. It would be better to find a smaller x that still produces a smooth curve.

4 In the cell toolbar, click the Multiply button once. The increment for x as shown in line 4 changes from `0.0625` to `0.125`.

The resulting curve is still smooth.

5 Save these changes. Select **File > Save**.

6 Now you can apply the plot properties, defined in the second cell, that is, lines 7 through 12. You do not need to evaluate the entire file to apply the plot properties. Instead, position the cursor in the second cell and use the shortcut **Ctrl+Enter** to evaluate the current cell. (The shortcut appears with the menu item, **Cell > Evaluate Current Cell**.)

MATLAB updates the figure.
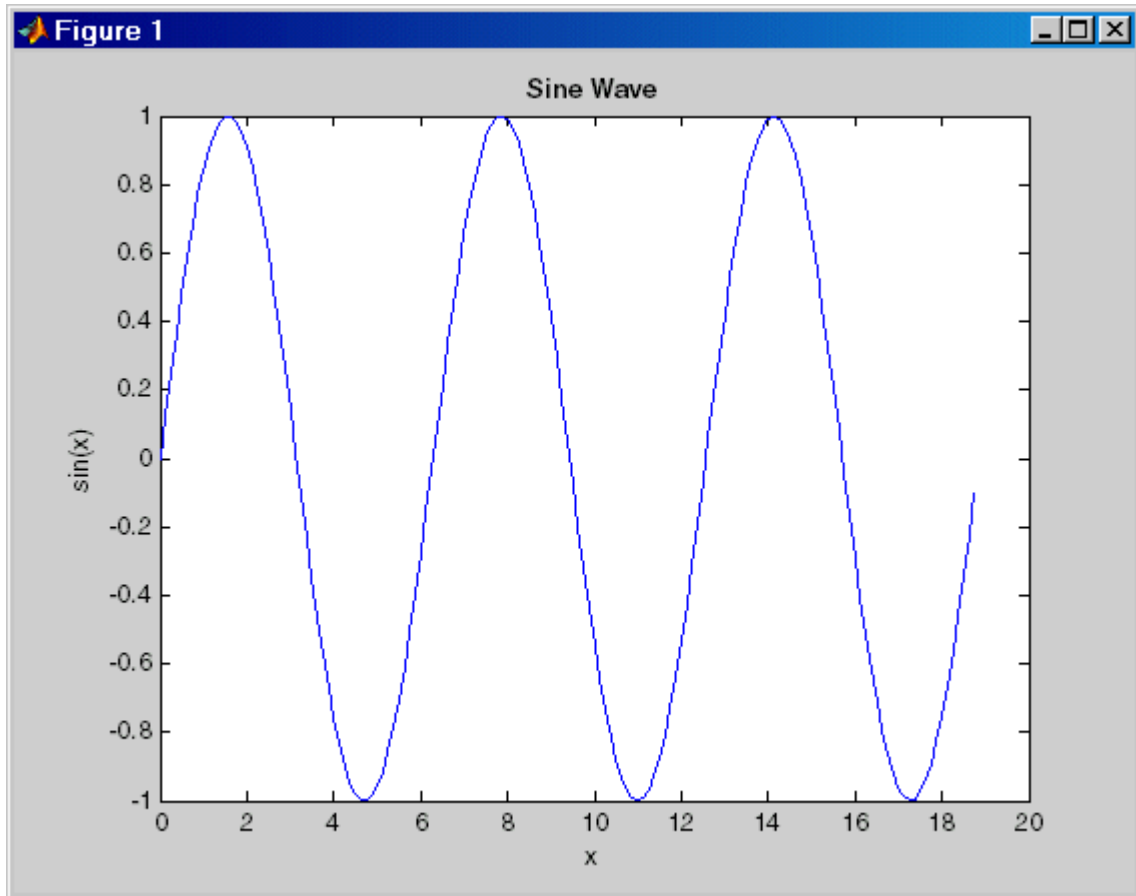
Tuning and Managing M-Files

This set of tools Postscript provides useful information about the M-files in a directory that can help you refine the files and improve performance. The tools can help you polish M-files before providing them to others to use.

- "Using M-File Reports" on page 7-2
- "M-Lint Code Check Report" on page 7-21
- "Profiling for Improving Performance" on page 7-33

Using M-File Reports

In this section...
"Refining and Improving M-Files Using Reports" on page 7-2
"Identifying M-Files with Reminder Annotations" on page 7-4
"Generating a Summary View of the Help Components in M-Files" on page 7-8
"Displaying and Updating a Report on the Contents of a Directory" on page 7-12
"Displaying Dependencies Among M-Files" on page 7-15
"Identifying How Much of an M-File Ran When Profiled" on page 7-19

See also "M-Lint Code Check Report" on page 7-21, and the File and Directory Comparisons tool.

Refining and Improving M-Files Using Reports

Reports help you refine the M-files in a directory and improve their performance. They are also useful for checking the quality of files before you distribute them for use by others, such as for a finished project, to share on MATLAB Central, or for a toolbox. (A toolbox is a collection of files for use with MATLAB and related products.)

Accessing Reports

Access reports from the MATLAB Current Directory browser, as follows:

1 Select **Desktop > Current Directory**.

For more information, see "Managing Files and Working with the Current Directory" on page 5-55.

2 Navigate to the directory containing the M-files for which you want to produce reports.

3 In the Current Directory browser toolbar, click the **Actions** button ⚙,
and then select the type of report you want to run for all the M-files in
the current directory.
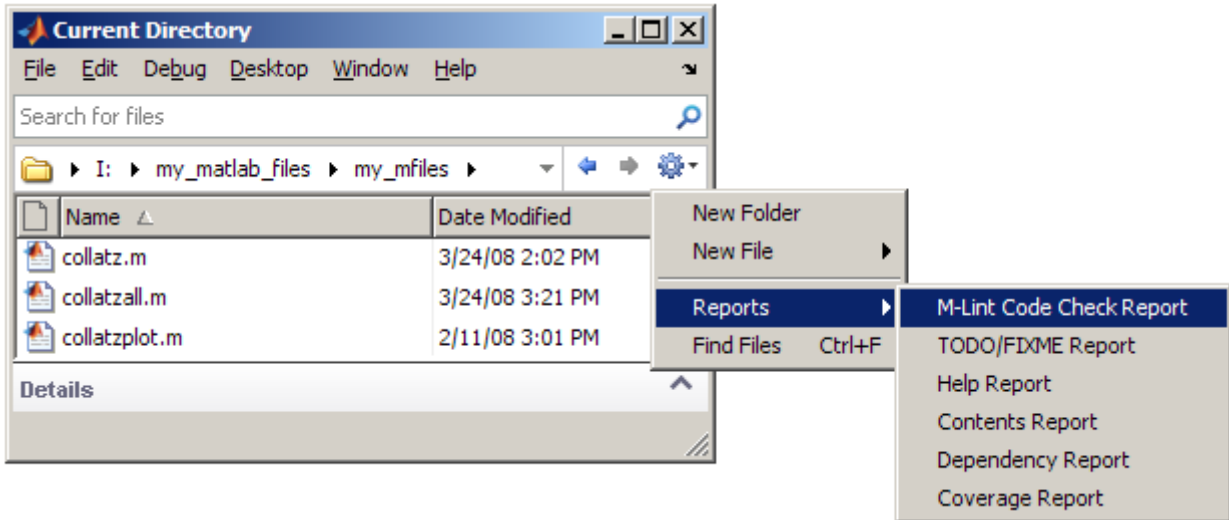
The report you select appears as an HTML document in the MATLAB Web
Browser.

Note You cannot run reports when the path is a UNC (Universal Naming
Convention) path, that is, starts with \\. Instead, use an actual hard drive on
your system, or a mapped network drive.

Using Reports

All M-File reports contain various links that enable you to access additional
information, as described in the table that follows:

To	Do this
Open a file in the Editor to view it or make changes to it.	Click a file name in the report.

To	Do this
Open a file at the line listed in a report.	Click the line number.
Update a report after making changes to the report options.	Click **Rerun This Report.**
Update a report after changing any files in the directory.	Click **Rerun This Report.**
Generate the same type of report for a different directory.	**1** Keep the current report open. **2** Change the MATLAB current directory. **3** Click **Run Report on Current Directory**.

Note Clicking **Rerun This Report** reruns the report for the directory shown in the report, not for the MATLAB current directory.

Identifying M-Files with Reminder Annotations

The TODO/FIXME Report identifies all the M-files in a given directory that you have annotated by adding comments with the text TODO, FIXME, or a string of your choosing. This enables you mark, and then find later, areas in an M-File with annotations to indicate that you intend to improve, complete, or perform some other update in the future. The TODO/FIXME Report presents a list of files containing the annotations in a Web browser.

Enter any text string in this field.

Click the line number to open the M-File at that line in the Editor, so you can make changes.

Working with TODO/FIXME Reports

1 Select **Desktop > Current Directory** and navigate to the directory containing the M-files for which you want to produce a TODO/FIXME report.

2 On the Current Directory browser toolbar, click the **Actions** button 🔅, and then select **Reports > TODO/FIXME Report**.

The TODO/FIXME Report opens in the MATLAB Web browser.

3 In the TODO/FIXME Report window, select one or more of the following to specify the lines that you want the report to include:

- TODO

- FIXME

- The text field check box

  Then, enter any text string, including a regular expression, in the text field. For example, you might enter NOTE, tbd, or re.*check.

4 Run the report on the M-Files in the current directory, by clicking **Rerun This Report**

The Window refreshes with a list of all the lines in the M-File programs within the specified directory that contain the strings you selected in step 1. Matches are not case-sensitive.

Optionally, if you want to run the report on a directory other than the one currently specified in the TODO/FIXME Report window, change the current directory to the one where you want to run the report, and then click **Run Report on Current Directory**.

To open an M-File in the Editor at a specific line, click the line number in the report. Then you can make changes, as needed.

Suppose you have an M-File, area.m, in the current directory. The code for area.m is shown in the image that follows.

```
Editor - I:\my_matlab_files\my_mfiles\area.m                              _ □ ×
File  Edit  Text  Go  Cell  Tools  Debug  Desktop  Window  Help              ↘  ↗ ×
 1    function [output] = area(flag,radius)
 2    % This function calculates the area of the entity
 3    % flag = 1 for calculating the area of a
 4    % flag = 2 for calculating the surface area of a sphere
 5    % radius = radius of the entity
 6
 7 –  switch flag
 8    % Modify the function to include the area of square
 9    % and rectangle. (todo)
10 –      case 1
11 –          output = pi * radius^2;
12 –      case 2
13 –          output = 4 * pi * radius^2;
14    % Fixme: Is the area of hemisphere as below?
15    %   case 3
16    %       output = 2 * pi * radius^2;
17    % fIXME
18 –      otherwise
19 –          disp('Incorrect flag');
20 –          output = NaN;
21    % NOTE: Find out from the manager if we need to include
22    % the area of a cone
23 –  end
                                              area          Ln  25    Col  1    OVR
```

When you run the TODO/FIXME report on the directory containing `area.m`, with the `TODO` and `FIXME` strings selected and the string `NOTE` specified and selected, the report (as shown at the beginning of this section) lists:

9 and rectangle. (todo)
14 Fixme: Is the area of hemisphere as below?
17 fIXME
21 NOTE: Find out from the manager if we need to include

Notice the report includes the following:

- Line 9 as a match for the `TODO` string. The report includes lines that have a selected string regardless of its placement within a comment.

- Lines 14 and 17 as a match for the `FIXME` string. The report matches selected strings in the M-File regardless of their casing.

- Line 21 as a match for the `NOTE` string. The report includes lines that have a string specified in the text field, assuming it is selected.

Generating a Summary View of the Help Components in M-Files

The Help Report presents a summary view of the help component of your M-files. Use this information to help identify files of interest or files that lack help information. It is a good practice to provide help for your files not only to help you recall their purpose, but to help others who might use the files.

In MATLAB, the M-file help component is all contiguous nonexecutable lines (comment lines and blank lines), starting with the second line of a function M-file or the first line of a script M-file. For more information about creating help for your own M-files, see the reference page for the `help` function.

Working with Help Reports

1 Select **Desktop > Current Directory** and navigate to the directory containing the M-files for which you want to produce a Help Report.

2 On the Current Directory browser toolbar, click the **Actions** button 🔧, and then select **Reports > Help Report**.

The Help report opens in the MATLAB Web Browser.

3 Select one or more options, described in the following list, to have the Help Report display the specified help information:

- **Show subfunctions** to have the Help Report display help information for all subfunctions called by each function. Help information for subfunctions is highlighted in gray.

- **Description** to have the Help Report display the first line of help in the M-file. If the first comment line is empty, or if there is not a comment before the executable code, then **No description line**, highlighted in pink, appears instead.

- **Examples** have the Help Report display the line number where the examples section of the M-file help begins. The Help Report looks for a line in the M-file help that begins with the string `example` or `Example` and displays any subsequent nonblank comment lines. Select this option to easily locate and go to examples in your M-files.

  It is a good practice to include examples in the help for your M-files. If you do not have examples in the help for all your M-files, use this option to identify those without examples. If the report does not find examples in the M-file help, **No example**, highlighted in pink, appears.

- **Show all help** have the Help Report display complete M-file help, which is all contiguous nonexecutable lines (comment lines and blank lines), starting with the second line of a function M-file, or the first line of a script M-file. The M-file help shown also includes overloaded functions and methods, which are not actually part of the M-file help comments, but are automatically generated when `help` runs.

  If the comment lines before the executable code are empty, or if there are no comments before the executable code, **No help**, highlighted in pink, appears instead.

- **See Also** have the Help Report display the line number for the see-also line in the M-file help. The see- also line in M-file help lists related functions. When the MATLAB Command Window displays the help for an M-file, any function name listed on the see-also line appears as a link you can click to display its help. It is a good practice to include a see-also line in the help for your M-files.

  The report looks for a line in the M-file help that begins with the string `See also`. If the report does not find a see-also line in the M-file help, **No see-also line**, highlighted in pink, appears. This helps you identify those M-files without a see-also line, should you want to include one in each M-file.

  The report also indicates when an M-file noted in the see-also line is not in a directory on the search path. You might want to move that file to a directory that is on the search path. If not, you will not be able to click

the link to get help for the file, unless you then add its directory to the path or make its directory become the current directory.

- **Copyright** have the Help Report display the line number for the copyright line in the M-file. The report looks for a comment line in the M-file that begins with the string `Copyright` and is followed by `year1-year2` (with no spaces between the years and the hyphen that separates them). It also notes if the end of the date range is not the current year.

  It is a good practice to include a copyright line in the help for your M-files, that notes the year you created the file and the current year. For example, for an M-file you created in 2001, include this line

  ```
  % Copyright 2001-2008
  ```

  If the report does not find a copyright line in the M-file help, **No copyright line**, highlighted in pink, appears. This helps you identify those files without a copyright line, should you want to include one in each M-file.

4 Click **Rerun This Report**. Your report resembles the following image.

Help Report

❌ Help Report	_ □ ×

File Edit View Go Debug Desktop Window Help

← → ↻ 🖨 🔍

Help Report

Rerun This Report	Run Report on Current Directory

☑ Show subfunctions ☑ Description ☑ Examples

☑ Show all help ☑ See also ☑ Copyright

Report for directory D:\mymfiles\mydemos

M-File List

chirpy	Chirping
	Chirping Usa a polynomial-defined curve to signal. Make the chirping sound m
	No example No see-also line No copyright line

Check **Description** to
see the first help line.

collatzall	Plot length of sequence for Collatz

Check **Show all help** to
see the rest of the help.

	Plot length of sequence for Colla Prepare figure
	No example No see-also line No copyright line

Check **Show subfunctions**
to see help information
for subfunctions.

collatzall/collatz	Collatz problem. Generate a sequenc
	Collatz problem. Generate a seque For any positive integer, n:

Displaying and Updating a Report on the Contents of a Directory

The Contents Report displays information about the integrity of the `Contents.m` file for a given directory. A `Contents.m` file includes the file name and a brief description of each M-file in the directory. The Contents Report helps you to maintain the `Contents.m` file. It displays discrepancies between the `Contents.m` file and the M-files in the directory.

When you type `help` followed by the directory name, such as `help mydemos`, The MATLAB Command window displays the information contained within the `mydemos/Contents.m` file. For more information, see "Providing Help for Your Program" in the MATLAB Programming documentation.

Working with Contents Reports

1 Select **Desktop > Current Directory** and navigate to the directory containing the M-files for which you want to produce a Contents report.

2 On the Current Directory browser toolbar, click the **Actions** button ⚙, and then select **Reports > Contents Report**.

The Contents Report opens in the MATLAB Web browser—or, if there is no `Contents.m` file for the directory, the report tells you the `Contents.m` file does not exist and asks if you want to create one. Click **yes** to automatically create the `Contents.m` file. Edit the `Contents.m` file in the Editor to include the names of files you plan to create, or to remove files that you do not want to expose when displaying help for the directory, such as files for internal use.

3 Update the `Contents.m` file to reflect changes you make to files in the directory. For example, when you remove a file from a directory, remove its entry from the `Contents.m` file.

The following options are available for updating the contents:

- **edit Contents.m**— Opens the `Contents.m` file in the Editor.

- **fix spacing**—Automatically align the file names and descriptions in the `Contents.m` file.

- **fix all** makes all of the suggested changes at once.

- To make changes on a case-by-case basis, read each question in the Contents Report and if you want to make the change suggested, click **yes**.
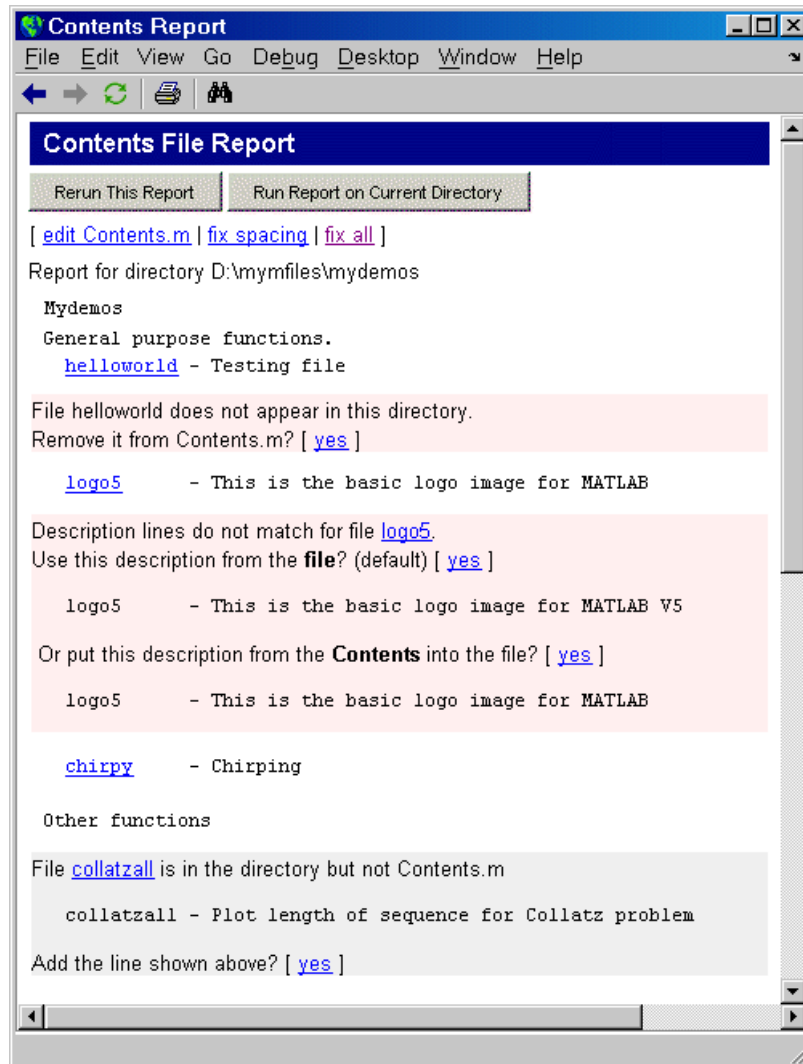
Messages in the Contents File Report

No Contents File. This message appears if there is no `Contents.m` file in the directory. Click **yes** to automatically create a `Contents.m` file, which contains the file names and descriptions for all M-files in the directory.

```
No Contents.m file. Make one? [ yes ]
```

File Not Found. This message appears when a file included in `Contents.m` is not in the directory. These messages are highlighted in pink. For example, a message such as

```
File helloworld does not appear in this directory.
Remove it from Contents.m? [ yes ]
```

means the `Contents.m` file includes an entry for `helloworld`, but that file is not in the directory. This might be because you removed the file `helloworld`, or you manually added it to `Contents.m` because you planned to create the file but have not as yet, or you renamed `helloworld`.

Description Lines Do Not Match. This message appears when the description line in the M-file help does not match the description provided for the M-file in `Contents.m`. These messages are highlighted in pink. Click **yes** to replace the description in the `Contents.m` file with the description from the M-file. Or select the option to replace the description line in the M-file help using the description for that file in `Contents.m`.

```
Description lines do not match for file logo5.
Use this description from the file? (default) [ yes ]
 logo5      - This is the basic logo image for MATLAB 7
Or put this description from the Contents into the file? [ yes ]
 logo5 - This is the basic logo image for MATLAB
```

Files Not In Contents.m. This message appears when a file in the directory is not in `Contents.m`. These messages are highlighted in gray. Click **yes** to add the file name and its description line from the M-file help to the `Contents.m` file.

```
collatzall is in the directory but not Contents.m
 collatzall - Plot length of sequence for Collatz problem
Add the line shown above? [ yes ]
```

Creating a New Contents.m File to Reflect All Files in the Current Directory

If you always want the Contents.m file to reflect all files in the current directory, you can automatically generate a new Contents.m file rather than changing the file based on the Contents Report, as follows:

1 Delete the existing Contents.m file.

2 Run the Contents Report.

3 Click yes when prompted for MATLAB to automatically create a Contents Report.

Displaying Dependencies Among M-Files

The Dependency Report shows dependencies among M-files in a directory. This is useful for determining:

- Which files in the directory are required by other files in the directory

- If any files in the current directory will fail if you delete a file

- If any called files are missing from the current directory

The report does not list M-files in the toolbox/matlab directory as dependencies because every MATLAB user has those files.

To provide meaningful results, the dependency report requires that the following be true:

- The search path when you run the report is the same as when you run the files in the directory. (That is, the current directory is at the top of the search path.)

- The files in the directory for which you are running the report do not change the search path or otherwise manipulate it.

- The files in the directory do not load variables, or otherwise create name clashes that result in different program elements with the same name.

Working with Dependency Reports

1 Select **Desktop > Current Directory** and navigate to the directory containing the M-files for which you want to produce a Dependency Report.

2 On the Current Directory browser toolbar, click the **Actions** button ⚙, and then select **Reports > Dependency Report**.

The Dependency report opens in the MATLAB Web Browser.

3 If you want, select one or more options within the report, as follows:

- To see a list of all M-files (children) called by each M-file in the directory (parent), select **Show child functions**.

  The report indicates where each child function resides, for example, in a specified toolbox. If a child function's location is listed as unknown, it might be because the child function is not on the search path or in the current directory, or the file may have been moved or deleted.

- To list the M-files that call each M-file, select **Show parent functions**.

  The report limits the parent (calling) functions to those in the current directory.

- To include subfunctions in the report, select **Show subfunctions**. Subfunctions are listed directly after the main function and are highlighted in gray.

4 Click **Run Report on Current Directory**.

The file chirpy.m calls two M-files in the Signal Processing Toolbox and one in the Image Processing Toolbox.

The file go.m calls moebius.m, located in the current directory.

Because the report is a static analysis, it cannot determine run-time data types and, therefore, cannot identify the particular class methods required by an M-file. If multiple class methods match a referenced method, the Dependency Report inserts a question mark link next to the file name, as shown in the following image.
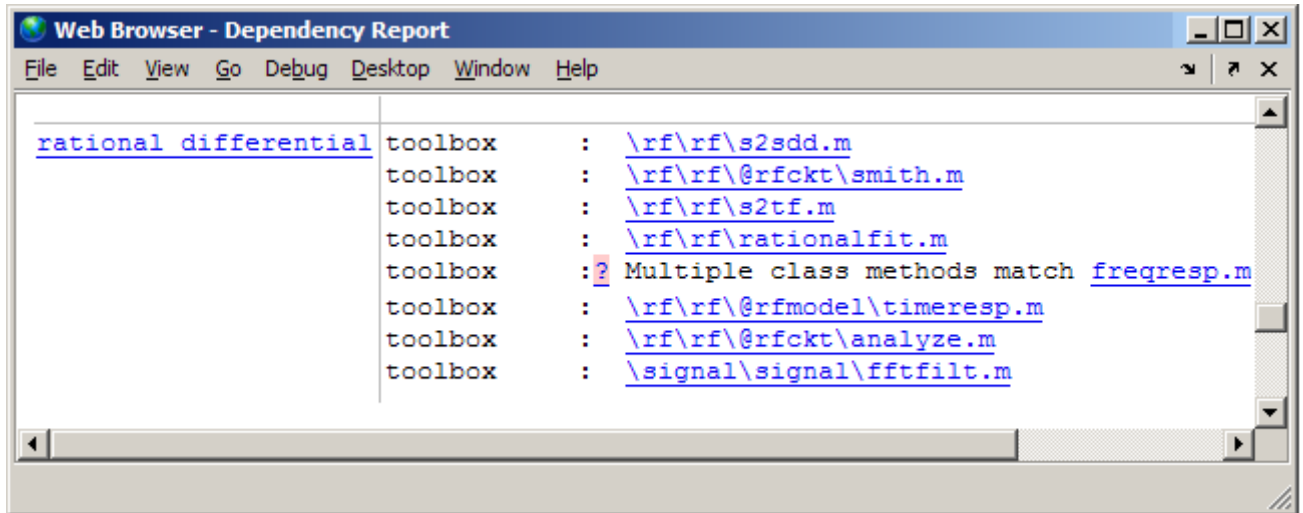
Click the question mark link to see a list of the class methods with the specified name that the MATLAB software might use. MATLAB lists *almost all* the method files on the search path that match the specified method file (in this case, freqresp.m). Do not be concerned if the list includes methods of classes and MATLAB built-in functions that are unfamiliar to you.

It is not necessary for you to determine which file will be used. MATLAB determines which method to use using the object that the program calls at run time. This list is provided as an indication of the possible toolboxes and method files used.

The following image shows the contents of the right side of the Web browser after you click the question mark link.

```
toolbox :    \rf\rf\s2sdd.m
toolbox :    \rf\rf\@rfckt\smith.m
toolbox :    \rf\rf\s2tf.m
toolbox :    \rf\rf\rationalfit.m
toolbox :? Multiple class methods match freqresp.m

   Unable to determine which of the following files will run: (Learn More)
   \control\ctrlobsolete\freqresp.m
   W:\bat\Akernel\perfect\matlab\toolbox\control\control\@lti\freqresp.m
   W:\bat\Akernel\perfect\matlab\toolbox\ident\ident\@idproc\freqresp.m
   W:\bat\Akernel\perfect\matlab\toolbox\ident\ident\@idpoly\freqresp.m
   W:\bat\Akernel\perfect\matlab\toolbox\ident\ident\@idmodel\freqresp.m
   W:\bat\Akernel\perfect\matlab\toolbox\ident\ident\@idfrd\freqresp.m
   W:\bat\Akernel\perfect\matlab\toolbox\rf\rf\@rfmodel\freqresp.m
```

The Dependency Report is similar to running the depfun function, although the two do not provide identical results. For performance purposes, the Dependency Report limits the functions considered. Therefore, do not use the Dependency Report to determine which M-files you need to provide when you tell someone to run a particular M-file; instead use the depfun function.

Identifying How Much of an M-File Ran When Profiled

Run the Coverage Report after you run the Profiler to identify how much of a file ran when it was profiled. For example, when you have an if statement in your code, that section might not run during profiling, depending on conditions.

You can run the Coverage Report from the Profiler, or follow these steps:

1 In the MATLAB desktop, select **Desktop > Profiler**. Profile an M-file in the Profiler. For detailed instructions, see "Profiling for Improving Performance" on page 7-33.

2 Select **Desktop > Current Directory** and navigate to the directory containing the M-file for which you ran the Profiler.

3 On the Current Directory browser toolbar, click the **Actions** button ⚙, and then select **Reports > Coverage Report**.

The **Coverage Report** appears, providing a summary of coverage for the M-file you profiled.

4 Click the **Coverage** link to see the Profile Detail Report for the file.

The Coverage Report shows the percentage of a file that ran when it was profiled.
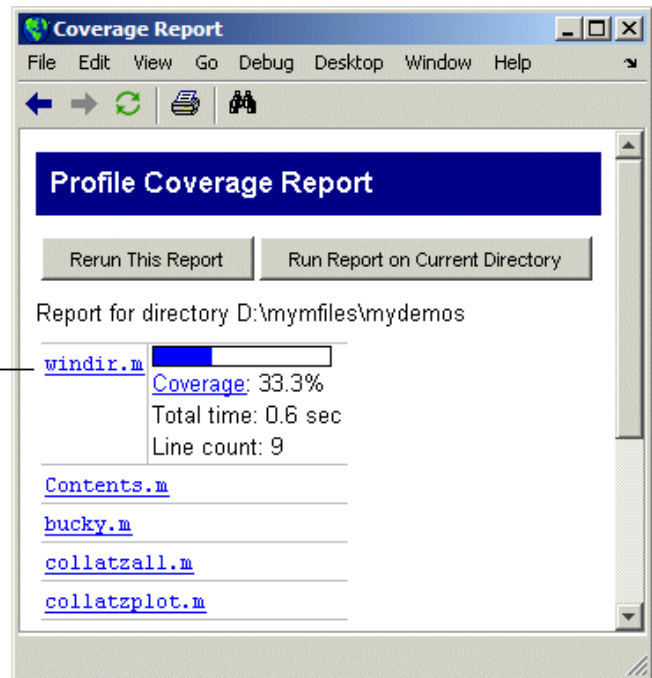
M-Lint Code Check Report

Running the M-Lint Code Check Directory Report

The M-Lint Code Check Report displays potential errors and problems, as well as opportunities for improvement in your code. The term "lint" is the name given to similar tools used with other programming languages such as C. M-Lint produces a message for each line of an M-file that it determines might be improved. For example, a common M-Lint message is that a variable foo in line 12 is defined but never used in the M-file.

To run the M-Lint code check directory report, follow these steps:

1 In the Current Directory browser, navigate to the directory that contains the M-files you want to check with M-Lint. To use the example shown in this documentation, lengthofline.m, you can change the current directory by running

```
cd(fullfile(matlabroot,'help','techdoc','matlab_env','examples'))
```

2 If you plan to modify the example, save the file to a directory for which you have write access, and then make that directory the current MATLAB directory. In this example, the file is saved to I:\MATLABFiles\mymfiles.

3 In the Current Directory browser toolbar, click the **Actions** button 🔅, and then select **Reports > M-Lint Code Check Report**.

The M-Lint code Check Report displays in the MATLAB Web browser, showing those M-files that M-Lint identified as having potential problems or opportunities for improvement.
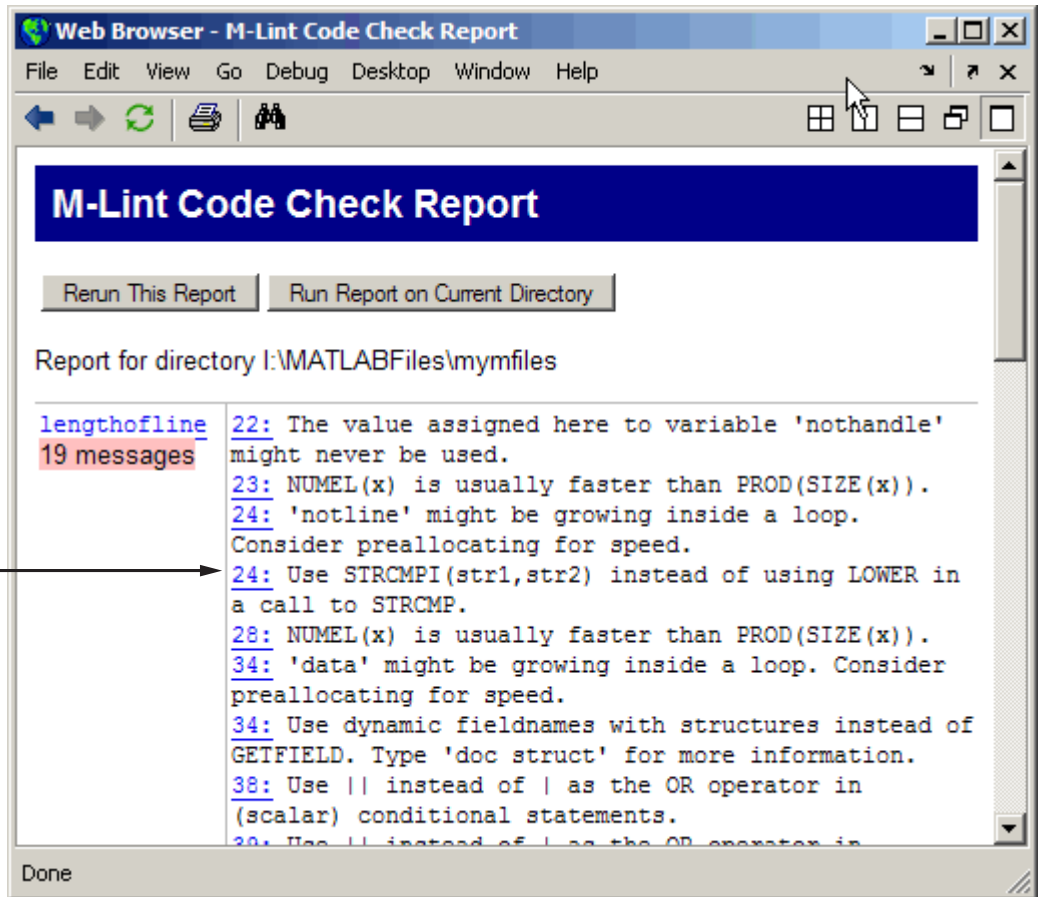
Line number and message describing a potential problem or improvement opportunity.

Click a line number to open the M-file in the Editor at the line.

4 For each message, review the suggestion and your code, click the line number to open the M-file in the Editor at that line, and make changes based on the message. Use the following general advice:

- If you are not sure what a message means or what to change in the code as a result, click on the link in the M-Lint message if one is provided, as described in "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 6-123.

- If the M-Lint message does not contain a link and you are not sure what a message means or what to change in the code as a result, use the

Help browser to look for related topics in the online documentation. For examples of messages and what to do about them, including specific changes to make for the example, `lengthofline.m`, see "Making Changes Based on M-Lint Messages" on page 7-24.

- M-Lint does not provide perfect information about every situation and in some cases, you might not want to make any changes based on the M-Lint message. In the event you do not want to change the code but you also do not want to see the M-Lint message for that line in the M-Lint Report, instruct M-Lint to ignore a line by adding `%#ok` to the end of the line in the M-file. (You can override the `%#ok` by running the `mlint` function with the `'-notok'` tag.)

- If there are certain messages or types of messages you do not want to see, you can set a preference so that M-Lint does not report them. Select **File > Preferences > M-Lint**. In **Select messages to enable**, clear the check box for messages you do not want to see. Review the settings for all messages to ensure you are seeing those pertinent to your file. Click **OK**. For more information, click the **Help** button in the **M-Lint Preferences** pane. The next time you run the report, the messages will not appear. You can use `%#ok` with a specific message ID so that only that type of message is suppressed—for more information, see the reference page for `mlint`.

5 After making changes, save the M-file. Consider saving the file to a different name if you made significant changes that might introduce errors. Then you can refer to the original file if needed to resolve problems with the updated file. Use **Tools > Compare Against** in the Editor to help you identify the changes you made to the file. For more information, see "Comparing Two Text Files" on page 6-79.

6 Run and debug the file(s) again to be sure you have not introduced any inadvertent errors.

7 If the M-Lint Code Check Report is already displayed, click **Rerun This Report** to update the report based on the changes you made to the file. Ensure the M-Lint messages are gone, based on the changes you made to the M-files.

Making Changes Based on M-Lint Messages

For information on how to correct the potential problems presented by M-Lint, use the following resources:

- If available, click the link for the extended message in the M-Lint message tooltip, as shown in the image following this list. Not all M-Lint messages have extended messages.

- Look for relevant topics in the Programming Fundamentals and "Programming Tips" documentation.

- Use the Help browser **Search** and **Index** panes to find documentation about terms presented in the M-Lint messages.

The following image shows an M-Lint tooltip with a link to an extended M-Lint message. The orange *underline* indicates that an extended M-Lint message will be presented in a tooltip when you hover over the word demolint in the code. (The orange *highlighting* indicates that an automatic fix is available.)
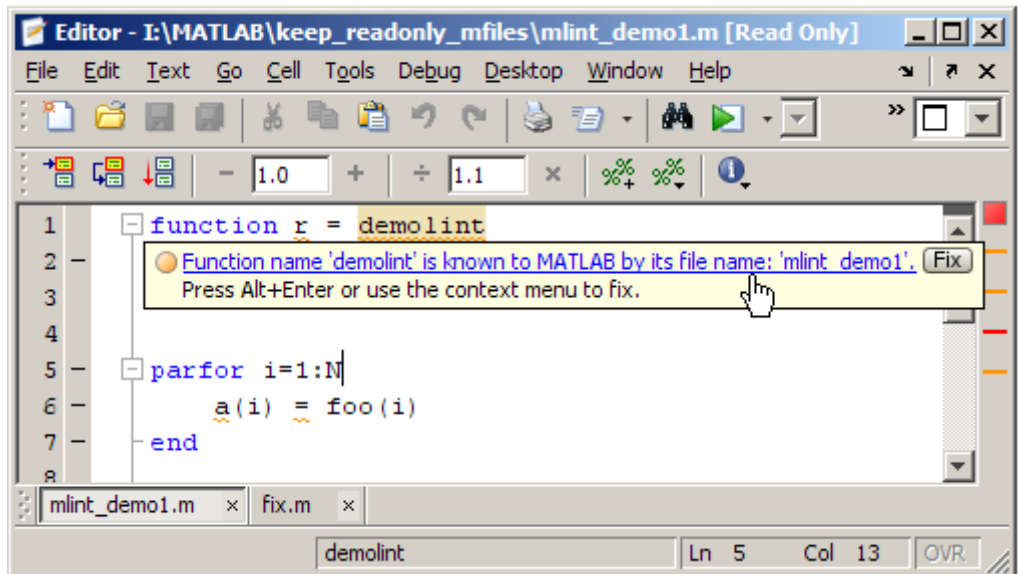
Other techniques to help you identify problems in and improve your M-files are in these topics:

- "Syntax Highlighting" on page 6-50 in the Command Window and the Editor

- "Examining Errors" on page 3-7 generated when you run the M-file

- "Finding Errors, Debugging, and Correcting M-Files" on page 6-120, namely the Editor and debugging functions

- "Profiling for Improving Performance" on page 7-33 for improving performance

Example Using M-Lint Messages to Improve Code

An example file, lengthofline.m, is included with the MATLAB product in *matlabroot*/matlab/help/techdoc/matlab_env/examples.

To run the M-Lint Code Check Report for lengthofline.m, change the current directory to its location by running

```
cd(fullfile(matlabroot,'help','techdoc','matlab_env','examples'))
```

In the Current Directory browser toolbar, click the **Actions** button ⚙, and then select **Reports > M-Lint Code Check Report**

The M-Lint Code Check Report appears, with its list of messages suggesting improvements you can make to lengthofline.m and any other files in the directory.

Web Browser - M-Lint Code Check Report

File   Edit   View   Go   Debug   Project   Desktop   Window   Help

M-Lint Code Check Report

Rerun This Report   Run Report on Current Directory

Report for directory I:\MATLABFiles\mymfiles

19 messages

22: The value assigned here to variable 'nothandle' might never be used.
23: NUMEL(x) is usually faster than PROD(SIZE(x)).
24: 'notline' might be growing inside a loop. Consider preallocating for speed.
24: Use STRCMPI(str1,str2) instead of using LOWER in a call to STRCMP.
28: NUMEL(x) is usually faster than PROD(SIZE(x)).
34: 'data' might be growing inside a loop. Consider preallocating for speed.
34: Use dynamic fieldnames with structures instead of GETFIELD. Type 'doc struct' for more information.
38: Use || instead of | as the OR operator in (scalar) conditional statements.
39: Use || instead of | as the OR operator in (scalar) conditional statements.
40: Use || instead of | as the OR operator in (scalar) conditional statements.
42: 'data' might be growing inside a loop. Consider preallocating for speed.
43: 'dim' might be growing inside a loop. Consider preallocating for speed.
45: 'dim' might be growing inside a loop. Consider preallocating for speed.
48: There may be a parenthesis imbalance around here.
48: There may be a parenthesis imbalance around here.
48: There may be a parenthesis imbalance around here.
48: There may be a parenthesis imbalance around here.
49: Terminate statement with semicolon to suppress output (in functions).
49: Use of brackets [] is unnecessary. Use parentheses to group, if needed.

Done

Messages and Resulting Changes for the lengthofline Example. The following table describes each message and demonstrates a way to change the file, based on the message.

Message — Code (Original Line Numbers)	Explanation and Updated Code (New Line Numbers)
22: The value assigned here to variable 'nothandle' might never be used. ——————————————————— 22 nothandle = ~ishandle(hline); 23 for nh = 1:prod(size(hline)) 24 notline(nh) = ~ishandle(hline(nh)) ...	In line 22, nothandle is assigned a value, but nothandle is probably not used anywhere after that in the file. The line might be extraneous and you could delete it. But it might be that you actually intended to use the variable, which is the case for the lengthofline example. Update line 24 to use nothandle, which is faster than computing ~ishandle for each iteration of the loop, as shown here. 22 nothandle = ~ishandle(hline); 23 for nh = 1:numel(hline) 24 notline(nh) = nothandle(nh) ...
23: NUMEL(x) is usually faster than PROD(SIZE(x)). ——————————————————— 23 for nh = 1:prod(size(hline))	While prod(size(x)) returns the number of elements in a matrix, the numel function was designed to do just that, and therefore is usually more efficient. Type doc numel to see the numel reference page. Change the line to 23 for nh = 1:numel(hline)
24: 'notline' might be growing inside a loop. Consider preallocating for speed. ——————————————————— 22 nothandle = ~ishandle(hline); 23 for nh = 1:numel(hline) 24 notline(nh) = ~ishandle(hline(nh)) ...	When you increase the size of an array within a loop, it is inefficient. Before the loop, preallocate the array to its maximum size to improve performance. For more information, see "Preallocating Memory" in the MATLAB Mathematics documentation. In the example, add a new line to preallocate notline before the loop. 23 notline = false(size(hline)); 24 for nh = 1:numel(hline) 25 notline(nh) = nothandle(nh) ...

Message — Code (Original Line Numbers)	Explanation and Updated Code (New Line Numbers)

`24:  Use STRCMPI(str1,str2) instead of using LOWER in a call to STRCMP.`
 	
— — — — — — — — — — — — — — — —
 	
`24 notline(nh)=~ishandle(hline(nh)) ||`
`~strcmp('line',lower(get(hline(nh),`
`'type')));`
	While
	
	`strcmp`
	`('line',lower(get(hline(nh)'type'))`
	
	converts the result of the `get` function to a lowercase string before doing the comparison, the `strcmpi` function ignores the case while performing the comparison, with advantages that include more efficiency. Change line 25 to
	
	`notline(nh) = nothandle(nh) ||`
	`~strcmpi('line',get(hline(nh),'type'));`

`28:  NUMEL(x) is usually faster than PROD(SIZE(x)).`
 	
— — — — — — — — — — — — — — — —
 	
`28 for nl = 1:prod(size(hline))`
	See the same message and explanation reported for line 23. Change the line 29 to
	
	`for nl = 1:numel(hline)`

`34:  'data' might be growing inside a loop.  Consider preallocating for speed.`
 	
— — — — — — — — — — — — — — — —
 	
`33 for nd = 1:length(fdata)`
`34 data{nd} = getfield(flds,fdata{nd});`
	See the same message and explanation reported for line 24. Add this line, 34, before the loop
	
	`data = cell(size(fdata));`

`34:  Use dynamic fieldnames with structures instead of GETFIELD. Type 'doc struct' for more information.`
 	
— — — — — — — — — — — — — — — —
 	
`34 data{nd} = getfield(flds,fdata{nd});`
	You can access a field in a structure as a variable expression that MATLAB evaluates at run-time. This is more efficient than using `getfield`. For more information, type `doc struct` to see the reference page for structures, or see in the MATLAB Programming documentation. Change line 37 to
	
	`data{nd} = flds.(fdata{nd});`

Message — Code (Original Line Numbers)	Explanation and Updated Code (New Line Numbers)

38: Use || instead of | as the OR operator in (scalar) conditional statements.
 	
39: Use || instead of | as the OR operator in (scalar) conditional statements.
 	
40: Use || instead of | as the OR operator in (scalar) conditional statements.
 	
————————————————————
 	
38 if isempty(data{3}) | ...
 	
39 (length(unique(data{1}(:)))==1 | ...
 	
40 length(unique(data{2}(:)))==1 | ...
 	
41 length(unique(data{3}(:)))==1)
	While | (the element-wise logical OR operator) performs the comparison correctly, use the || (short circuit OR operator) for efficiency. For details, see "Logical Operators" in the MATLAB Programming documentation. Change lines 40, 41, and 42 to
	
	`if isempty(data{3}) || ...`
	
	`(length(unique(data{1}(:)))==1 || ...`
	
	`length(unique(data{2}(:)))==1 || ...`

42: 'data' might be growing inside a loop. Consider preallocating for speed. ———————————————————— 42 data{3} = zeros(size(data{1}));	This message no longer appears due to the change made to line 34 `data{nd} = getfield(flds,fdata{nd});`. Sometimes fixing code in one line automatically clears a message for another line. If the reason for a message or the action to take for a message is not obvious at first, it could be because another line is causing the message. Address the issues that are easy to fix first and rerun the report. Do not make any changes to line 44.
43: 'dim' might be growing inside a loop. Consider preallocating for speed. 43 dim(nl) = 2;	See the same message and explanation reported for line 24. Add this line before the first line of the loop `dim = len;`

Message — Code (Original Line Numbers)	Explanation and Updated Code (New Line Numbers)
48: There may be a parenthesis imbalance around here. 48: There may be a parenthesis imbalance around here. 48: There may be a parenthesis imbalance around here. 48: There may be a parenthesis imbalance around here.	There is an error in this line, which you would see by running `lengthofline`. M-Lint suggests that it might be due to a parenthesis imbalance. You can check that by moving the arrow key over each of the delimiters, to see if MATLAB indicates a mismatch. This requires that **File > Preferences > Keyboard > Delimiter Matching** has the **Match on arrow key** option selected. There are no mismatched delimiters. The actual problem is the semicolon in parentheses, `data{3}(:)` is incorrect and should be a colon. In line 51, change `data{3}(;)` to `data{3}(:)`. That single change addressed the issues in all the messages for that line.
49: Terminate statement with semicolon to suppress output (in functions).	Adding a semicolon to the end of a statement suppresses output and is a common practice. M-Lint alerts you to lines that produce output but lack the terminating semicolon. If you want to view output from this line, do not add the semicolon. You can instruct M-Lint to ignore all messages on this line so that the messages on it will not appear by adding `%#ok` to the end of the line. However, because there is currently another message on the line, do not add `%#ok` until you have addressed the other message. Alternatively, you can add `%#ok` with the message ID for the specific message you want to suppress. To determine the message ID, run `mlint('lengthofline.m', '-id')`, which indicates the ID is `NOPRT`—for more information, see the `mlint` function reference page. For this example, assume you want to display the output and suppress the M-Lint message.

Message — Code (Original Line Numbers)	Explanation and Updated Code (New Line Numbers)
	To do so, add %#ok<NOPRT> to the end of the line. Note that there is a similar message for M-file scripts. This is so you can suppress the message for M-files that are cell-mode scripts, because they are often intended as demos and the display of output is intentional.
49: Use of brackets [] is unnecessary. Use parentheses to group, if needed. — — — — — — — — — — — — — — — — — — — 49 len(nl) = sum([sqrt(dot(temp',temp'))])	For more information about the use of brackets and parentheses, see the Special Characters reference page. In this example, remove the brackets because they are not needed. They add processing time because MATLAB concatenates unnecessarily. Change line 52 to len(nl) = sum(sqrt(dot(temp',temp'))) %#ok

Updated M-Lint Code Check Report after making changes to the lenghtofline file based on M-Lint messages. Now, no messages are reported.
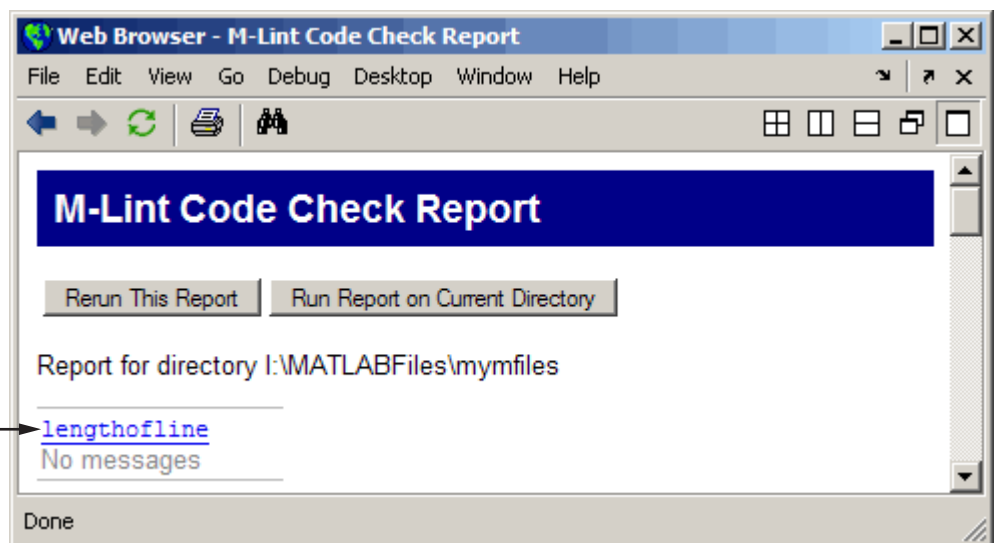
The M-file that includes all of the changes suggested by M-Lint is `lengthofline2.m`. To view it, issue the following in the Command Window:

```
edit(fullfile(matlabroot,'help','techdoc','matlab_env', ...
'examples','lengthofline2.m'))
```

Other Ways to Access M-Lint

You can get M-Lint messages using any of the following methods. Each provides the same M-Lint messages, but in a different format:

- Access the M-Lint Code Check report for an M-file from the Editor Tools menu or from the Profiler detail report.

- Run the `mlint` function, which analyzes the specified file and displays messages in the Command Window, or `mlintrpt`, which runs `mlint` and displays the messages in the Web Browser.

- Use automatic M-Lint analysis and code correction while you work on a file in the Editor—see "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 6-123.

Profiling for Improving Performance

In this section...

What Is Profiling?

Profiling is a way to measure where a program spends time. To assist you in profiling, MATLAB software provides a graphical user interface, called the Profiler, which is based on the results returned by the `profile` function. Once you identify which functions are consuming the most time, you can determine why you are calling them and look for ways to minimize their use and thus improve performance. It is often helpful to decide whether the number of times a particular function is called is reasonable. Because programs often have several layers, your code may not explicitly call the most time-consuming functions. Rather, functions within your code might be calling other time-consuming functions that can be several layers down in the code. In this case it is important to determine which of your functions are responsible for such calls.

Profiling helps to uncover performance problems that you can solve by:

- Avoiding unnecessary computation, which can arise from oversight
- Changing your algorithm to avoid costly functions
- Avoiding recomputation by storing results for future use

When you reach the point where most of the time is spent on calls to a small number of built-in functions, you have probably optimized the code as much as you can expect.

Note When using the Parallel Computing Toolbox™ software, you can use the parallel profiler to profile parallel jobs. See "Using the Parallel Profiler" for details.

Profiling Process and Guidelines

Here is a general process you can follow to use the Profiler to improve performance in your M-files. This section also describes how you can use profiling as a debugging tool and as a way to understand complex M-files.

Note Premature optimization can increase code complexity unnecessarily without providing a real gain in performance. Your first implementation should be as simple as possible. Then, if speed is an issue, use profiling to identify bottlenecks.

1 In the summary report produced by the Profiler, look for functions that used a significant amount of time or were called most frequently. See "Profile Summary Report" on page 7-41 for more information.

2 View the detail report produced by the Profiler for those functions and look for the lines that use the most time or are called most often. See "Profile Detail Report" on page 7-43 for more information.

You might want to keep a copy of your first detail report to use as a reference to compare with after you make changes, and then profile again.

3 Determine whether there are changes you can make to the lines most called or the most time-consuming lines to improve performance.

For example, if you have a `load` statement within a loop, `load` is called every time the loop is called. You might be able to save time by moving the `load` statement so it is before the loop and therefore is only called once.

4 Click the links to the files and make the changes you identified for potential performance improvement. Save the files and run `clear all`. Run the Profiler again and compare the results to the original report. Note that there are inherent time fluctuations that are not dependent on your code.

If you profile the exact same code twice, you can get slightly different results each time.

5 Repeat this process to continue improving the performance.

Using Profiling as a Debugging Tool

The Profiler is a useful tool for isolating problems in your M-files.

For example, if a particular section of the file did not run, you can look at the detail reports to see what lines did run, which might point you to the problem.

You can also view the lines that did not run to help you develop test cases that exercise that code.

If you get an error in the M-file when profiling, the Profiler provides partial results in the reports. You can see what ran and what did not to help you isolate the problem. Similarly, you can do this if you stop the execution using **Ctrl+C**, which might be useful when a file is taking much more time to run than expected.

Using Profiling for Understanding an M-File

For lengthy M-files that you did not create or that you have not used for awhile and are unfamiliar with, you can use the Profiler to see how the M-file actually worked. Use the Profiler detail reports to see the lines actually called.

If there is an existing GUI tool (or M-file) similar to one that you want to create, start profiling, use the tool, then stop profiling. Look through the Profiler detail reports to see what functions and lines ran. This helps you determine the lines of code in the file that are most like the code you want to create.

Using the Profiler

Use the Profiler to help you determine where you can modify your code to make performance improvements. The Profiler is a tool that shows you where an M-file is spending its time. This section covers

• "Opening the Profiler" on page 7-36

- "Running the Profiler" on page 7-36
- "Profiling a Graphical User Interface" on page 7-40
- "Profiling Statements from the Command Window" on page 7-40
- "Changing Fonts for the Profiler" on page 7-41

For information about the reports generated by the Profiler, see "Profile Summary Report" on page 7-41 and "Profile Detail Report" on page 7-43.

Opening the Profiler

You can use any of the following methods to open the Profiler:

- Select **Desktop > Profiler** from the MATLAB desktop.
- Click the Profiler button ⊟ in the MATLAB desktop toolbar.
- With a file open in the MATLAB Editor, select **Tools > Open Profiler**.
- Select one or more statements in the Command History window, right-click to view the context menu, and choose **Profile Code**.
- Enter the following function in the Command Window:

  ```
  profile viewer
  ```

Running the Profiler

The following illustration summarizes the typical steps for profiling.

1. Type profile viewer to open the Profiler.

2. Type the statement to run.

3. Click Start profiling.
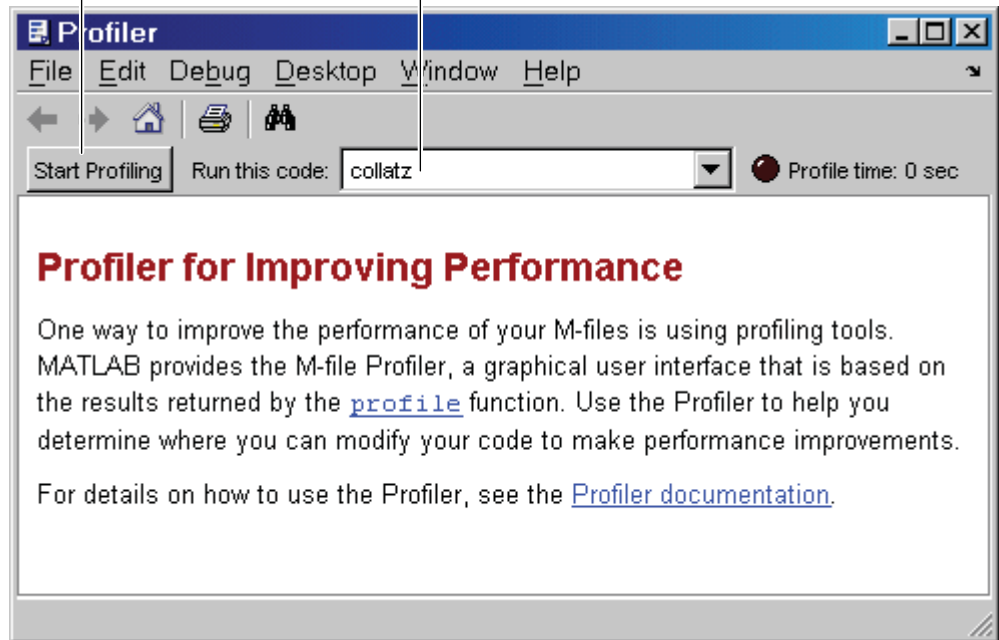
To profile an M-file or a line of code, follow these steps:

1 If your system uses Intel® multi-core chips, you may want to restrict the active number of CPUs to one. See "Intel Multi-Core Processors — Setting for Most Accurate Profiling" on page 7-39 for details on how to do this.

2 In the **Run this code** field in the Profiler, type the statement you want to run.

You can run this example

```
[t,y] = ode23('lotka',[0 2],[20;20])
```

as the code is provided with MATLAB demos. It runs the Lotka-Volterra predator-prey population model. For more information about this model, type `lotkademo`, which runs the demonstration.

To run a statement you previously profiled in the current MATLAB session, select the statement from the list box—MATLAB automatically starts profiling the code, so skip to step 3.

3 Click **Start Profiling** (or press **Enter** after typing the statement).

While the Profiler is running, the **Profile time** indicator (at the top right of the Profiler window) is green and the number of seconds it reports increases.
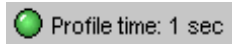
○ Profile time: 1 sec

When the profiling is finished, the **Profile time** indicator becomes dark red and shows the length of time the Profiler ran. The statements you profiled are shown as having been executed in the Command Window.

● Profile time: 6 sec
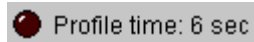
This is not the actual time that your statements took to run; it is the wall clock (or `tic`/`toc`) time elapsed from when you clicked **Start Profiling** until profiling stops. If the time reported is much different from what you expected (for example, hundreds of seconds for a simple statement), you might have had profiling on longer than you realized. This time also does not match the time reported in Profiler Summary report statistics, which is based on `cpu` time by default, not wall clock time. To view profile statistics based on wall clock time, use the `profile` function with the `-timer real` option as shown in "Using the profile Function to Change the Time Type Used by the Profiler" on page 7-54.

4 When profiling is complete, the Profile Summary report appears in the Profiler window. For more information about this report, see "Profile Summary Report" on page 7-41.

5 If you restricted the number of active CPUs in Step 1, reset the number of active CPUs to the original setting.

Intel Multi-Core Processors — Setting for Most Accurate Profiling. If your system uses Intel multicore chips, and you plan to profile using CPU time, set the number of active CPUs to one before you start profiling. This results in the most accurate and efficient profiling.

1 Open Windows Task Manager.

2 On the **Processes** tab, right-click MATLAB.exe and then click **Set Affinity**.

The Processor Affinity dialog box opens.

3 In the Processor Affinity dialog box, note the current settings, and then clear all the CPUs except one.

Your Processor Affinity dialog box should appear similar to the following image:
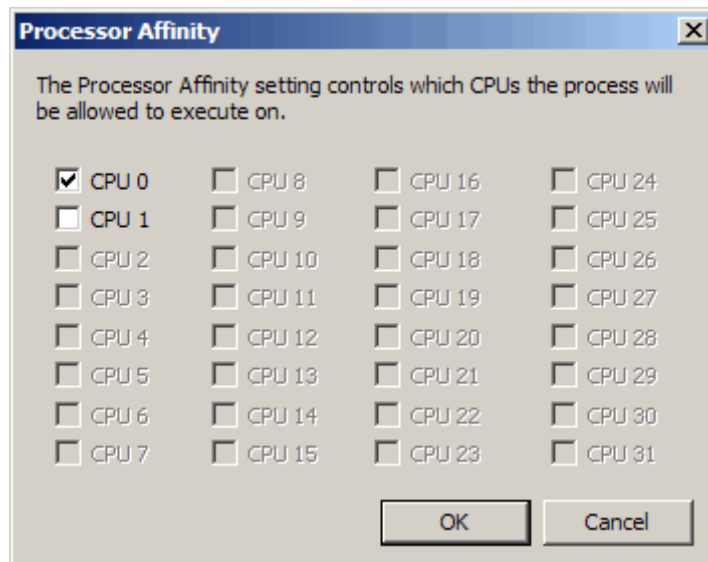
4 Click **OK**.

5 Reset the state of the Profiler so that it recognizes the processor affinity changes you made. The easiest way to do this is to change the Profiler

timer setting from `real` and then back to `cpu`, by issuing the following in the Command Window:

```
profile -timer real
profile -timer cpu
```

Remember to set the number of CPUs back to their original settings when you finish profiling by rerunning the preceding steps, but restoring the original selections in the Processor Affinity dialog box in Step 3.

Note Setting the number of computational threads to 1 in the General Multithreading Preferences dialog box does not have the same effect as setting the Processor Affinity to one CPU.

Profiling a Graphical User Interface

You can run the Profiler for a graphical user interface, such as the Filter Design and Analysis tool included with Signal Processing Toolbox. You can also run the Profiler for an interface you created, such as one built using GUIDE.

To profile a graphical user interface, follow these steps:

1 In the Profiler, click **Start Profiling**. Make sure that no code appears in the **Run this code** field.

2 Start the graphical user interface. (If you do not want to include its startup process in the profile, do not click **Start Profiling**, step 1, until after you have started the graphical interface.)

3 Use the graphical interface. When you are finished, click **Stop Profiling** in the Profiler.

The Profile Summary report appears in the Profiler.

Profiling Statements from the Command Window

To profile more than one statement, follow these steps:

1 In the Profiler, clear the **Run this code** field and click **Start Profiling**.

2 In the Command Window, enter and run the statements you want to profile.

3 After running all the statements, click **Stop Profiling** in the Profiler.

The Profile Summary report appears in the Profiler.

Changing Fonts for the Profiler

To change the fonts used in the Profiler, follow these steps:

1 Select **File > Preferences > Fonts** to open the **Font** Preferences dialog box.

2 In the Font Preferences dialog box, select the code or text font that you want to use in the Profiler. The Profiler is an HTML Proportional Text tool. For more information, click the **Help** button in the dialog box.

3 Click **Apply** or **OK**. The Profiler font reflects the changes.

Profile Summary Report

The Profile Summary report presents statistics about the overall execution of the function and provides summary statistics for each function called. The report formats these values in four columns.

- **Function Name** — A list of all the functions and subfunctions called by the profiled function. When first displayed, the functions are listed in order by the amount of time they took to process. To sort the functions alphabetically, click the **Function Name** link at the top of the column.

- **Calls** — The number of times the function was called while profiling was on. To sort the report by the number of times functions were called, click the **Calls** link at the top of the column.

- **Total Time** — The total time spent in a function, including all child functions called, in seconds. The time for a function includes time spent on child functions. To sort the functions by the amount of time they consumed, click the **Total Time** link at the top of the column. By default, the summary report displays profiling information sorted by **Total Time**. Note that the Profiler itself uses some time, which is included in the

results. Also note that total time can be zero for files whose running time was inconsequential.

- **Self Time** — The total time spent in a function, *not* including time for any child functions called, in seconds. If MATLAB can determine the amount of time spent for profiling overhead, MATLAB excludes it from the self time also. (MATLAB excludes profiling overhead from the total time and the time for individual lines in the Profile Detail Report as well.)

  The bottom of the Profiler page contains a message similar to one of the following, depending on whether MATLAB can determine the profiling overhead:

  - Self time is the time spent in a function excluding both the time spent in its child functions and most of the overhead resulting from the process of profiling. In the present run, self time excludes 0.240 seconds of profiling overhead. The amount of remaining overhead reflected in self time cannot be determined, and therefore is not excluded.

  - Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes some overhead resulting from the process of profiling.

  To sort the functions by this time value, click the **Self Time** link at the top of the column.

- **Total Time Plot** — Graphic display showing self time compared to total time.

Following is the summary report for the Lotka-Volterra model described in "Example: Using the profile Function" on page 7-51.

To print a summary report, click the Print button 🖨.

To get more detailed information about a particular function, click its name in the **Function Name** column. See "Profile Detail Report" on page 7-43 for more information.

Click any column label to sort by that column.

Click any function name to display the detailed report.

Profile Detail Report

The Profile Detail report shows profiling results for a selected function that was called during profiling. A Profile Detail report is made up of seven sections, summarized below. By default, the Profile Detail report includes

all seven sections, although, depending on the function, not every section contains data. To return to the Profile Summary report from the Profile Detail report, click the Home button 🏠 in the toolbar. The following sections provide more detail:

- "Controlling the Contents of the Detail Report Display" on page 7-44
- "Profile Detail Report Header" on page 7-46
- "Parent Functions" on page 7-46
- "Busy Lines" on page 7-47
- "Child Functions" on page 7-47
- "M-Lint Results" on page 7-48
- "File Coverage" on page 7-48
- "Function Listing" on page 7-49

Controlling the Contents of the Detail Report Display

You can determine which sections are included in the display by selecting them and then clicking the **Refresh** button. The following sections provide more detail about each section of this report.

Select report options to display, and then click Refresh.

Profiler — window

Start Profiling | Run this code: [t,y] = ode23('lotka',[0 2],[20;20]) ▼ ● Profile time: 1 sec

funfun\private\odearguments (1 call, 0.259 sec)

Generated 15-Jan-2006 06:13:27 using cpu time.
M-function in file
\\mathworks\devel\bat\R2006adnightly\matlab\toolbox\matlab\funfun\private\odearguments.m
[Copy to new window for comparing multiple runs]

Refresh

☑ Show parent functions ☐ Show busy lines ☑ Show child functions
☑ Show M-Lint results ☑ Show file coverage ☑ Show function listing

Parents (calling functions)

Function Name	Function Type	Calls
ode23	M-function	1

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
lotka	M-function	1	0.105 s	40.3%	▆
odeget	M-function	5	0.013 s	4.9%	▪
double.superiorfloat	M-function	1	0.000 s	0.1%	
Self time (built-ins, overhead, etc.)			0.142 s	54.7%	▆
Totals			0.259 s	100%	

M-Lint results

Line number	Message
51	EXIST with two input arguments is faster than with one input.
79	EXIST with two input arguments is faster than with one input.

Coverage results
[Show coverage for parent directory]

Total lines in file	188
Non-code lines (comments, blank lines)	51

Profile Detail Report Header

The detail report header includes the name of the function that was profiled, the number of times it was called in the parent function, and the amount of time it used.

The header includes a link that opens the function in your default text editor.

The header also includes a link that copies the report to a separate window. Creating a copy of the report can be helpful when you make changes to the file, run the Profiler for the updated file, and compare the Profile Detail reports for the two runs. Do not make changes to M-files provided with products from The MathWorks, that is, files in *matlabroot*/toolbox directories.

Open file in default editor.———————

funfun\private\odearguments (1 call, 0.259 sec)
Generated 15-Jan-2006 06:13:27 using cpu time.
M-function in file
\\mathworks\devel\bat\R2006adnightly\matlab\toolbox\matlab\funfun\private\odearguments.m
[Copy to new window for comparing multiple runs]

Copy this detail report to a new window.

Parent Functions

To include the **Parents** section in the detail report, select the **Show parent functions** check box. This section of the report provides information about the parent functions, with links to their detail reports.

	Parents (calling functions)		
	Function Name	Function Type	Calls
Click to open parent detail report.	ode23	M-function	1

Busy Lines

To include information about the lines of code that used the most amount of processing time in the detail report, select the **Show busy lines** check box. Note that this was not selected in the example. Click a line number to view that line of code in the source listing.

Click a line number to go to that line in the file.

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
110	f0 = feval(ode,t0,y0,args{:}); ...	1	0.105 s	41.6%	
146	rtol = odeget(options,'RelTol'...	1	0.067 s	25.8%	
80	if (nargin(ode) == 2) ...	1	0.018 s	7.2%	
94	end	1	0.009 s	3.6%	
79	if (exist(ode)==2)	1	0.005 s	1.9%	
Other lines & overhead			0.054 s	20.9%	
Totals			0.253 s	100%	

Child Functions

To include the **Children** section of the detail report, select the **Show child functions** check box. This section of the report lists all the functions called by the profiled function. If the called function is an M-file, you can view the source code for the function by clicking its name.

Click to view detail report for functions.

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
lotka	M-function	1	0.105 s	40.3%	▬▬▬
odeget	M-function	5	0.013 s	4.9%	▪
double.superiorfloat	M-function	1	0.000 s	0.1%	
Self time (built-ins, overhead, etc.)			0.142 s	54.7%	▬▬▬▬
Totals			0.259 s	100%	

M-Lint Results

To include the **M-Lint results** section in the detail report display, select the **Show M-Lint results** check box. This section of the report provides information about problems and potential improvements, generated by M-Lint about the function. For more information about M-Lint, see "M-Lint Code Check Report" on page 7-21.

M-Lint results

Click a line number to go to line in code.

Line number	Message
51	EXIST with two input arguments is faster than with one input.
79	EXIST with two input arguments is faster than with one input.

File Coverage

To include the **Coverage results** section in the detail report display, select the **Show file coverage** check box. This section of the report provides statistical information about the number of lines in the code that executed during the profile run.

Coverage results [<u>Show coverage for parent directory</u>]	
Total lines in file	188
Non-code lines (comments, blank lines)	51
Code lines (lines that can run)	137
Code lines that did run	53
Code lines that did not run	84
Coverage (did run/can run)	38.69 %

Percentage of file that executed during profile run. ———— Coverage (did run/can run)

Function Listing

To include the **Function listing** section in the detail report display, select the **Show function listing** check box. If the file is an M-file, the Profile Detail report includes a column listing the execution time for each line, a column listing the number of times the line was called, and the source code for the function.

In the function listing, comment lines appear in green, lines of code that executed appear in black, and lines of code that did not execute appear in gray. If you click a function name in the listing, you can view its detail report.

By default, the Profile Detail report uses the color red to highlight the lines of code with the longest execution time. The darker the shade of red, the longer the line of code took to execute. Using the menu in this section of the detail report you can change this default and choose to highlight lines of code based on other criteria such as the lines called the most, lines called out by M-Lint, or lines of code that were (or were not) executed. Using this menu, you can also turn off highlighting completely.

The profile Function

The Profiler is based on the results returned by the `profile` function. The `profile` function provides some features that are not available in the GUI. For example, using the function, you can specify that the statistics display the time it takes for statements to run as clock time instead of CPU time.

This section includes the following topics with respect to the `profile` function:

- "Example: Using the profile Function" on page 7-51
- "Accessing profile Function Results" on page 7-52
- "Saving profile Function Reports" on page 7-54
- "Using the profile Function to Change the Time Type Used by the Profiler" on page 7-54

Example: Using the profile Function

This example demonstrates how to run `profile`:

1 To start `profile`, type in the Command Window

```
profile on
```

2 Execute an M-file. This example runs the Lotka-Volterra predator-prey population model. For more information about this model, type `lotkademo`, which runs a demonstration.

```
[t,y] = ode23('lotka',[0 2],[20;20]);
```

3 Generate the profile report and display it in the Profiler window. This suspends `profile`.

```
profile viewer
```

4 Restart `profile`, without clearing the existing statistics.

```
profile resume
```

The `profile` function is now ready to continue gathering statistics for any more M-files you run. It will add these new statistics to those generated in the previous steps.

5 Stop `profile` when you finish gathering statistics.

```
profile off
```

6 To view the profile data, call `profile` specifying the `'info'` argument. The profile function returns data in a structure.

```
p = profile('info')

p =
        FunctionTable: [10x1 struct]
      FunctionHistory: [2x0 double]
       ClockPrecision: 3.3333e-010
           ClockSpeed: 3.0000e+009
                 Name: 'MATLAB'
```

The `FunctionTable` indicates that statistics were gathered for 10 functions.

7 To save the profile report, use the `profsave` function. This function stores the profile information in separate HTML files, for each function listed in `FunctionTable` of `p`.

```
profsave(p)
```

By default, `profsave` puts these HTML files in a subdirectory of the current directory named `profile_results`, and displays the summary report in your system browser. You can specify another directory name as an optional second argument to `profsave`.

Accessing profile Function Results

The `profile` function returns results in a structure. This example illustrates how you can access these results:

1 To start `profile`, specifying the `history` option, type in the Command Window.

```
profile on -history
```

The history option specifies that the report include information about the sequence of functions as they are entered and exited during profiling.

2 Execute an M-file. This example runs the Lotka-Volterra predator-prey population model. For more information about this model, type `lotkademo`, which runs a demonstration.

```
[t,y] = ode23('lotka',[0 2],[20;20]);
```

3 Stop profiling.

```
profile off
```

4 Get the structure containing profile results.

```
stats = profile('info')
stats =
        FunctionTable: [43x1 struct]
```

```
            FunctionHistory: [2x754 double]
             ClockPrecision: 3.3333e-010
                 ClockSpeed: 3.0000e+009
                       Name: 'MATLAB'
```

5 The FunctionTable field is an array of structures, where each structure
represents an M-function, M-subfunction, MEX-function, or, because the
builtin option is specified, a MATLAB built-in function.

```
stats.FunctionTable

ans =

41x1 struct array with fields:
    CompleteName
    FunctionName
    FileName
    Type
    NumCalls
    TotalTime
    TotalRecursiveTime
    Children
    Parents
    ExecutedLines
    IsRecursive
    PartialData
```

6 View the second structure in FunctionTable.

```
stats.FunctionTable(2)

ans =
              CompleteName: [1x79 char]
              FunctionName: 'ode23'
                  FileName: [1x73 char]
                      Type: 'M-function'
                  NumCalls: 1
                 TotalTime: 0.3902
        TotalRecursiveTime: 0
                  Children: [20x1 struct]
```

```
                   Parents: [0x1 struct]
            ExecutedLines: [139x3 double]
              IsRecursive: 0
              PartialData: 0
```

7 To view the history data generated by `profile`, view the `FunctionHistory`, for example, `stats.FunctionHistory`. The history data is a 2-by-n array. The first row contains Boolean values, where `0` (zero) means entrance into a function and `1` means exit from a function. The second row identifies the function being entered or exited by its index in the `FunctionTable` field. To see how to create a formatted display of history data, see the example on the `profile` reference page.

Saving profile Function Reports

To save the profile report, use the `profsave` function.

This function stores the profile information in separate HTML files, for each function listed in the `FunctionTable` field of the structure, `stats`.

```
profsave(stats)
```

By default, `profsave` puts these HTML files in a subdirectory of the current directory named `profile_results`. You can specify another directory name as an optional second argument to `profsave`.

```
profsave(stats,'mydir')
```

Using the profile Function to Change the Time Type Used by the Profiler

By default, MATLAB generates the Profiler Summary report using CPU time, as opposed to real (wall clock) time. This example illustrates how you can direct MATLAB to use real time instead.

The following image shows the Profiler Summary report as it appears by default, using CPU time:
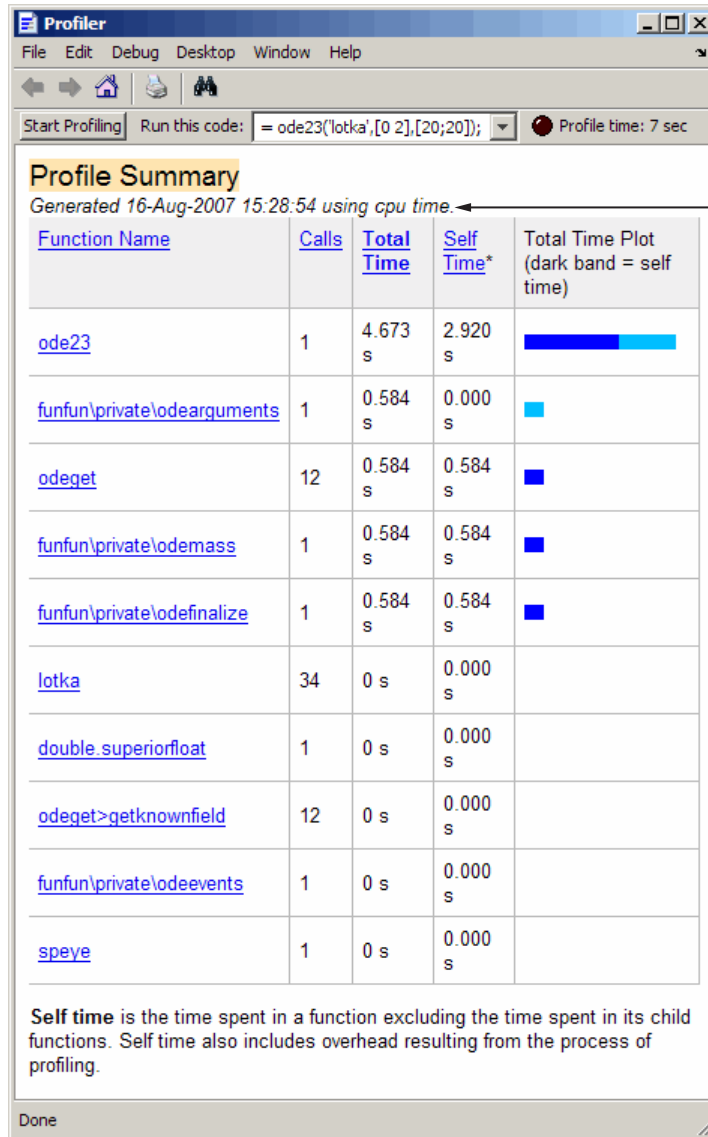
Specify that the Profiler use real time instead, by using the `profile` function with the `-timer real` option, as shown in this example:

1 If the Profiler is currently open, close the Profiler, and if prompted, stop profiling.

2 Set the timer to real time by typing the following in the Command Window:

```
profile on -timer real
```

3 Run the M-file that you want to profile. This example runs the Lotka-Volterra predator-prey population model.

```
[t,y] = ode23('lotka',[0 2],[20;20]);
```

4 Open the Profiler by typing the following in the Command Window:

```
profile viewer
```

The Profiler opens and indicates that real time is being used, as shown in the following image:

Generated using real time
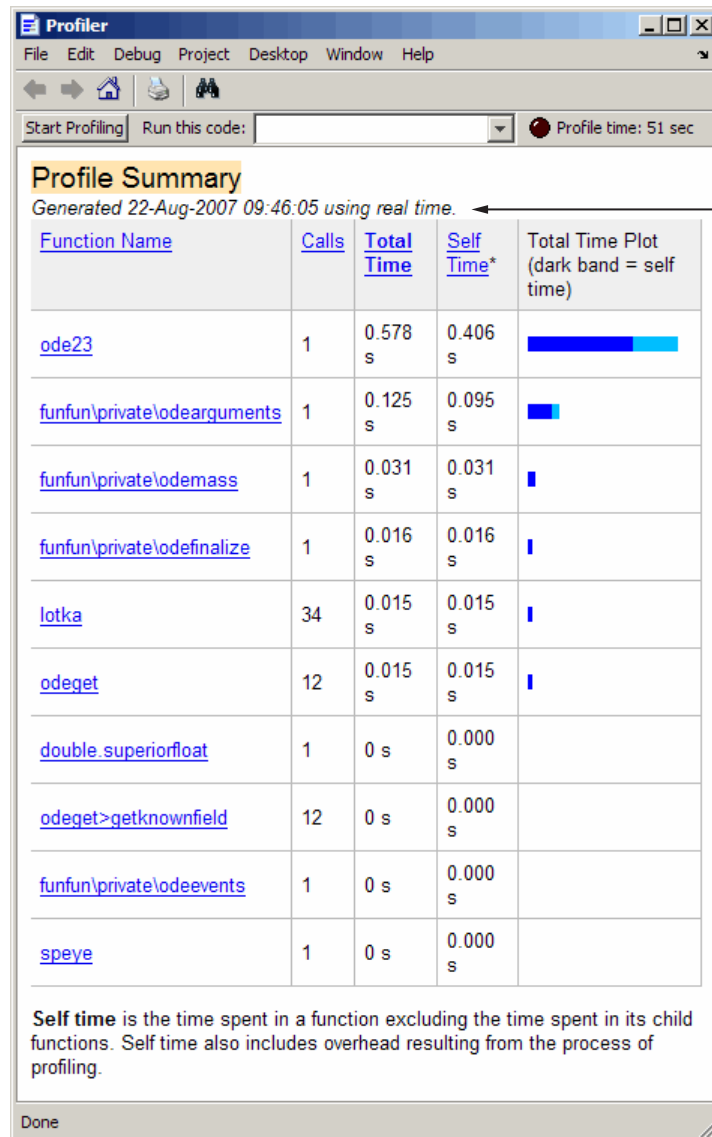
5 To change the timer back to using CPU time:

   a Close the Profiler, and if prompted, stop profiling.

b Type the following in the Command Window:

```
profile on -timer cpu
```

c Type the following in the Command Window to reopen the Profiler:

```
profile viewer
```

8

Publishing M-Files

MATLAB software lets you to format M-files and publish them to various output types.

- "Overview of Publishing M-Files" on page 8-2
- "Formatting M-File Comments for Publishing" on page 8-18
- "Formatting M-File Code for Publishing" on page 8-64
- "Producing Published Output from M-Files" on page 8-68

Overview of Publishing M-Files

In this section...
"What Is Meant by Publishing M-Files?" on page 8-2
"Using Cells" on page 8-2
"Process for Publishing M-Files" on page 8-3
"Example of a Published M-File" on page 8-4
"Producing the Formatting for the Example" on page 8-11

What Is Meant by Publishing M-Files?

The MATLAB product allows you to quickly publish your M-file code to enable you to describe and share your code with others who may or may not have MATLAB software. You can include the following within the published M-file:

- Formatted commentary on the code, including bulleted and numbered lists, bold and monospace font, preformatted text, TeX equations, and so on

- M-file code

- Results of running the code, including output to the Command Window and figures created or modified by the code

You can publish in a variety of formats, including HTML, XML, and LaTeX. If Microsoft Word or Microsoft® PowerPoint® applications are on your Microsoft Windows system, you can publish to their formats as well.

If you have an active Internet connection, you can watch the Publishing M Code from the Editor video demo for an overview of the major publishing features using cells with text markup.

Using Cells

After you write and debug an M-file and are ready to share it with others, you can insert cells and commentary using text markup features available in MATLAB. This enables you to publish a formatted M-file. A cell is a section of M-file code (see "What Are Cells?" on page 6-185). For the purposes of publishing, a cell can be a section of the code that you want to present as a

titled subsection within the published document, or a portion of code for which you want the results of code evaluation to display as it occurs (for example, each iteration of a `for` loop), or both.

Any cell features that you use for evaluating and improving your code, as described in "Using Cells for Rapid Code Iteration and Publishing Results" on page 6-185, you also can use for publishing purposes. However, to have formatted comments in the output document, those comments must appear at the start of a cell, before any executable code. This requirement might mean that you need to change cells that you inserted for rapid code iteration. If you do so, be aware that this changes the cells for evaluation purposes, as well.

"Example of a Published M-File" on page 8-4 shows how the cells and formatted comments appear when an M-file is published.

Although you typically include the text markup after you write and debug the code, you can also include text markup as you write the code, or a combination of the two.

Note Cell mode is supported for use with M-files only. It is not intended for use with plain text files.

Process for Publishing M-Files

The overall process to publish an M-file using cell features in the Editor is as follows:

1 Open your M-file in the Editor.

2 Select **Cell > Insert Text Markup** as described in "Formatting M-File Comments for Publishing" on page 8-18.

This enables you to specify how M-file comments appear in the published document. For example, you can specify that comments appear as bold or monospaced text in the published document.

3 To publish the M-file, do one of the following, as described in "Producing Published Output from M-Files" on page 8-68:

- To publish the M-file with default publishing properties, select **File > Publish *file name***. When you use this method, MATLAB publishes the M-file to HTML in an `/html` subdirectory of the directory that contains the M-file you are publishing. However, if you previously specified custom property values, as described in the next list item, the last configuration you specified is used and the output type and directory may be different.

- To specify custom publishing properties, select **File > Publish Configuration for *file name* > Edit Publish Configurations for *file name***, adjust properties, and then click **Publish**. You can, for example, choose to include or exclude the executable code from the published document.

Example of a Published M-File

This section provides an example to demonstrate how an M-file appears when published. It shows how the M-file appears before and after text markup is added to cells to achieve the formatted results. This section contains the following topics:

- "Sample M-File Before Formatting" on page 8-4
- "Published Sample M-File Before Formatting" on page 8-5
- "Published Sample M-File After Formatting" on page 8-7

For detailed information on inserting text markup, see "Formatting M-File Comments for Publishing" on page 8-18.

Sample M-File Before Formatting

```
function fourier_demo
    t = 0:.1:pi*4;
    y = sin(t);
    updatePlot(t,y);

    % In each iteration of the for loop add an odd
    % harmonic to y. As "k" increases, the output
    % approximates a square wave with increasing accuracy.

    for k = 3:2:9
```

```
        % Perform the following mathematical operation
        % at each iteration:
        y = y + sin(k*t)/k;

        display(sprintf('When k = %.1f',k));
        display('Then the plot is:');
        updatePlot (t,y)
    end

end

% Even though the approximations are constantly
% improving, they will never be exact because of the
% Gibbs phenomenon, or ringing.

function updatePlot(t,x)
% Subfunction to update the plot
    cla
    plot(t,x)
end
```

Published Sample M-File Before Formatting

Before you add text markup and cell breaks, publishing fourier_demo.m includes the last plot generated by the for loop, but otherwise, has little effect. For example, if you select **File > Publish fourier_demo.m**, the published results, as shown in the following figure, are of limited use.
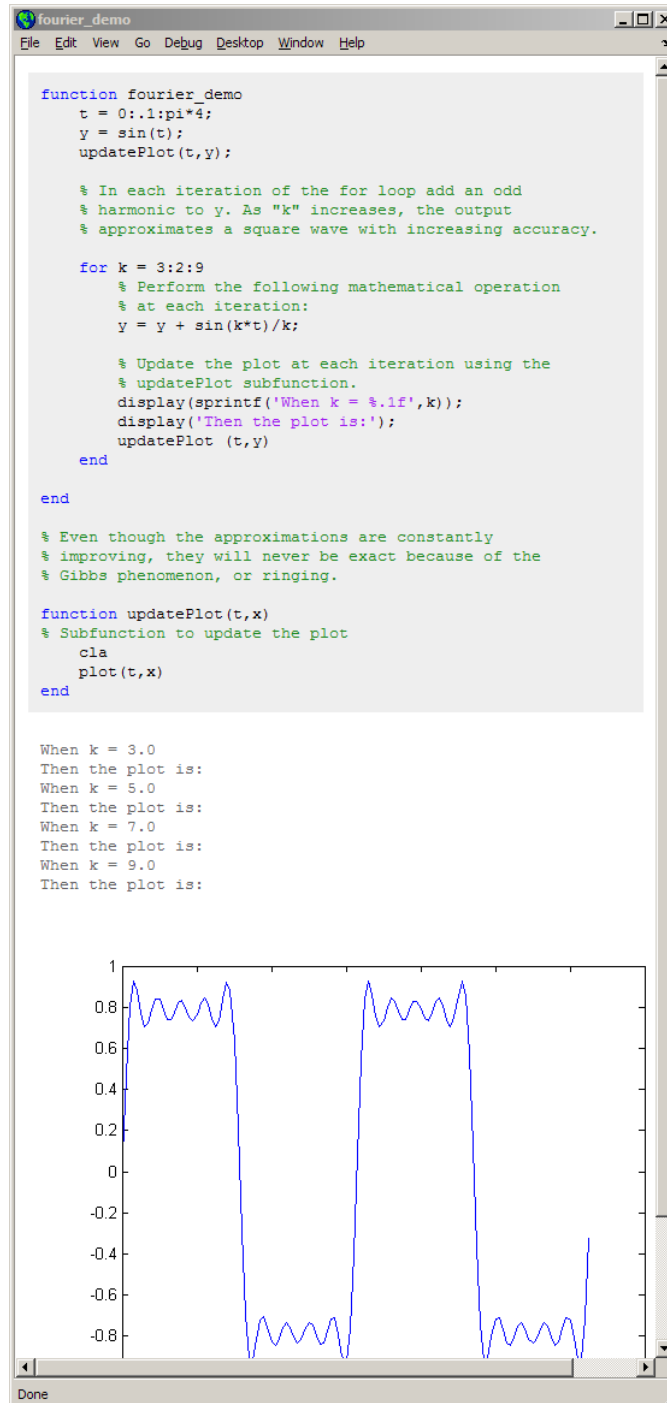
Published Sample M-File After Formatting

If you add a few comments for clarity, apply text markup, and insert cell breaks, as described in "Producing the Formatting for the Example" on page 8-11, the published M-file is transformed as shown in the following three figures:

- The first figure shows the top of the published document.
- The second figure shows the middle of the published document.
- The third figure shows the bottom of the document.

Add a title for the document.

Add a table of contents.

MATLAB code displays with a gray background to distinguish it from results.

Reduce the size of the figure in the document.

Web Browser - Square Waves from Sine Waves

File   Edit   View   Go   Debug   Desktop   Window   Help

Square Waves from Sine Waves

The Fourier series expansion for a square-wave is made up of a sum of odd harmonics, as shown here using MATLAB®.

Contents

- Add an Odd Harmonic and Plot It.
- Note About Gibbs Phenomenon.

Add an Odd Harmonic and Plot It.

```
function fourier_demo
```

```
t = 0:.1:pi*4;
y = sin(t);
updatePlot(t,y);
```

Done

Make selected comment text, such as the k here, appear in italic.

Publish an equation using TeX format.

Include each iteration of the for loop in the document.

Web Browser - Square Waves from Sine Waves

File Edit View Go Debug Desktop Window Help

In each iteration of the for loop add an odd harmonic to y. As *k* increases, the output approximates a square wave with increasing accuracy.

```
for k = 3:2:9
```

Perform the following mathematical operation at each iteration:

$$y = y + \frac{sin(k*t)}{k}$$

```
y = y + sin(k*t)/k;

display(sprintf('When k = %.1f',k));
display('Then the plot is:');
updatePlot (t,y)
```

```
When k = 3.0
Then the plot is:
```

```
When k = 5.0
Then the plot is:
```
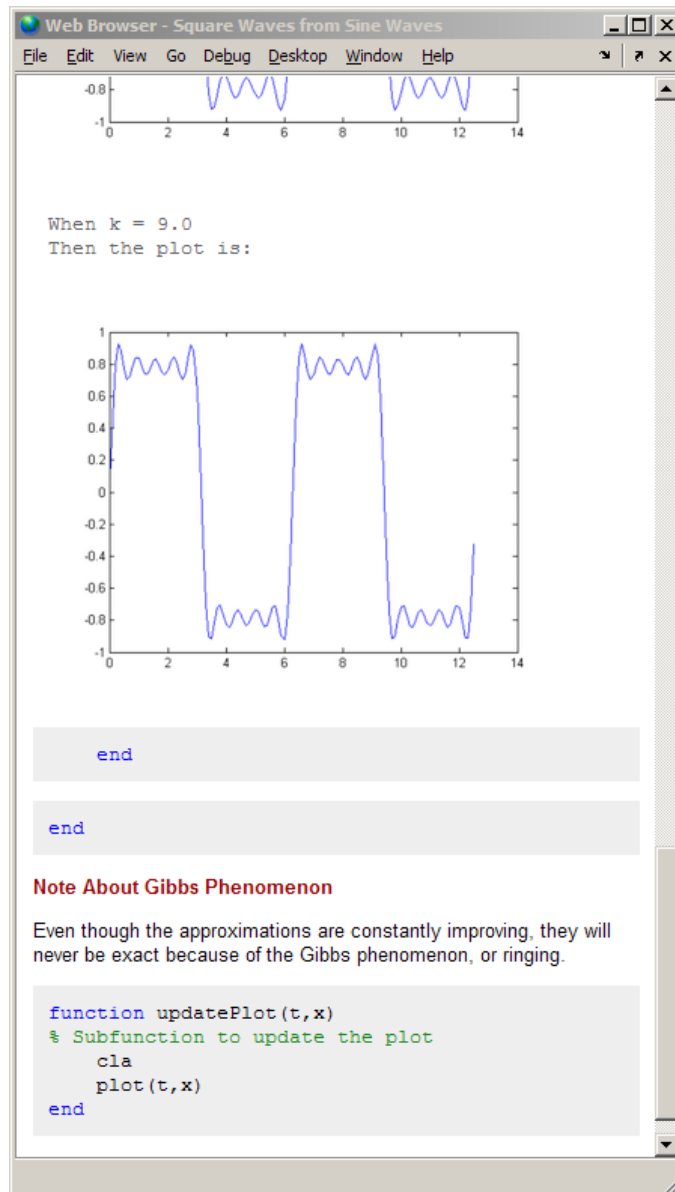
Producing the Formatting for the Example

The following steps apply text markup to the `fourier_demo.m` file. When published to HTML, the results appear as shown in "Published Sample M-File After Formatting" on page 8-7.

For detailed information about each **Cell** menu option, see "Formatting M-File Comments for Publishing" on page 8-18.

1 Open `fourier_demo.m` by running the following command:

```
edit(fullfile(matlabroot,'help','techdoc',...
'matlab_env','examples','fourier_demo.m'))
```

To work with `fourier_demo.m` on your system, save the file to a directory for which you have write permission. In the example, the file is saved to `I:\my_matlab_files\my_mfiles\fourier_demo.m`.

2 Enable cell mode by selecting **Cell > Enable Cell Mode.**

3 Add an overall title and introduction for the published document:

a Select **Cell > Insert Text Markup > Document Title and Introduction**. MATLAB adds the following at the top of the file:

```
%% DOCUMENT TITLE
% INTRODUCTORY TEXT
```

The double percent signs (`%%`) indicate the start of a new cell. A single percent sign indicates the beginning of a comment line.

b Replace `DOCUMENT TITLE` with `Square Waves from Sine Waves`.

c Replace `% INTRODUCTORY TEXT` with one or more comments about the overall file, for example:

```
% The Fourier series expansion for a square-wave is
% made up of a sum of odd harmonics, as shown here
% using MATLAB(R).
```

The string "(R)" will appear as a registered trademark symbol in the published document.

d On line 5, insert a blank line for better readability. Notice that the file now contains two cells. The first cell extends from line 6 to the top of the file; the second cells extends from line 6 to the bottom of the file. The cell break at line 6, splits the file into two cells.

4 On line 6, where the second cell begins (as indicated by %%), type a title for the cell: `Add an Odd Harmonic and Plot It`.

Notice that when you move from one cell to the next in the file, the highlighting in the M-file indicates which cell the cursor is currently in.

5 To display the text that describes the purpose of the loop in the published document as explanatory text, rather than M-file code, insert a cell break before the explanation. That is:

a Place the cursor at line 12.

b Select **Cell > Insert Cell Break**.

Note that the cell that begins on line 12, continues to the end of the fourier_demo function. If you insert a cell break anywhere within the code block, MATLAB inserts an implicit cell break at the end of a code block. A *code block* is the body of any programming control statement or function.

6 Remove the quotation marks around the k at line 14 and present it in italic instead:

a Delete the quotation marks.

b Select the letter k.

c Select **Cell > Insert Text Markup > Italic Text**.

Instead of being enclosed in quotation marks, the letter now appears as _k_.

d To see the effect, click Publish ▧.

7 Because MATLAB publishes output generated by code immediately after the end of the cell that contains the code, the current cell would cause MATLAB to include the phrase When k = n Then the plot is: four times in succession in the published document. In addition, only the final plot generated by the for loop would be in the published document.

To have MATLAB include every plot generated by the for loop in the published document, each preceded by the phrase When k = n ..., create a cell within the for loop, as follows:

a Place the cursor at the end of line 17, after for k = 3:2:9.

b Select **Cell > Insert Cell Break**.

Now the current cell includes only the body of the for loop.

```
H:\Documents\my_matlab_files\my_mfiles\fourier_demo.m*          _ □ ×
File  Edit  Text  Go  Cell  Tools  Debug  Desktop  Window  Help          ↘

13        % In each iteration of the for loop add an odd
14        % harmonic to y. As _k_ increases, the output
15        % approximates a square wave with increasing accuracy
16
17        for k = 3:2:9
18            %%
19            % Perform the following mathematical operation
20            % at each iteration:
21            y = y + sin(k*t)/k;
22
23            display(sprintf('When k = %.1f',k));
24            display('Then the plot is:');
25            updatePlot (t,y)
26        end
27
28     end
29
30     % Even though the approximations are constantly

                          fourier_demo          Ln 18    Col 11    OVR
```

c To see the effect, click Publish 📄.

8 Display equations with symbols and Greek characters (such as pi) using the TeX format. In this example, to create a comment containing a nicely formatted form of the equation, `y = y + sin(k*t)/k`, in the published document, use text markup as follows:

a Position the cursor at the end of the comment on line 20, `% at each iteration`.

b Select **Cell > Insert Text Markup > TeX Equation**.

MATLAB inserts the following lines; the second line is a sample equation with text markup applied:

```
%
% $$e^{\pi i} + 1 = 0$$
%
```

The sample equation, which is the text between the set of two dollar signs ($$), is highlighted.

c Replace the sample equation with the following TeX equation:

```
y = y + \frac{sin(k*t)}{k}
```

The three lines that display the TeX equation now appear as follows in the M-file:

```
%
% $$y = y + \frac{sin(k*t)}{k} $$
%
```

d To see the effect, click Publish ⬚.

9 Reduce the size of the published figures by editing the publish configuration for the file:

a Select **File > Publish Configuration for fourier_demo > Edit Publish Configurations for fourier_demo.m**.

The Edit M-File Configurations dialog box opens.

b In the column to the right of **Max image width (pixels)**, double-click **Inf**, and type the value 350.

c In the column to the right of **Max image height (pixels)**, double-click **Inf**, and type the value 350.

d Click **Save As**. The Save Publish Settings dialog box opens.

e In the **Settings name** field, type small_images, and then click **Save**.

f Click **Close**.

g To see the effect, click Publish ⬚.

10 To create a section header without including a cell break, follow these steps:

a Position the cursor at the beginning of line 33, where the comment % Even though the approximations are constantly appears.

b Select **Cell > Insert Text Markup > Section Title without Cell Break.**

c Replace SECTION TITLE with Note About Gibbs Phenomenon.

    d Delete line 34, where the comment `% DESCRIPTIVE TEXT` appears.

11 Select **File > Publish fourier_demo**.

The published document, an HTML file, appears in the MATLAB Web Browser, as shown in "Published Sample M-File After Formatting" on page 8-7.

By default, MATLAB stores the HTML document, `fourier_demo.html`, and the associated image files in an `/html` subdirectory within the directory containing the source M-file.

See "M-File Code After Text Markup" on page 8-16 for the resulting M-file code.

M-File Code After Text Markup

After adding text markup, the `fourier_demo.m` M-file appears as follows. When you publish the file to HTML, it appears as shown in "Published Sample M-File After Formatting" on page 8-7.

```
%% Square Waves from Sine Waves
% The Fourier series expansion for a square-wave is
% made up of a sum of odd harmonics, as shown here
% using MATLAB(R).

%% Add an Odd Harmonic and Plot It
function fourier_demo
    t = 0:.1:pi*4;
    y = sin(t);
    updatePlot(t,y);

    %%
    % In each iteration of the for loop add an odd
    % harmonic to y. As _k_ increases, the output
    % approximates a square wave with increasing accuracy.

    for k = 3:2:9
        %%
        % Perform the following mathematical operation
        % at each iteration:
```

```
          %
          % $$ y = y + \frac{sin(k*t)}{k} $$
          %
          y = y + sin(k*t)/k;

          display(sprintf('When k = %.1f',k));
          display('Then the plot is:');
          updatePlot (t,y)
      end

   end

   %% Note About Gibbs Phenomenon
   % Even though the approximations are constantly
   % improving, they will never be exact because of the
   % Gibbs phenomenon, or ringing.

   function updatePlot(t,x)
   % Subfunction to update the plot
       cla
       plot(t,x)
   end
```

To open the formatted file in the Editor, instead of following the steps in "Producing the Formatting for the Example" on page 8-11 run the following command:

```
edit(fullfile(matlabroot,'help','techdoc',...
'matlab_env','examples','fourier_demo2.m'))
```

To work with fourier_demo2.m on your system, save the file to a directory for which you have write permission.

Formatting M-File Comments for Publishing

Note Many examples in this section show the effects of publishing to HTML. For information on how to publish to HTML, see "Producing Published Output from M-Files" on page 8-68.

Overview of Formatting M-File Comments for Publishing

This section describes the types of text formatting you can specify for M-files, so that your published output appears like a polished document, rather than a text file of code. This enables you to single-source your M-file code with documentation that describes what the code is doing.

You can format M-file comments in either of the following ways to specify how the published results will appear:

- Use **Cell > Insert Text Markup** menu options to format the comments. This automatically inserts the text markup symbols for you.

- Type the markup symbols directly in the comments; the markup symbols you enter are the same as the text markup that results when you use the equivalent menu item. See "Summary of Markup for Formatting M-Files for Publishing" on page 8-61 for details.

You can use text markup as you create a new file, to mark up an existing M-file, or a combination of the two. When you use the **Cell** menu options, the Editor may insert more comment lines and other markup that you want. See "Cleaning Up Text Markup Before Publishing M-Files" on page 8-58 for information on how you can adjust the inserted text when you are done formatting a file.
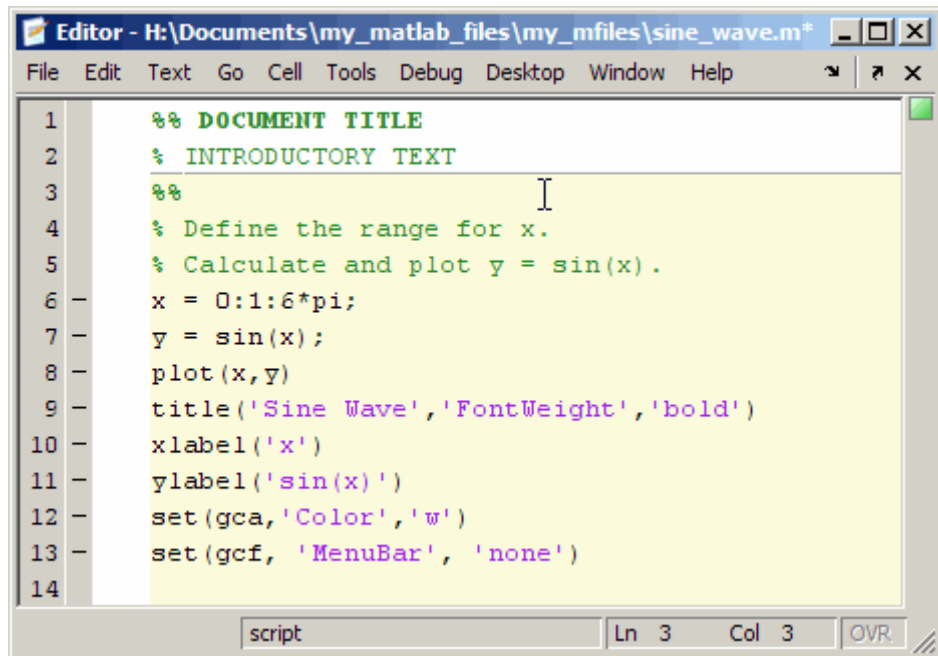
Several examples in the formatting sections that follow use this M-file, sine_wave.m:

```
% Define the range for x.
% Calculate and plot y = sin(x).
% Display plot in published document.
x = 0:1:6*pi;
y = sin(x);
plot(x,y)
title('Sine Wave', 'FontWeight','bold')
xlabel('x')
ylabel('sin(x)')
set(gca, 'Color', 'w')
set(gcf, 'MenuBar', 'none')
```

Creating Document Titles and Introductory Text for Publishing an Existing M-File

To specify a document title and introductory text for an M-file, follow these steps:

1 In the Editor, position the cursor anywhere in an M-file.

2 Select **Cell > Insert Text Markup > Document Title and Introduction**. The first three lines of the file appear as shown in the following image.
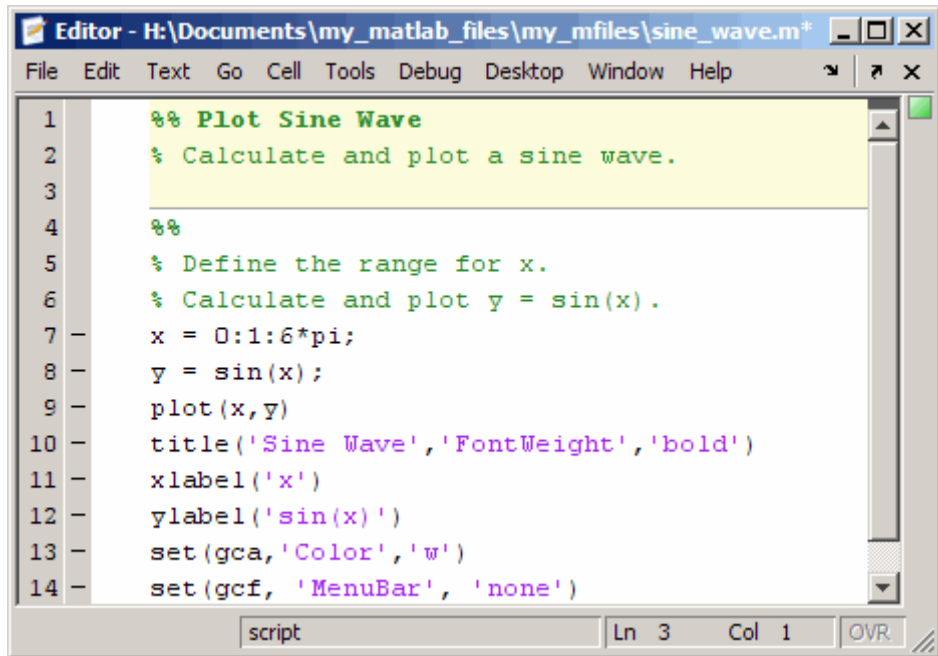

```
1    %% DOCUMENT TITLE
2    % INTRODUCTORY TEXT
3    %%
4    % Define the range for x.
5    % Calculate and plot y = sin(x).
6  - x = 0:1:6*pi;
7  - y = sin(x);
8  - plot(x,y)
9  - title('Sine Wave','FontWeight','bold')
10 - xlabel('x')
11 - ylabel('sin(x)')
12 - set(gca,'Color','w')
13 - set(gcf, 'MenuBar', 'none')
14
```

3 Replace `DOCUMENT TITLE` with the cell heading that you want to use; for example, `Plot Sine Wave`.

4 Replace INTRODUCTORY TEXT with text that introduces the M-file; for example, `Calculate and plot a sine wave`.

5 Insert a blank comment line to increase readability.

If you specify the example text suggested in the previous list, then the first four lines of the resulting M-file appear as follows.

```
%% Plot Sine Wave
% Calculate and plot a sine wave.

%%
% Define the range for x.
% Calculate and plot y = sin(x).
x = 0:1:6*pi;
y = sin(x);
plot(x,y)
title('Sine Wave','FontWeight','bold')
xlabel('x')
ylabel('sin(x)')
set(gca,'Color','w')
set(gcf, 'MenuBar', 'none')
```

Notice that a horizontal rule, which indicates a cell break, ends the title and introductory text. When you insert a document title and introduction, the Editor also adds a cell break in preparation for the first section within the M-file.

When you publish the M-file, the document title is formatted as a top-level heading (h1 in HTML), using a large size, bold font; the introductory text appears as formatted text. The following figure shows the M-file published to HTML and presented in the MATLAB Web Browser.
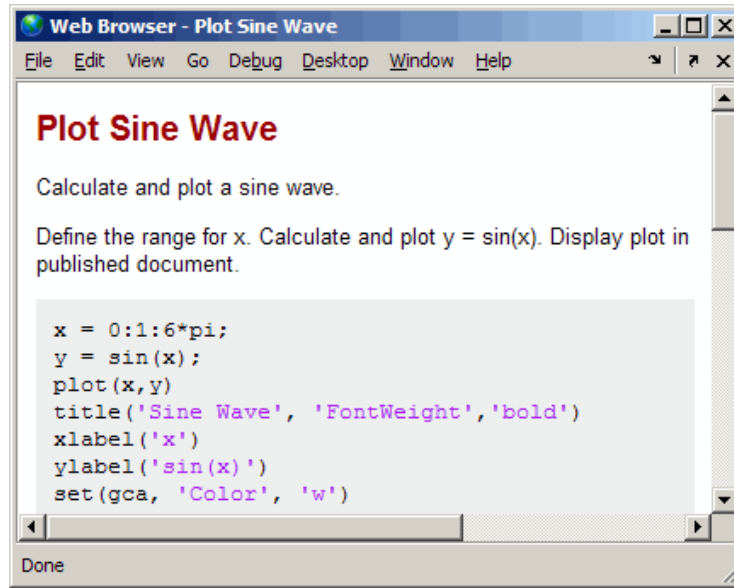
Specifying a Title for the New Section that the Editor Inserts with the Document Title

When you follow the steps in the previous section, "Creating Document Titles and Introductory Text for Publishing an Existing M-File" on page 8-20, the first cell, demarcated in the Editor with the horizontal rule followed by a line with double percent signs (%%), is not evident in the published file because the section does not have a title.

To provide a title for the section, insert text after the double percent signs—for example, `Calculate and Plot Sine Wave`. When you republish the file to HTML, it appears in the MATLAB Web Browser as shown in the following image. Notice that MATLAB automatically inserts the Contents heading and the link to the section when you publish the file to HTML.
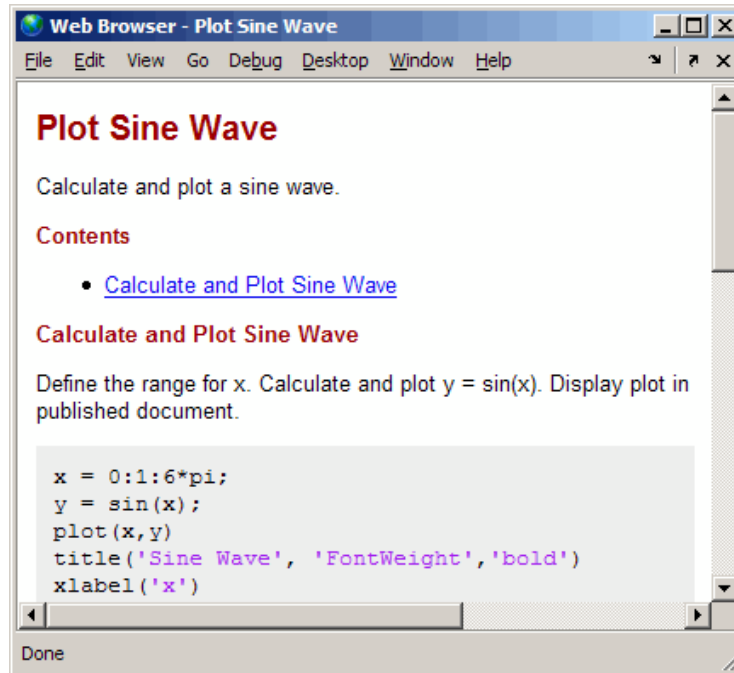
The file now has a document title, introductory text, and a first section. You can add more sections, as described in "Creating New Section Titles" on page 8-23.

Note You can add any comments in the lines immediately following the title. However, if you want the title to appear as the overall document title, you cannot add any other text before the next cell (a line starting with %%) .
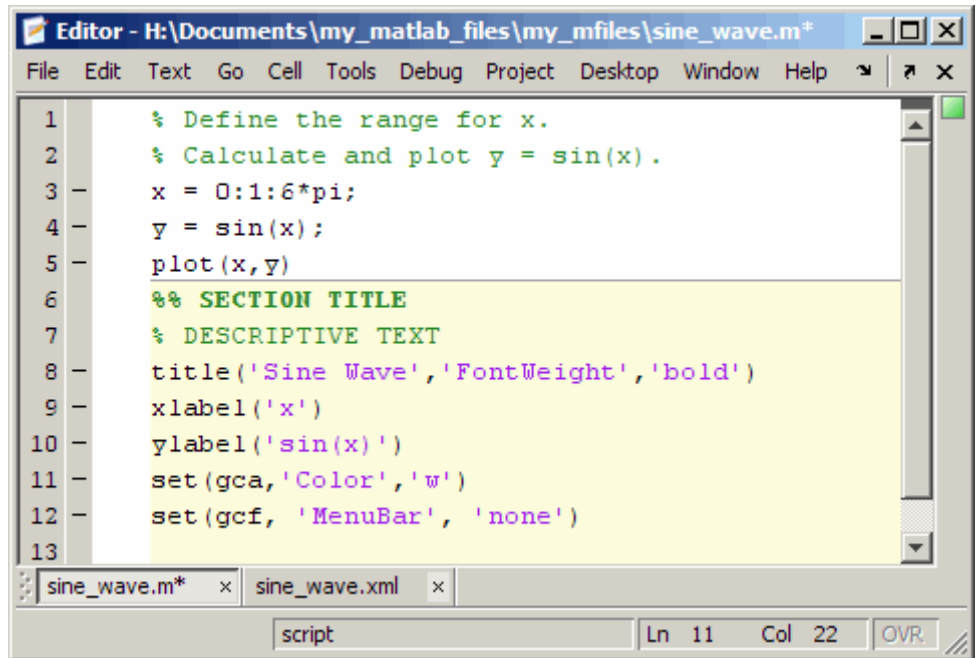
Creating New Section Titles

To insert a new section title and descriptive text within an M-file, follow these steps:

1 Position the cursor where you want to insert a new cell—before the `title` function shown in the previous example, for instance.
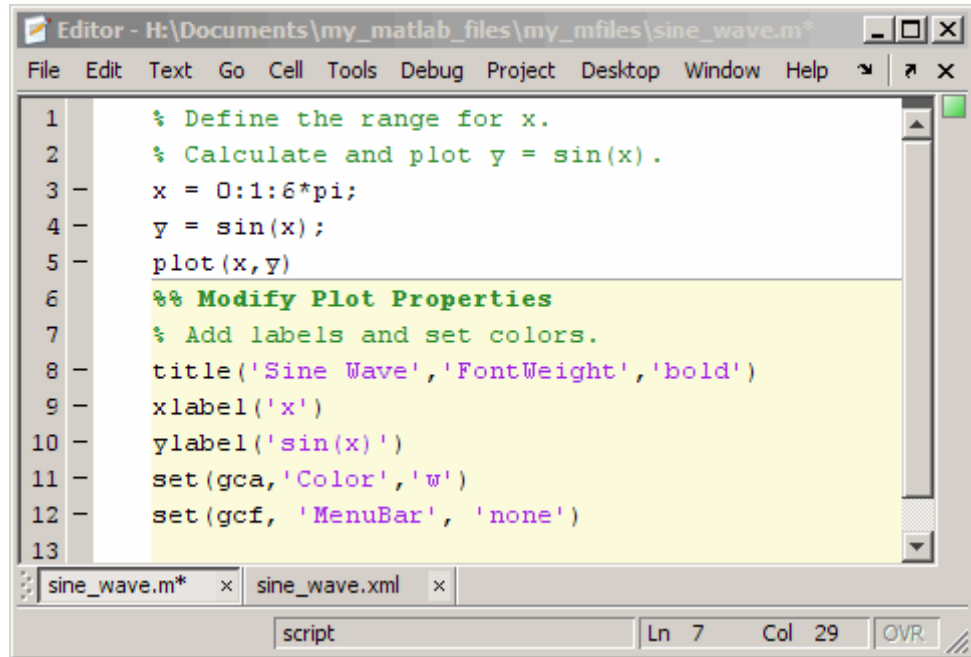
2 Select **Cell > Insert Text Markup > Section Title with Cell Break.**
The file appears as follows.

3 Replace `SECTION TITLE` with your title—`Modify Plot Properties`, for
example.

4 Replace `DESCRIPTIVE TEXT` with text that describes the cell—`Add labels
and set colors.`, for example.

If you specify the example text suggested in the previous list and "Specifying
a Title for the New Section that the Editor Inserts with the Document Title"
on page 8-22, then the resulting M-file appears as follows.

When you publish the M-file to HTML, the section title appears as a heading, using a medium size, bold font. Comments appear as formatted text in the published output. The following figure shows the results when you publish the updated `sine_wave.m` file to HTML output. Note that the `Modify Plot Properties` heading is formatted as an `h2`.
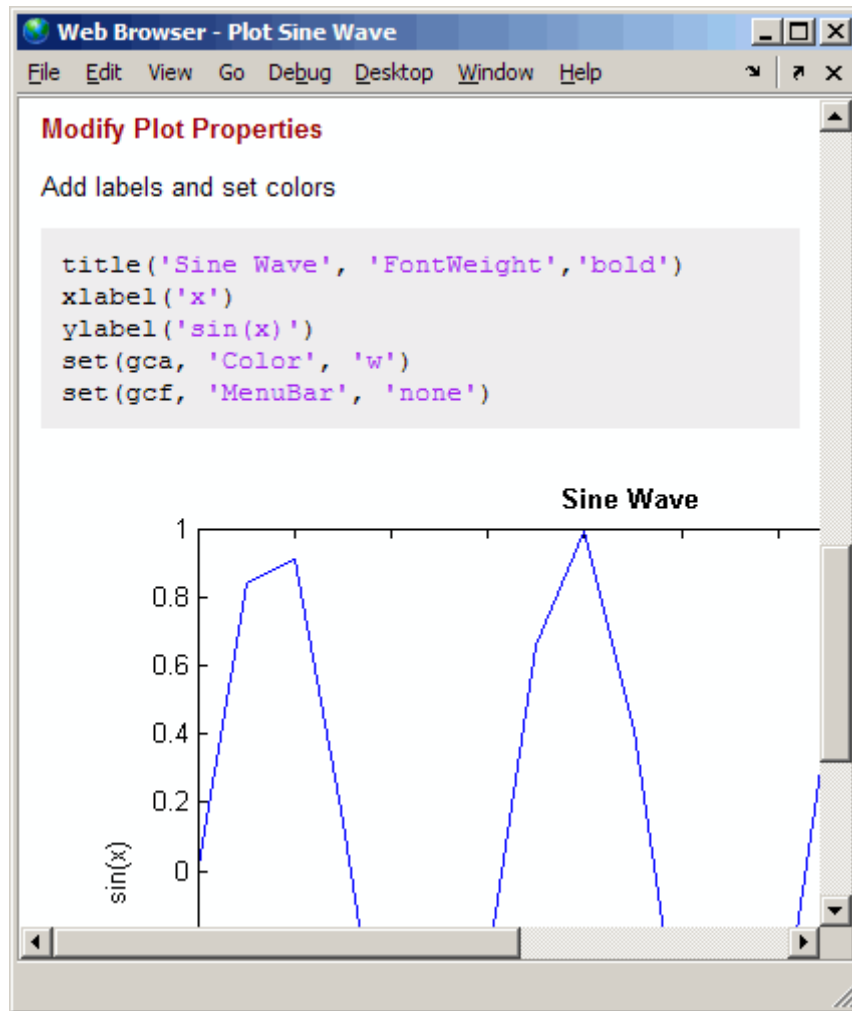
Specifying Preformatted Text in M-Files for Publishing

MATLAB software enables you to specify preformatted text in an M-file. Preformatted text appears in monospace font, maintains the white space that you specify and does not wrap long lines.

To insert preformatted text, follow these steps:

1 Position the cursor within the M-file where you want to insert preformatted text.

2 Select **Cell > Insert Text Markup > Preformatted Text**. Four lines of text are inserted, as shown.
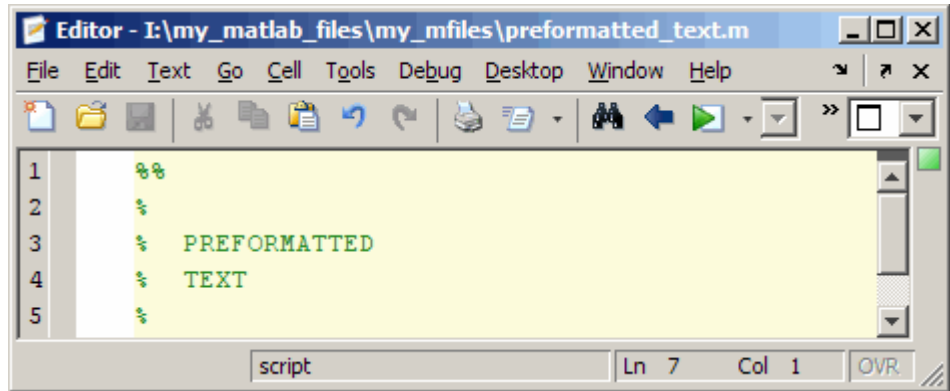
3 Being careful not to delete the two blank spaces before the word PREFORMATTED, replace the words PREFORMATTED and TEXT with your text, including tabs, spaces, and additional comment lines. For example:

  a Replace `PREFORMATTED` with `Text line 1`.

  b Insert a tab before `TEXT` on line 4.

  c On the last comment line type `Text line 3`.

The resulting comments appear as follows.
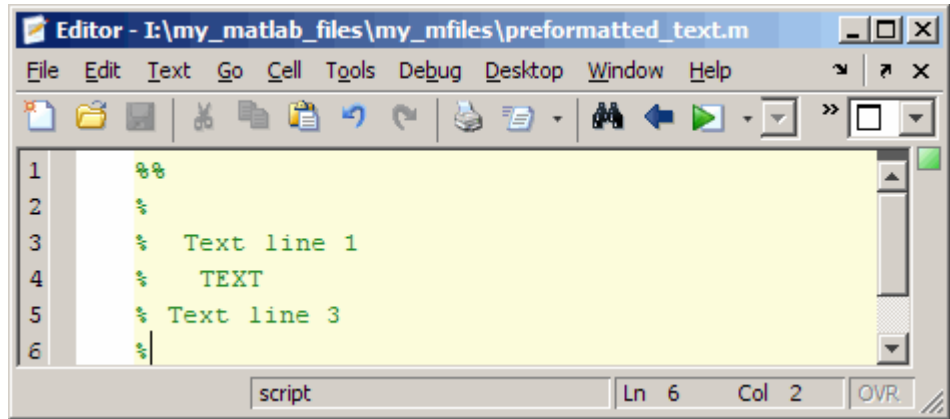
When you publish the M-file to HTML, the output appears as shown in the following figure.
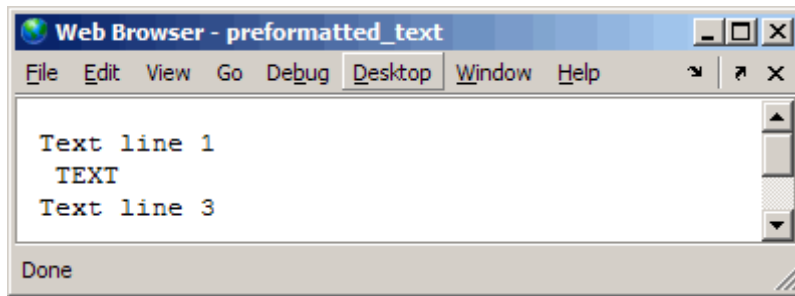
If the results are not as you expect, see "About Formatted Blocks" on page 8-54.

Specifying Bulleted or Numbered Lists in M-Files for Publishing

The following steps describe how to specify text markup for a bulleted or numbered list so as to create a published document that appears as shown in the following image when you click the Publish button .
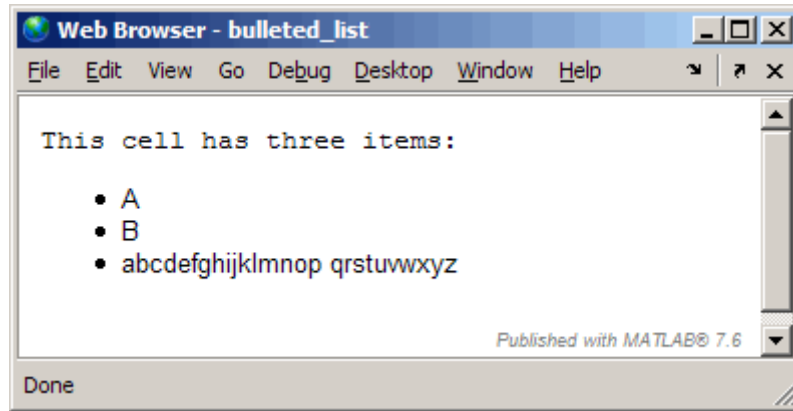
1 Position the cursor at the end of the line that precedes the location where you want to add a list. For example, if your M-file contains the following lines, position the cursor after the colon:

```
%%
% This cell has three items:
```

2 Select **Cell > Insert Text Markup > Bulleted List** or **Cell > Insert Text Markup > Numbered List**, depending on the type of list you want.

MATLAB adds four lines of formatted comments to the M-file. The following figure shows the result when you insert a bulleted list.
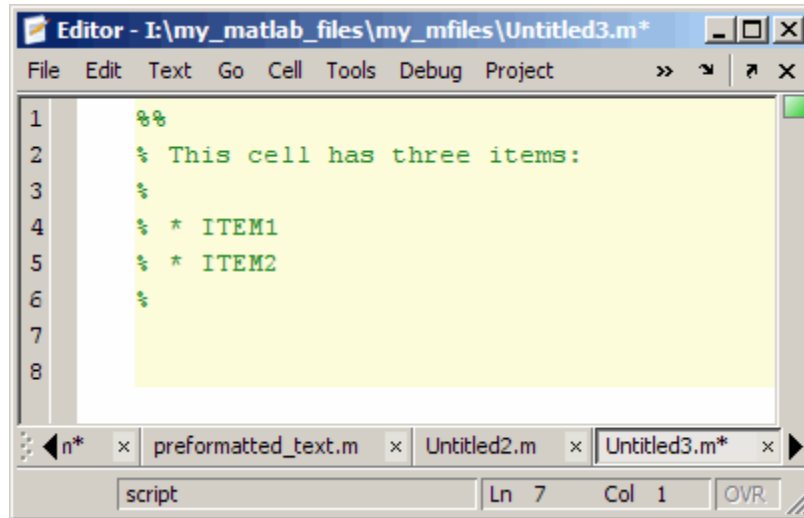
If you insert a numbered list, the text markup is the same, except a number sign (#) indicates a numbered list item.

3 Replace the sample text, ITEM1 and ITEM2, with your text. For example, replace ITEM1 with A and replace ITEM2 with B.

4 To create a multiline list item, break the line as desired, but do not insert the list item symbol (* or #) before the second line. For example, to insert the alphabet as a multiline list item, breaking the line at the letter p, type the alphabet as shown in the following figure.

Notice that the third list item is broken over two comment lines in the source, yet maintains the formatting of a list, as expected, when published (as shown at the beginning of this section).

If the results are not as you expect, see "About Formatted Blocks" on page 8-54.

Specifying Graphics in M-Files for Publishing

You can insert text markup such that the published document includes an image that was not generated by the M-file code, as shown in the following example. (By default, images generated by the M-file code are included in the published document.)

1 Position the cursor where you want to add a graphic. For example, if your M-file contains the following lines, position the cursor after the colon:

```
%% Image Example
% This is a graphic:
```

2 Select **Cell > Insert Text Markup > Image**. MATLAB adds text markup, as shown in the following figure.
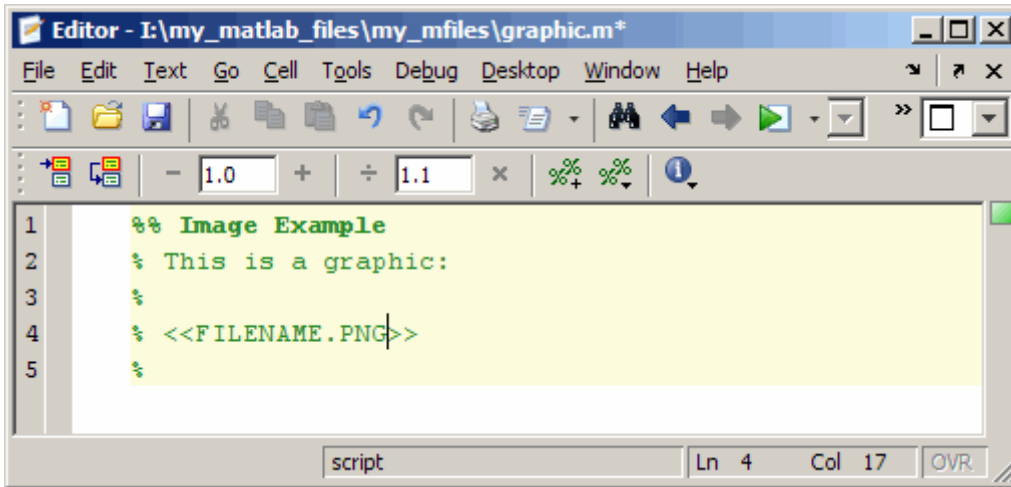
3 Replace FILENAME.PNG, with the file name of the graphic you want to insert, relative to the directory where MATLAB publishes the M-file.

For example, if you want to include the graphic, surfpeaks.jpg, and it is in the directory into which MATLAB publishes the M-file, then replace FILENAME.PNG with surfpeaks.jpg.

By default, MATLAB publishes the M-file to an /html subdirectory of the directory containing the M-file. You can change this directory, referred to as the output folder, by changing the publish configuration settings, as described in "Producing Published Output from M-Files" on page 8-68.

If the graphic is not in the directory to which the M-file is published, then you must specify the location of the graphic file as a relative path from the location of the published output file, as illustrated in the following table.

Directory Containing the M-File	Directory Where MATLAB Publishes the M-File	Image File Location	How to Specify the Image in the M-File Comment
I:/my_mfiles	I:/my_mfiles/html	I:/my_mfiles/html	% <<surfpeaks.jpg>>
I:/my_mfiles	I:/my_mfiles/doc	I:/my_mfiles/images	% <<../images/surfpeaks.jpg>>

When you publish the M-file to HTML, the output appears as shown in the following figure.
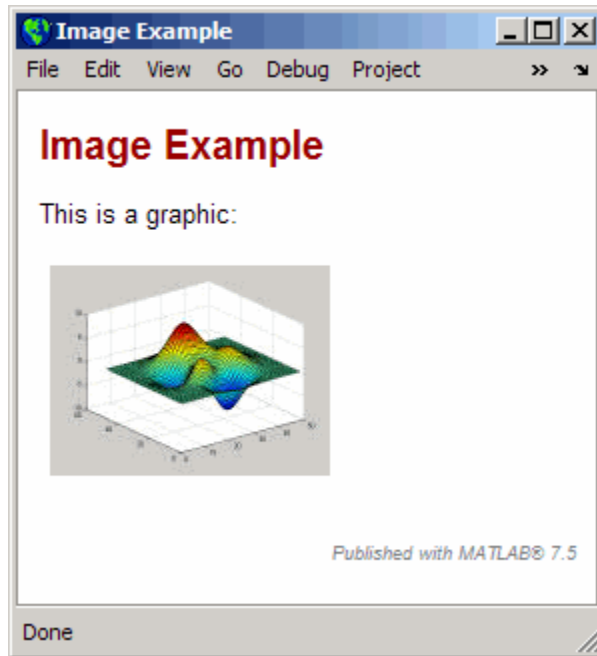
If the results are not as you expect, see "About Formatted Blocks" on page 8-54.

Creating the surfpeaks.jpg Image

To create the surfpeaks.jpg image used in the preceding example, follow these steps:

1 Create an html subdirectory in the directory where the M-file that references the graphic is located.

2 Enter the following in the Command Window:

```
>> surf(peaks)
```

A Figure window opens and displays the surfpeaks figure.

3 Save the figure as `surfpeaks.jpg` in the `html` subdirectory that you created in step 1.

Note Unless you reduce the size of `surfpeaks.jpg`, it will appear larger than that shown in the previous example.

Specifying HTML Markup Tags in M-Files for Publishing

You can use the **Cell** menu to insert HTML code into your M-file. When you do so, the Editor inserts HTML code for a one-column, two-row table. You can use the inserted code as a guideline for inserting other HTML code.

Note When you insert text markup for HTML code, the HTML code is published only when the specified output type is HTML. For example, if you add HTML markup, but then specify LaTeX as the output file format, MATLAB software does not publish the text enclosed within the HTML markup. See "Producing Published Output from M-Files" on page 8-68 for information on specifying the output file format.
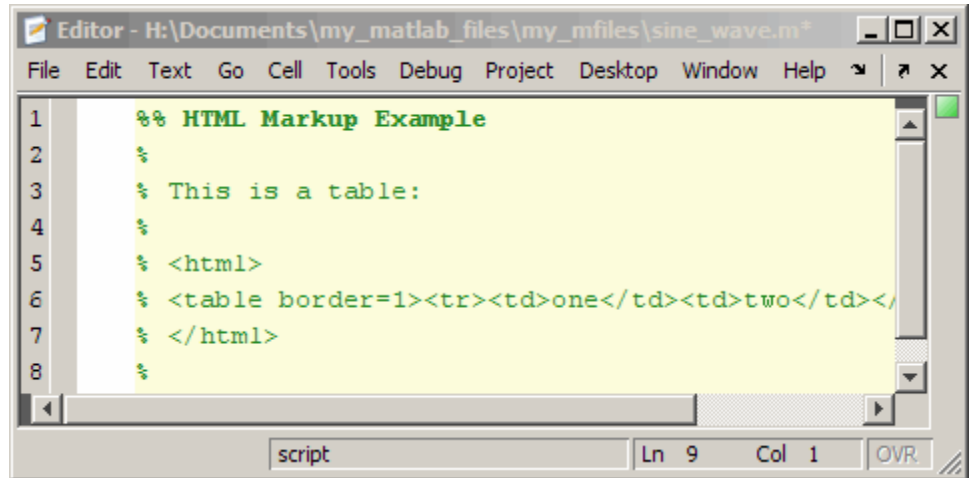
To insert the text markup for HTML code, follow these steps:

1 Position the cursor at the end of the comment that precedes the location where you want to insert HTML code. For example, if the M-file contains the following lines, position the cursor after the colon:

```
%% HTML Markup Example
% This is a table:
```

2 Select **Cell > Insert Text Markup > HTML Markup**. MATLAB adds HTML markup, as shown in the following figure.

3 Edit the inserted HTML code to specify the HTML code that you want to use.

If you publish the M-file to HTML and leave the inserted HTML code as is, MATLAB creates a single-row table with two columns, containing the values one and two as shown in the following figure.

If the results are not as you expect, see "About Formatted Blocks" on page 8-54.

Specifying LaTeX Markup for Publishing

You can use the **Cell** menu to insert LaTeX code. You can use the inserted code as a guideline for inserting other LaTeX code.

Note When you insert text markup for LaTeX code, that code is published only when the specified output type is LaTeX. For example, if you add LaTeX markup, but then specify HTML as the output file format, MATLAB software does not publish the code enclosed within the LaTeX markup. See "Producing Published Output from M-Files" on page 8-68 for information on specifying the output file format.
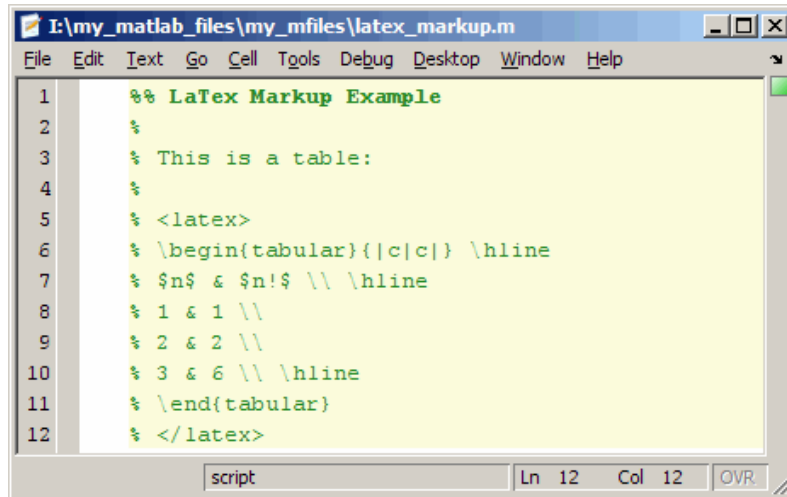
To insert the text markup for LaTeX code, follow these steps:

1 Position the cursor at the end of the comment that precedes the location where you want to insert LaTeX code. For example, if the M-file contains the following lines, position the cursor after the colon:

```
%% LaTeX Markup Example
% This is a table:
```

2 Select **Cell > Insert Text Markup > LaTeX Markup**. MATLAB adds LaTeX markup, as shown in the following figure.
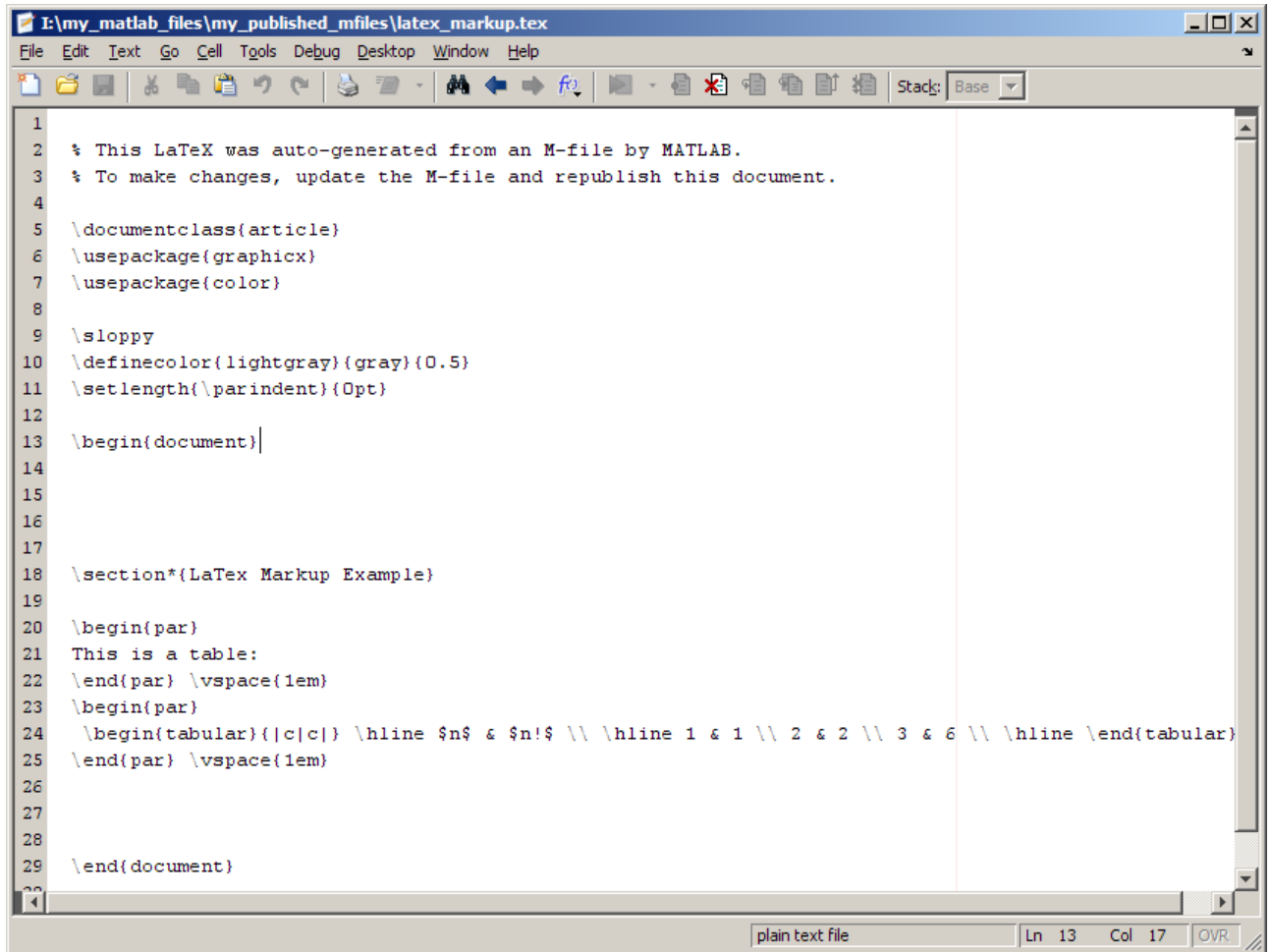
3 Edit the inserted LaTeX code to specify the LaTeX code that you want to use.

If you publish the M-file to LaTeX, and leave the inserted markup text as is, the Editor opens a new file with the LaTeX code, as shown in the following figure. (See "Creating a Publish Configuration for an M-File" on page 8-70 for information on specifying LaTeX as the output format for a published document.)

```
I:\my_matlab_files\my_published_mfiles\latex_markup.tex

File  Edit  Text  Go  Cell  Tools  Debug  Desktop  Window  Help

1
2    % This LaTeX was auto-generated from an M-file by MATLAB.
3    % To make changes, update the M-file and republish this document.
4
5    \documentclass{article}
6    \usepackage{graphicx}
7    \usepackage{color}
8
9    \sloppy
10   \definecolor{lightgray}{gray}{0.5}
11   \setlength{\parindent}{0pt}
12
13   \begin{document}
14
15
16
17
18   \section*{LaTex Markup Example}
19
20   \begin{par}
21   This is a table:
22   \end{par} \vspace{1em}
23   \begin{par}
24    \begin{tabular}{|c|c|} \hline $n$ & $n!$ \\ \hline 1 & 1 \\ 2 & 2 \\ 3 & 6 \\ \hline \end{tabular}
25   \end{par} \vspace{1em}
26
27
28
29   \end{document}

                                              plain text file        Ln  13    Col  17    OVR
```

If you compile the published LaTeX code, it appears as follows.

LaTex Markup Example

This is a table:

n	$n!$
1	1
2	2
3	6
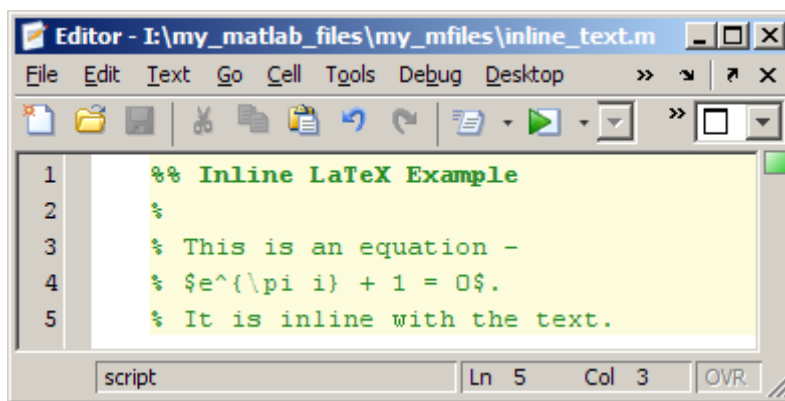
If the results are not as you expect, see "About Formatted Blocks" on page 8-54.

Specifying Inline LaTeX Math Symbols in M-Files for Publishing

You can make LaTeX math symbols appear inline when you publish an M-file to any format, except Microsoft PowerPoint®. For Microsoft PowerPoint output, consider using an "Specifying LaTeX Markup for Publishing" on page 8-36 instead.

To make a LaTeX math symbol appear inline, enclose the string within dollar sign symbols ($) within a formatted comment block.

For example, suppose your M-file appears as follows:

When you publish the file to HTML, it appears as in the image that follows. Notice that the LaTeX math symbols are inline with the rest of the text:
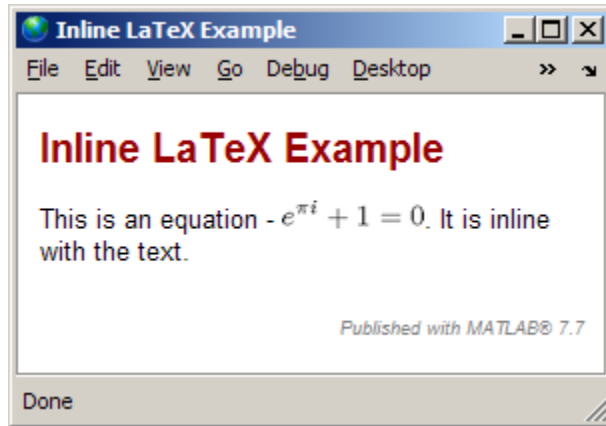
For a list of symbols you can display, and the character sequence to create them, see the MATLAB Text `String` property.
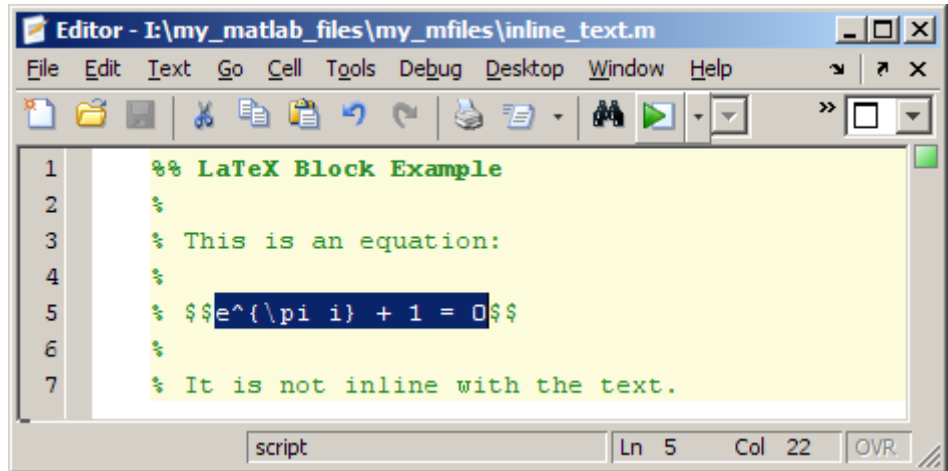
Specifying LaTeX Math Symbols as Blocks in M-Files for Publishing

You can use the **Cell** menu to insert LaTeX symbols in blocks offset from the main comment text. You can use the inserted code as a guideline for inserting other LaTeX code.

1 Position the cursor before the line where you want to add an equation or symbols. For example, if your M-file contains the following lines, position the cursor after the colon on the end of the third line:

```
%% LaTeX Block Example
%
% This is an equation:
% It is not inline with the text.
```

2 Select **Cell > Insert Text Markup > TeX Equation** to insert the sample equation markup.

3 Replace the inserted sample markup `e^{\pi i} + 1 = 0` with the LaTeX math symbols that you want.

For a list of symbols you can display, and the character sequence to create them, see the MATLAB Text `String` property.

If you publish the file to HTML, and leave the inserted sample text markup as is, the published document appears as shown in the following figure.
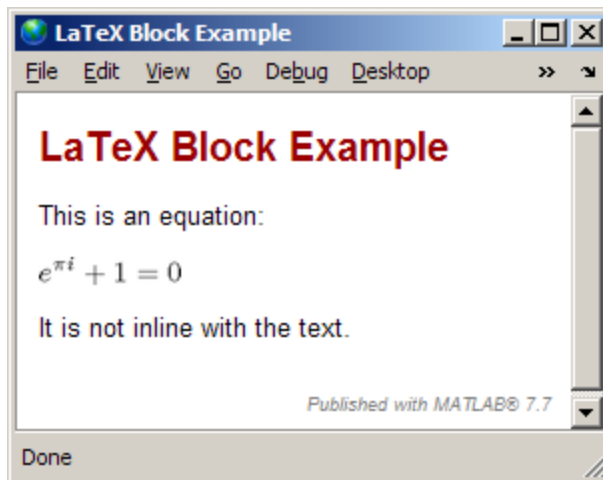
If the results are not as you expect, see "About Formatted Blocks" on page 8-54.

Forcing a Snapshot of Output in M-Files for Publishing

You can use the **Cell** menu to insert code that forces a snapshot of output, such as a figure. This is useful, for example, if you have a `for` loop that generates numerous figures and you want to include them all in the published output, after the `for` loop `end` statement.

1 Position the cursor at the end of the line where you want to force a snapshot of the output. For example, if your M-file contains the following lines, position the cursor after the line containing the `imagesc` function:

```
%% Scale magic Data and Display as Image:

for i=1:3
    imagesc(magic(i))
end
```

2 Select **Cell > Insert Text Markup > Force Snapshot** to insert the `snapnow` function:

```
%% Scale magic Data and Display as Image:

for i=1:3
    imagesc(magic(i))
    snapnow;
end
```

3 If you publish the file to HTML, the published document resembles the following figure. The images in your published document will be larger than shown in the figure. To resize of images generated by M-file code, you use the **Max image width** and **Max image height Publish settings**, as described in "Producing Published Output from M-Files" on page 8-68.
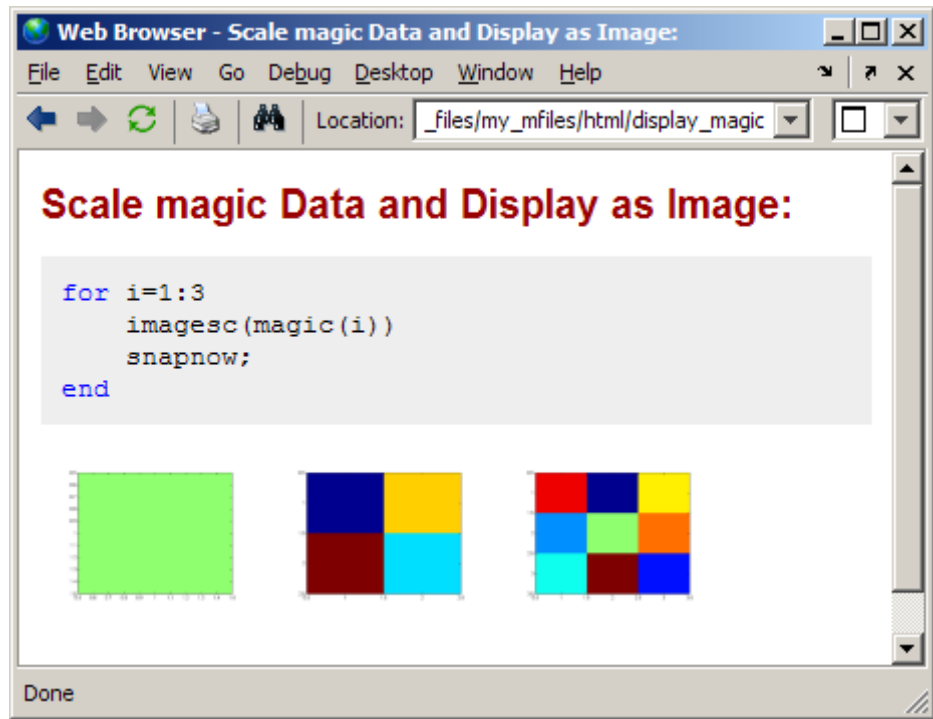
Specifying Bold, Italic, and Monospaced Text Formats in M-Files for Publishing

You can mark up selected strings in the M-file comments so that they appear in bold, italic, or monospaced text formats when you publish the M-file, as described in the following sections.

Marking Up Existing Comments with Font Formats

To mark up existing comments, follow these steps:

1 Within a comment, select text that you want to be bold, italic, or monospaced.

2 Select **Cell > Insert Text Markup**, and then select **Bold Text**, **Italic Text**, or **Monospaced Text**.

Inserting New Comments with Font Formats

To insert sample text that you will replace with your new comment text, follow these steps:

1 Select **Cell > Insert Text Markup**, and then select **Bold Text**, **Italic Text**, or **Monospaced Text**.

2 Replace the inserted text with the text that you want formatted.

When the Editor inserts sample text, the inserted text appears as follows:

```
% *BOLD TEXT*
% _ITALIC TEXT_
% |MONOSPACED TEXT|
```

Example of Font Formats

Suppose your M-file appears as follows.
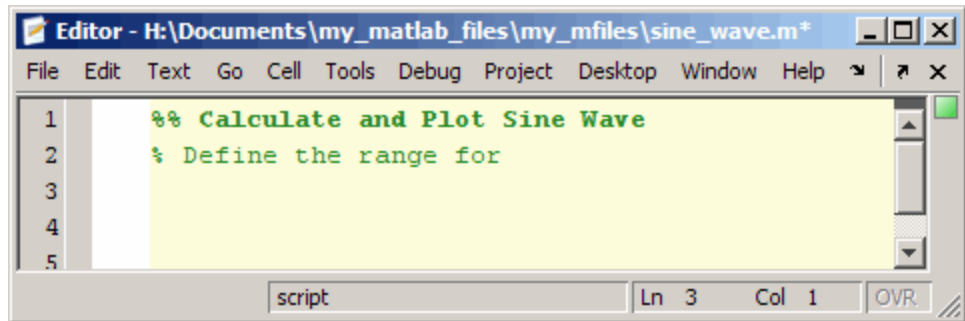
Format the comments as follows:

1 Select the word `Define`, and then select **Cell > Insert Text Markup > Bold Text**.

2 Select the word `range`, and then select **Cell > Insert Text Markup > Italic Text**.

3 Position the cursor after the word `for`, insert a space, and then select **Cell > Insert Text Markup > Monospaced Text**.

The M-file appears as follows.

4 Replace |MONOSPACED TEXT| with |x|.

If you publish the M-file to HTML, the output appears as shown in the following figure.
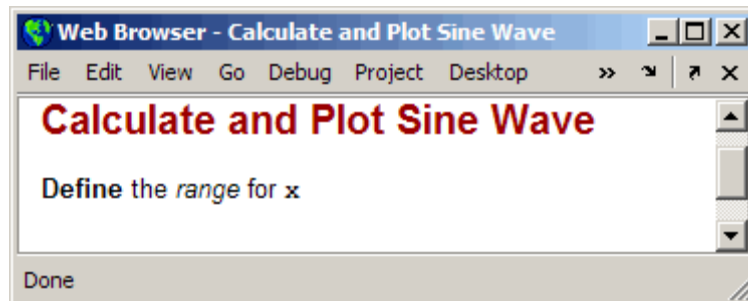
Specifying Trademarks in M-Files for Publishing

If the comments in your M-file include trademarked terms, you can include text to produce a trademark symbol (™) or registered trademark symbol (®) in the published output.

- To produce the trademark symbol, enter (TM) in an M-file comment.

- To produce the registered trademark symbol, enter (R) in an M-file comment.

8-45

For example, suppose you enter lines in an M-file as shown in the following image.
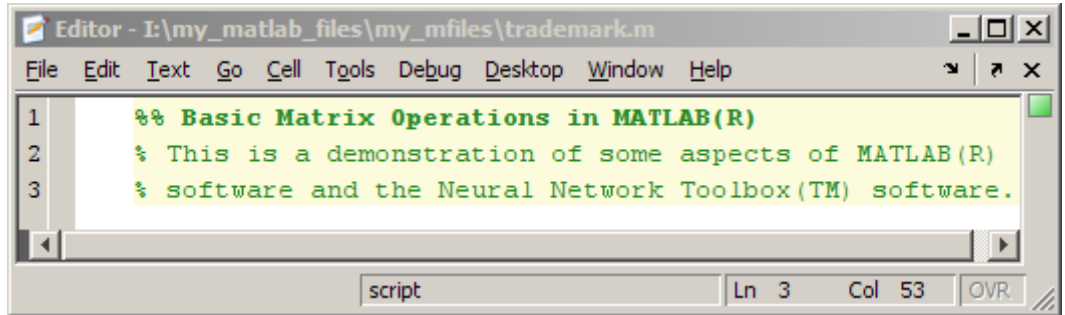
If you publish the M-file to HTML, it appears as follows in the MATLAB Web Browser.

Specifying Links in M-Files for Publishing

You can insert dynamic or static links within an M-file comment, and then publish the M-file to HTML, or XML. You can also publish static hyperlinked text to Microsoft Word.

When you specify a *dynamic* link, MATLAB resolves the URL when the user clicks the link in the published document. You do not specify the exact URL in the code. This is useful when, for example, you want to point the reader to

some MATLAB code or documentation. When you specify a *static* link, you specify the complete URL within the M-file code. This is useful when you want to point the user to a location on the Web. For both static and dynamic links, the publish document contains active links.

You can include or exclude the URL from a published static link. You might, for example, include the URL when you anticipate that readers of your published document might view it in printed format, and therefore need the URL. You might exclude the URL, when you are confident that readers will view your published document online and therefore be able to click the link.

This section includes the following topics:

- "Inserting Static Links and Publishing URLs" on page 8-47
- "Inserting Static Links Without Publishing URLs" on page 8-48
- "Inserting Dynamic Links" on page 8-49
- "Effect of Using Hyperlinked Text from the MATLAB Command Window" on page 8-53

Inserting Static Links and Publishing URLs

You can insert a URL that you know at the time the M-file is written, such as `www.mathworks.com`, by following these steps:

1 Within a comment, position the cursor where you want to insert the hyperlinked text. For example, suppose you want to specify a link to more information about a topic. You might have the following comment within the M-file:

```
%%
% For more information, see our Web site:
```

Position your cursor after the colon (:).

2 Select **Cell > Insert Text Markup > Hyperlinked Text**. The Editor inserts the following:

```
<http://www.mathworks.com The MathWorks>
```

3 Replace `www.mathworks.com` with the URL you want to use.

4 Delete the string, `The MathWorks`.

When you publish the M-file to HTML, the results resemble the following figure (except the URL in this image is still `http://www.mathworks.com`).
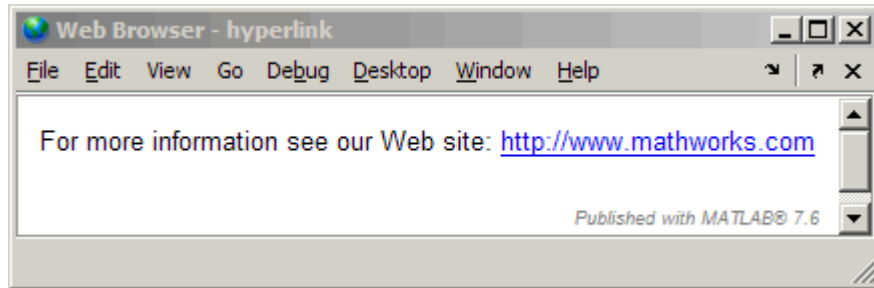
Inserting Static Links Without Publishing URLs

To insert hyperlinked text without a printed URL, follow these steps:

1 Within a comment, position the cursor where you want to insert the hyperlinked text. For example, suppose you want to specify a link to the MathWorks Web site. You might have the following lines within your M-file:

```
%%
% For more information, see the MathWorks Web site.
```

Select the text you want to replace with a link. For example, select "MathWorks," as shown in the following figure.
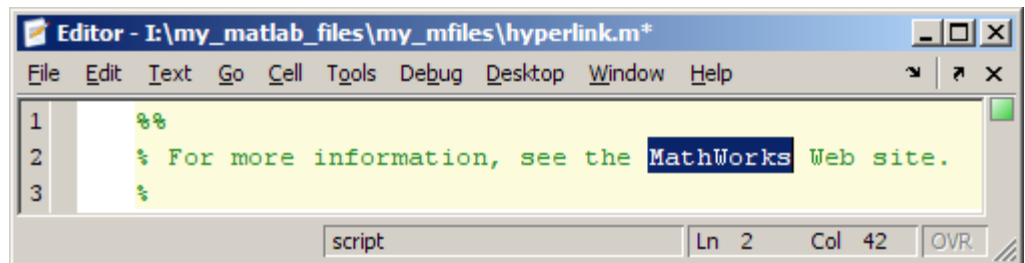
2 Select **Cell > Insert Text Markup
 > Hyperlinked Text**. The Editor replaces the selected text with the
following:

```
<http://www.mathworks.com MathWorks>
```

If you publish the M-file to HTML, the results are as shown in the following
figure.
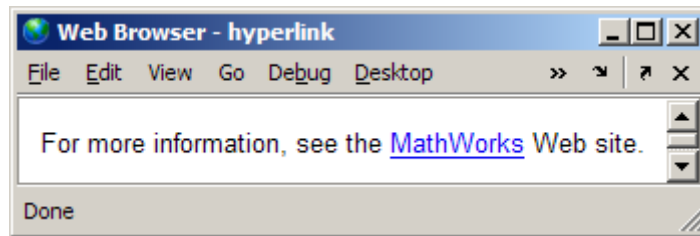
3 Replace www.mathworks.com with the URL that you want to use.

4 Replace MathWorks with the text that you want to appear as the
hyperlinked text.

Inserting Dynamic Links

You can insert a link that is evaluated by MATLAB at the time the user
clicks the link in the published document. You implement these as matlab:
links. Unlike other uses of matlab:, however, you are not required to use it
within another function, such as disp.

You cannot insert dynamic links in Microsoft Word files.

Note A reader of your published document must have MATLAB running for
dynamic links to work.

For more information, see matlabcolon.

The following sections provide two examples:

- "Inserting a Dynamic Link to a File" on page 8-50
- "Inserting a Dynamic Link to a MATLAB Function Reference Help Topic" on page 8-52

Inserting a Dynamic Link to a File. You can specify a link to a file that you know will be in your readers' `matlabroot`, even though you do not know exactly where each reader has installed MATLAB, as follows:

1 Within a comment, position the cursor where you want to insert the link. For example, suppose you want to specify a link to the MATLAB help topic for the `publish` function. You might have the following comment within the M-file:

```
%%
% See the code
% for the publish function.
```

2 Replace the word `code` with the following markup:

```
<matlab:web(fullfile(matlabroot,'toolbox','matlab','codetools','publish.m')) code>
```

The resulting M-file code now appears as follows:

```
%%
% See the <matlab:web(fullfile(matlabroot,'toolbox''matlab','codetools','publish.m')) code>
% for the publish function.
```

When you publish the file to HTML, the results resemble the following figure.
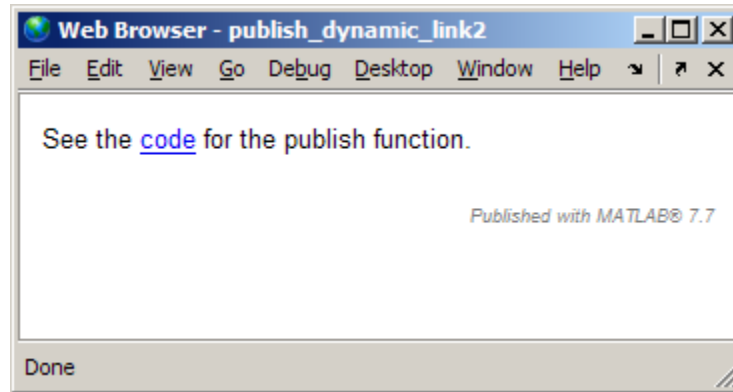
When the reader of the published document clicks the `code` link, the MATLAB Web browser opens and displays the code for the publish function. On the reader's system, MATLAB issues the command (although the command does not appear in their Command window). Notice in the Web browser title bar that the `matlabroot` specification in the M-File is resolved to the reader's installation directory for MATLAB.
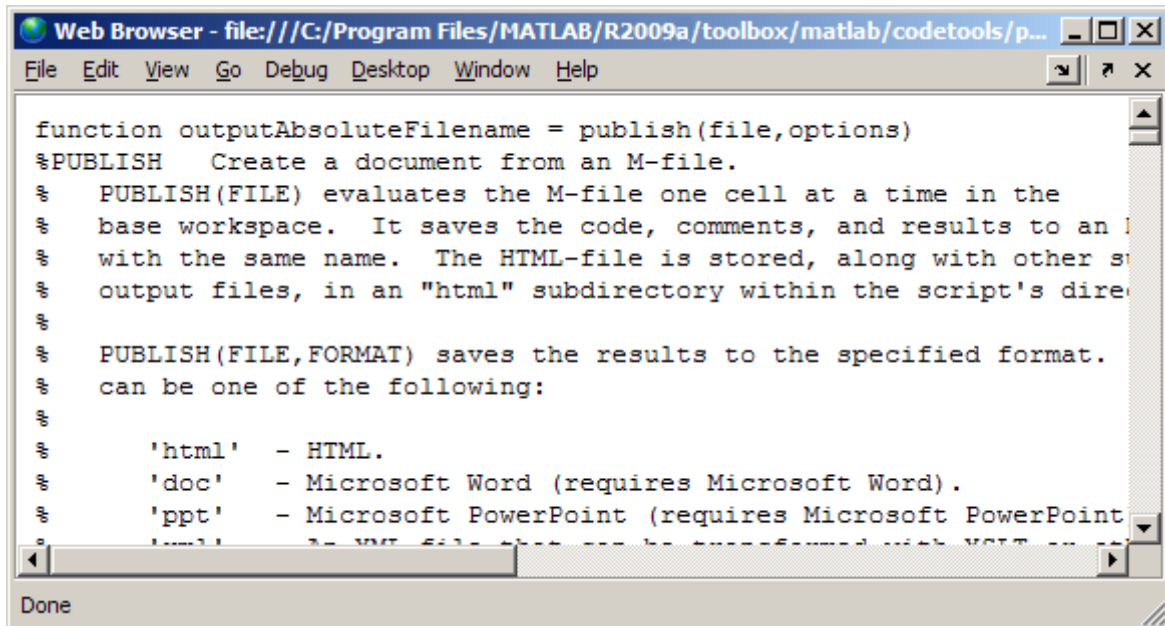
Inserting a Dynamic Link to a MATLAB Function Reference Help Topic.
You can specify a link to a MATLAB function reference help topic. For
example, suppose you know the readers of your published document will
have MATLAB installed and running, and you want to provide a link to the
publish reference page. Follow these steps:

1 Within a comment, position the cursor where you want to insert the
hyperlinked text. For example, suppose you want to specify a link to the
MATLAB help topic for the publish function. You might have the following
comment within the M-file:

```
%%
% See the help for the publish function.
```

2 Replace the word publish with the following markup:

```
<matlab:doc('publish') publish>
```

The resulting M-file code now appears as follows:

```
%%
% See the help for the <matlab:doc('publish') publish> function.
```

When you publish the file to HTML, the results resemble the following figure.
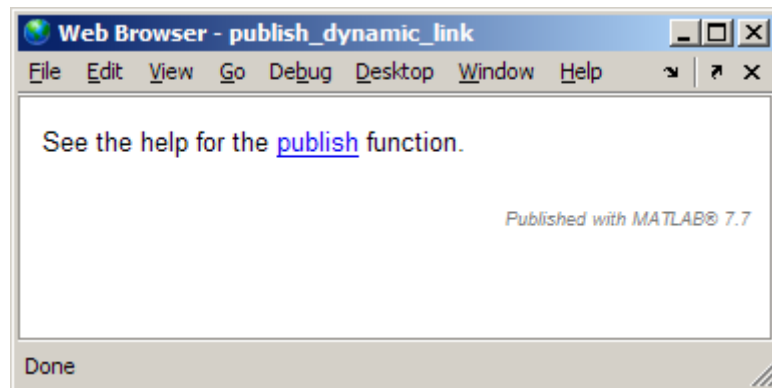
When the reader of the published document clicks the publish link, the
MATLAB help browser opens and displays the reference page for the publish

function. On the reader's system, MATLAB issues the command (although the command does not appear in their Command window).
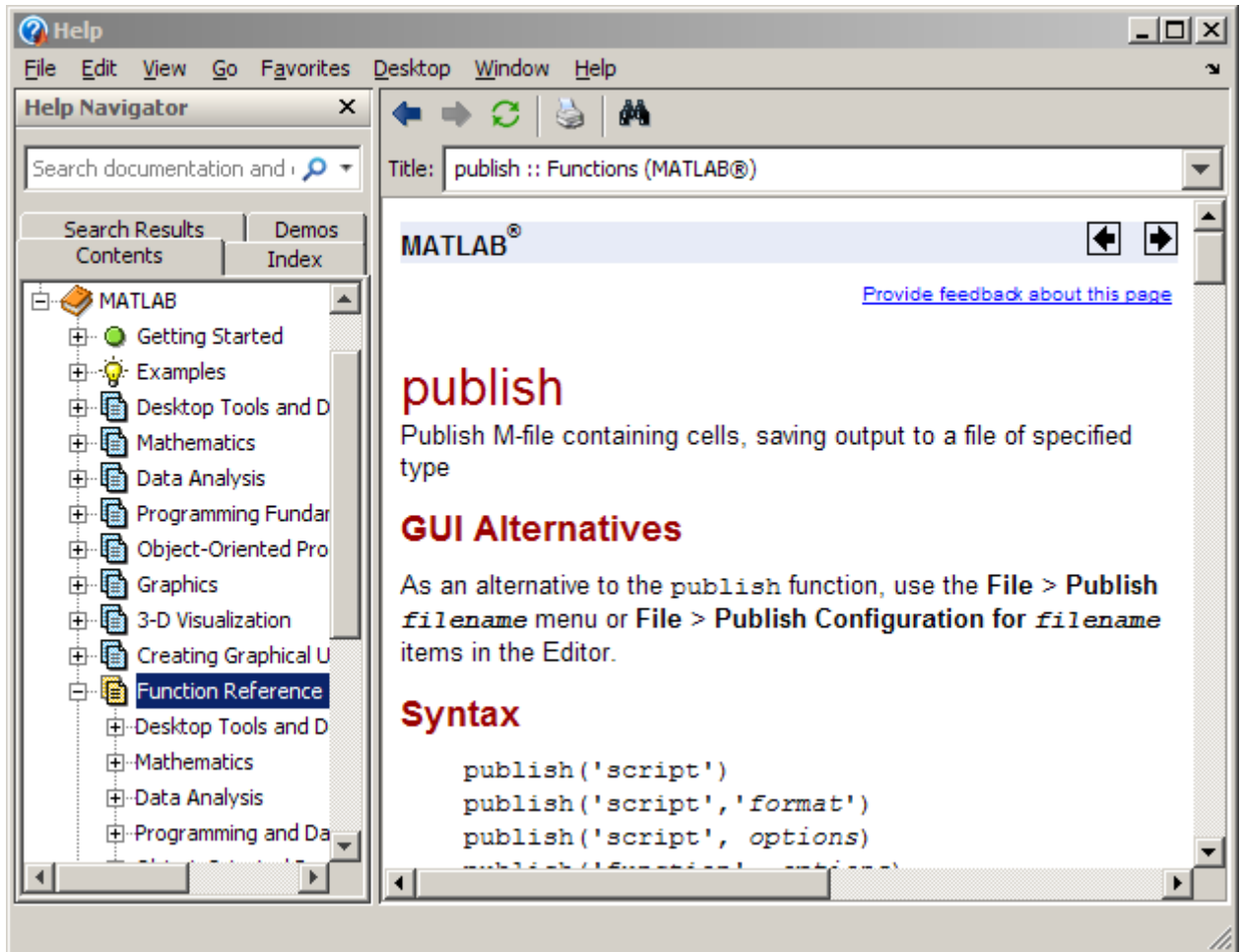
Effect of Using Hyperlinked Text from the MATLAB Command Window

You cannot use statements that display hyperlinked text in the MATLAB Command Window to create hyperlinked text in a published M-file. If you try, the published document shows the code rather than the hyperlink.

For example, suppose you enter the following code in the Command Window:

```
disp('<a href="http://www.mathworks.com">Link to MathWorks</a>')
```

When you press **Return**, the Command Window displays a link to the MathWorks Web site:

Link to MathWorks

However, if you include the preceding `disp` statement in an M-file that you publish, the HTML tag and the included text appear in the published document, rather than a link:

```
disp('<a href="http://www.mathworks.com">Link to MathWorks</a>')
```

Instead, use one of the methods described in these sections:

- "Inserting Static Links and Publishing URLs" on page 8-47
- "Inserting Static Links Without Publishing URLs" on page 8-48
- "Inserting Dynamic Links" on page 8-49

About Formatted Blocks

Multiple contiguous lines of comments immediately following a cell break are referred to as "descriptive" or "introductory text". Within these lines of comments, empty lines create formatted blocks. Formatted blocks control how comments appear within the final published document.

Specify each of the following items as a formatted block to achieve the intended results in the published document:

- Preformatted text
- Bulleted and numbered lists
- Graphics
- HTML markup
- LaTeX markup
- TeX equations

The following sections provide general information on formatted blocks:

- "Understanding How Formatted Blocks are Demarcated" on page 8-55
- "Understanding How Formatted Blocks Work" on page 8-55

Understanding How Formatted Blocks are Demarcated

A formatted block starts on the first comment line *after* one of the following:

- A cell break
- A blank comment line (a percent sign with no other characters on the line)

A formatted block ends on the last comment line *before* one of the following:

- A cell break
- A blank comment line
- Any line of uncommented M-file code

Understanding How Formatted Blocks Work

A comment is part of a formatted block if, on the first line of the block there are two spaces between the single percent sign (%) and the next character. Any number of white spaces can precede the percent sign. That is, the percent sign can be indented. The following two images demonstrate the difference between blocks that are and are not formatted:

- The first image show the M-file source code.
- The second image shows the M-file published to an HTML document.

Two spaces on the first line of the block ...

... therefore, this list will be formatted.

One space on the first line of the block ...

... therefore, this list will not be formatted.

One space on the first line of this block ...

... therefore, this tab will not be published.

Two spaces on the first line of an indented block ...

... therefore, this tab will be published.

```
Editor - I:\my_matlab_files\my_mfiles\formatted_comments.m
File  Edit  Text  Go  Cell  Tools  Debug  Desktop  Window  Help

 1    %% Formatted Block Demonstration
 2    % This M-file shows the difference between blocks
 3    % that are and are not formatted.
 4    %
 5    %% Block That Is Formatted
 6    %  This first block begins on line 6 of the M-file code.
 7    % Because there are two spaces between the percent sign
 8    % and the next character on line 6, this block is
 9    % formatted.
10    % Here is a list:
11    %  * Item one
12    %  * Item two
13    %
14    %% Block That Is Not Formatted
15    % This second block begins on line 15 of the M-file code.
16    % Because there is only one space between the percent sign
17    % and the next character on line 15, this block
18    % is NOT formatted.
19    % Here is a list:
20    %  * Item one
21    %  * Item two
22    %
23    %% Text That Contains Formatted and Unformatted Blocks
24    %  This third block begins on line 24 of the M-file code.
25    % Because there are two spaces between the percent sign
26    % and the next character on line 24, this block is
27    % formatted. Note that the 25th and subsequent lines
28    % in this block do not have two spaces between the single
29    % percent sign and the next character on the line. The first
30    % line within the block determines whether or not the
31    % entire block is formatted.
32    %
33    % This fourth block begins on line 33 of the M-file code.
34    % It is not a continuation of the third block because a
35    % blank comment line precedes line 33.
36    %        Even though a tab exists on this line in the
37    % M-file code, its spacing is not preserved in the
38    % published document; the tab occurs in the middle
39    % of a block that is not formatted.
40    %
41    %% Indented Text
42    %  This fifth block begins on line 42 of the M-file
43    % code. It is formatted because there are two
44    % spaces between the percent sign and the first
45    % character on line 42. Indenting a line has
46    % no effect on preformatting and is not presented
47    % in the published document as indented.

script                                    Ln 47    Col 45    OVR
```

Web Browser - Formatted Block Demonstration

File   Edit   View   Go   Debug   Desktop   Window   Help

Formatted Block Demonstration

This M-file shows the difference between blocks that are and are not formatted.

Contents

- Block That Is Formatted
- Block That Is Not Formatted
- Text That Contains Formatted and Unformatted Blocks
- Indented Text

Block That Is Formatted

```
This first block begins on line 6 of the M-file code.
Because there are two spaces between the percent sign
and the next character on line 6, this block is
formatted.
Here is a list:
* Item one
* Item two
```

Block That Is Not Formatted

This second block begins on line 15 of the M-file code. Because there is only one space between the percent sign and the next character on line 15, this block is NOT formatted. Here is a list: * Item one * Item two

Text That Contains Formatted and Unformatted Blocks

```
This third block begins on line 24 of the M-file code.
Because there are two spaces between the percent sign
and the next character on line 24, this block is
formatted. Note that the 25th and subsequent lines
in this block do not have two spaces between the single
percent sign and the next character on the line. The first
line within the block determines whether or not the
entire block is formatted.
```

This fourth block begins on line 33 of the M-file code. It is not a continuation of the third block because a blank comment line precedes line 33. Even though a tab exists on this line in the M-file code, its spacing is not preserved in the published document; the tab occurs in the middle of a block that is not formatted.

Indented Text

```
This fifth block begins on line 42 of the M-file
code. It is formatted because there are two
spaces between the percent sign and the first
character on line 42. Indenting a line has
no effect on preformatting and is not presented
in the published document as indented.
```

If you want, you can experiment with the code presented in the previous images. Open `formatted_block_demo.m` by running the following command:

```
edit(fullfile(matlabroot,'help','techdoc',...
'matlab_env','examples','formatted_block_demo.m'))
```

To work with `formatted_block_demo.m` on your system, save the file to a directory for which you have write permission. In the example, the file is saved to `I:\my_matlab_files\my_mfiles\formatted_comments.m`.

Cleaning Up Text Markup Before Publishing M-Files

When you insert text markup into an existing M-file using the **Cell** menu options, you might find that more comment lines than you need are inserted. You can adjust the inserted comments as needed for your purposes. If you delete blank comment lines that the **Cell** menu options insert there may be unintended consequences, however. See "Specifying Preformatted Text in M-Files for Publishing" on page 8-26 for details.
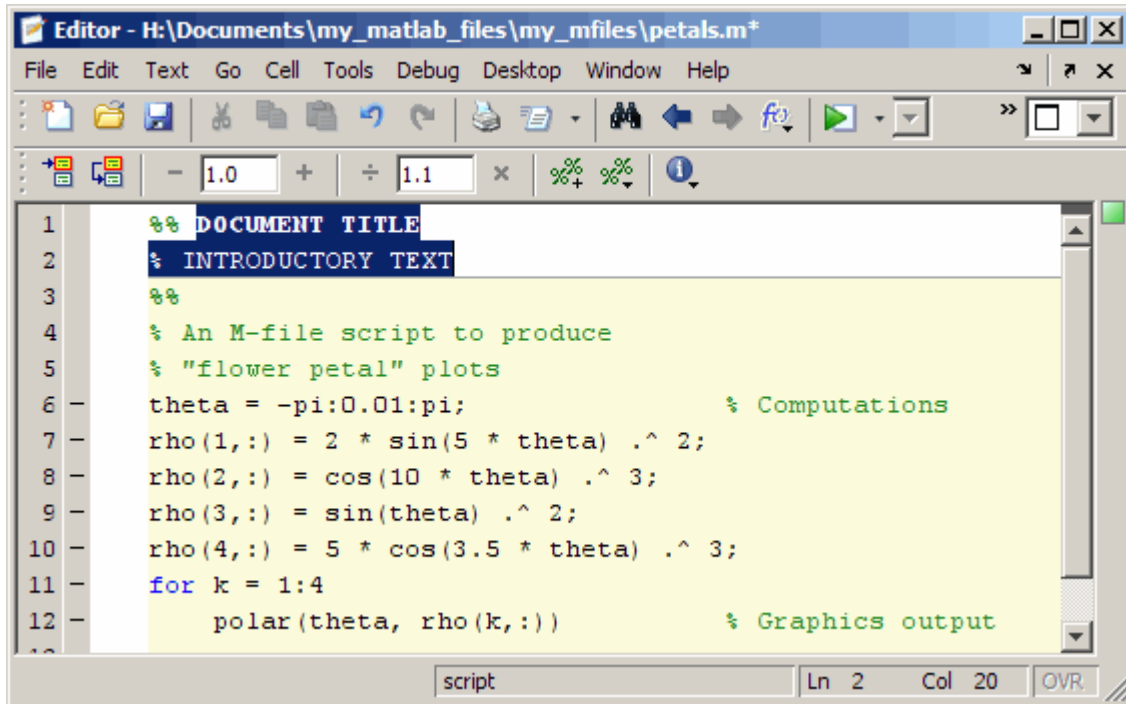
The following example shows how you might use **Cell** menu options with an existing M-file.

Suppose an M-file currently appears in the Editor as shown in the following image.

If you position the cursor anywhere within the file and select **Cell > Insert Text Markup > Document Title and Introduction**, the M-file looks like the following.

The file already contains a comment with introductory text, so you can delete the `% INTRODUCTORY TEXT` line and the double percent sign (`%%`) line, so the code appears as follows.

```
1     %% DOCUMENT TITLE
2     % An M-file script to produce
3     % "flower petal" plots
4  -  theta = -pi:0.01:pi;                    % Computations
5  -  rho(1,:) = 2 * sin(5 * theta) .^ 2;
6  -  rho(2,:) = cos(10 * theta) .^ 3;
7  -  rho(3,:) = sin(theta) .^ 2;
8  -  rho(4,:) = 5 * cos(3.5 * theta) .^ 3;
9  -  for k = 1:4
10 -      polar(theta, rho(k,:))               % Graphics output
11 -      pause
12 -  end
```

Summary of Markup for Formatting M-Files for Publishing

The following two tables provides a summary of the text markup that you can type into an M-file to achieve the same results as using the **Cell > Insert Text Markup** menu options. These tables are particularly useful if you are not using the MATLAB Editor, or if you do not want to use menus to apply formatting. For a description of the **Cell** menu options, see "Formatting M-File Comments for Publishing" on page 8-18.

Summary of Markup Not Requiring a Formatted Block

This table summarizes markup that does not require that it be included within a formatted block.

Result in Published Document	Example of Corresponding M-File Markup
Document title and introduction	```%% DOCUMENT TITLE``` ```% INTRODUCTORY TEXT```
Section title and description	```%% SECTION TITLE``` ```% DESCRIPTIVE TEXT```
Section title without cell break	```%%% SECTION TITLE``` ```% DESCRIPTIVE TEXT```
Cell break without title or description	```%%```
Bold text	```% *BOLD TEXT*```
Italic text	```% _ITALIC TEXT_```
Monospaced text	```% |MONOSPACED TEXT|```
Hyperlinked text	```% <http://www.mathworks.com MathWorks>```
Trademark symbol	```% TEXT(TM)```
Registered trademark symbol	```% TEXT(R)```
Code to force a snapshot of the current output	```snapnow;```

Summary of Markup Requiring a Formatted Block

This table summarizes markup that requires that it be included within a formatted block. See "About Formatted Blocks" on page 8-54 for details.

Result in Published Document	Example of Corresponding M-File Markup
Image	`% <<FILENAME.PNG>> %`
Bulleted list	`% * ITEM1` `% * ITEM2`
Numbered list	`% # ITEM1` `% # ITEM2`

HTML markup	`% <html>`
	`% <table border=1><tr>`
	`% <td>one</td>`
	`% <td>two</td></tr></table>`
	`% </html>`

LaTeX markup	`% <latex>`
	`% \begin{tabular}{|r|r}`
	`% \hline $n$&$n!$\\`
	`% \\hline 1&1\\ 2&2\\ 3&6\\`
	`% \\hline`
	`% \end{tabular}`
	`% </latex>`

TeX equation	`% $$e^{\pi i} + 1 = 0$$`

Formatting M-File Code for Publishing

In this section...
"Overview of Formatting M-File Code for Publishing" on page 8-64
"Example of Published M-File Output" on page 8-64

Overview of Formatting M-File Code for Publishing

This section describes ways to control how output that the MATLAB software generates when it evaluates executable M-file code appears in a published document. For example, you can direct MATLAB to include the last, or all plots generated by a `for` loop. You can interweave comments, code, and output throughout your published document to draw your readers' attention to certain areas of interest.

The tool you use to specify how output is presented in the document is the same tool you use to specify document titles and section headers; namely the double percent sign (%%) which is referred to as a cell break. When you insert a cell break into a file, it directs MATLAB to publish the code and output contained in the cells created by the break. Because MATLAB considers the entire M-file to be a cell, when you insert a cell break, MATLAB considers the file to contain two cells; one above the cell break and one below. The examples in the remaining topics demonstrate how you can use this behavior to control the output produced by M-file code.

Example of Published M-File Output

This section provides an example to demonstrate how an M-file appears when published. It demonstrates how the published example file appears before and after cell breaks are added to achieve the published results.

Sample M-File Before Inserting Cell Breaks in Code

Suppose your M-file contains the following code:

```
%% Scale magic Data and Display as Image

for i=1:3
    imagesc(magic(i))
```

```
end
```

The following image illustrates how the code presented appears when you publish it to HTML. The plot in the figure is smaller than it appears if you publish the M-code using factory default settings. For information on setting publishing properties for images, see "Producing Published Output from M-Files" on page 8-68.

Notice that the published document displays the plot after the end of the for loop and that only the last plot generated by the code is included.
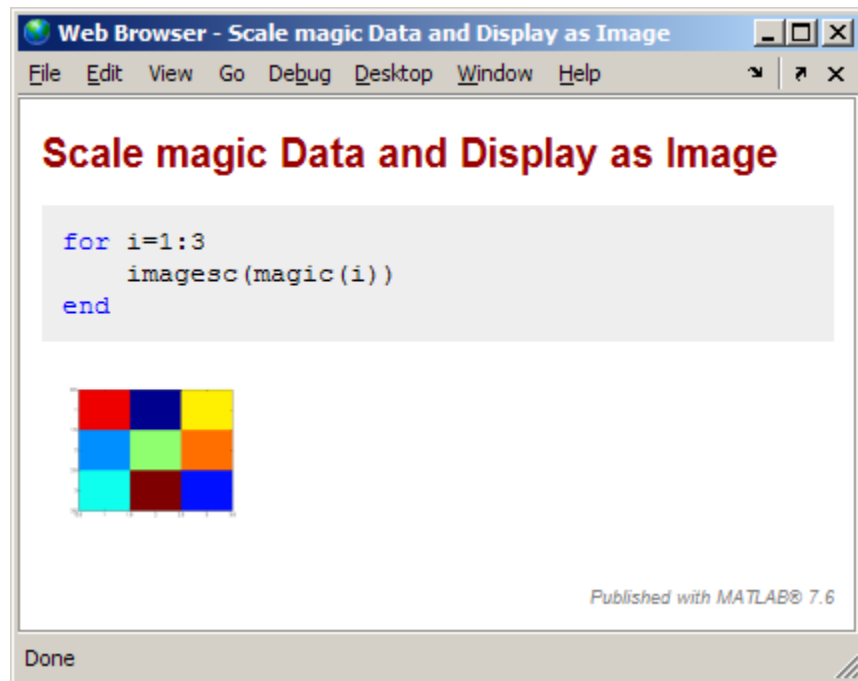
Sample M-File After Inserting Cell Breaks in Code

By placing cell breaks within a loop, you can display the output generated by M-file code when iterating a loop.

To include the plot generated by each iteration of the loop in the published document, insert a cell break after the opening for statement. Position the

cursor at the end of the first line of the `for` loop, and then select **Cell > Insert Cell Break**.

The code now appears like this:

```
%% Scale magic Data and Display as Image

for i=1:3
    %%
    imagesc(magic(i))
end
```

Now when you publish the code to HTML, it appears as follows. The plots in the figure are smaller than they appear if you publish the M-code using factory default settings. For information on setting publishing properties for images, see "Producing Published Output from M-Files" on page 8-68.

Notice that the published document displays the plot within the `for` loop code. You can also use text markup for similar results with figures. See "Formatting M-File Code for Publishing" on page 8-64 for details.
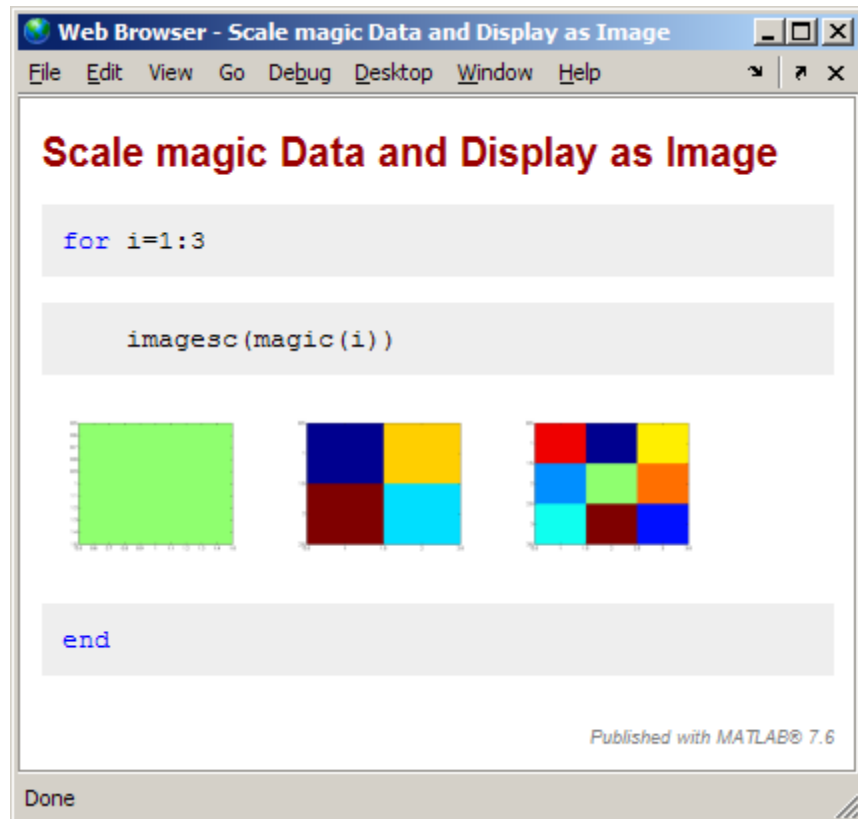
Producing Published Output from M-Files

In this section...

About Producing Published Output

Once you have formatted an M-file for publishing, as described in "Formatting M-File Comments for Publishing" on page 8-18 and "Formatting M-File Code for Publishing" on page 8-64 you are ready to publish it. The easiest method for publishing an M-file is to use factory defaults. This method is appropriate if your M-file requires no input arguments and you want to publish to HTML. However, if your M-file requires input arguments, or you want to specify preferences for publishing, such as the output directory, output format, image format, and so on, you need to specify custom property settings.

Publishing M-Files Using No Input Arguments and Factory Default Settings

To publish a script M-file, or a function M-file that requires no input arguments:

1 Open the file in the Editor.

2 Click the Publish button ⬚ on the Editor toolbar.

By default, the Editor publishes the file using factory default settings. Factory default settings specify that the output file format is HTML, that the M-code is evaluated and included in the published output file, and so on.

If the file is neither in a directory on the search path, nor in the current directory, a dialog box opens and presents you with options that allow you to publish the file. You can either change the current directory to the directory containing the file, or you can add the directory containing the file to the MATLAB search path.

If the file has unsaved changes, publishing it from the Editor automatically saves the changes before publishing.

Using Publish Configurations to Publish M-Files with Input Arguments or Custom Settings

Using a publish configuration, you can specify custom settings, including input arguments for a function M-file in the Editor. You can associate multiple publish configurations with an M-file for different publish settings, input arguments, or both. MATLAB saves the configuration between sessions.

For example, the function `collatzplot_new.m`, which computes and plots the Collatz sequence for any given positive integer, requires you to specify the integer as an input value. You cannot simply publish `collatplot_new.m` because the input value is not defined. A publish configuration enables you to publish `collatzplot_new(`*specific value*`)`.

You can also use publish configurations to provide preparatory or setup information prior to publishing an M-file, whether it takes input arguments or not.

Note M-file publish configurations use the base MATLAB workspace. Therefore, a value that you assign to a variable in an M-file publish configuration overwrites the value for that variable (assuming it currently exists) in the base workspace.

Function Alternative to Publishing

From the Command Window, execute the `publish` function to run the M-file and publish the results. See the `publish` function reference page for options you can set.

Creating a Publish Configuration for an M-File

Follow these steps to create a publish configuration for an M-file in the Editor. The example in this section shows how to create and use a publish configuration to specify input arguments to a function M-file.

These steps specify Editor toolbar buttons, but you can also use equivalent items in the **File** menu.

1 Open the file that you want to publish in the Editor. This example uses the code that follows. This code is similar to the `sine_wave.m` file, after it has been formatted as described in "Formatting M-File Comments for Publishing" on page 8-18, but it is slightly altered to make it a function M-file. Save the code as `sine_wave_f.m`

```
%% Plot Sine Wave
% Calculate and plot a sine wave.

%% Calculate and Plot Sine Wave
% Calculate and plot |y = sin(x)|.

function sine_wave_f(x)

y = sin(x);
plot(x,y)

%% Modify Plot Properties

title('Sine Wave', 'FontWeight','bold')
xlabel('x')
ylabel('sin(x)')
set(gca, 'Color', 'w')
set(gcf, 'MenuBar', 'none')
```

2 Click the down arrow next to the **Publish** button on the Editor toolbar, and click **Edit Publish Configuration for *file name***, where file name in this example is sine_wave_f.m.
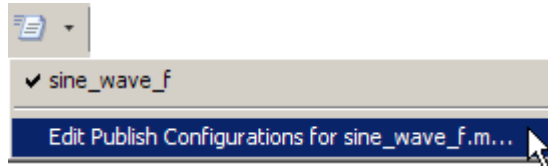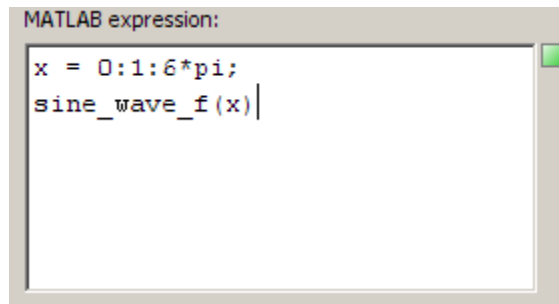
The Edit M-File Configurations dialog box opens, with the default publish configuration template for sine_wave_f.m, as shown in the following figure.
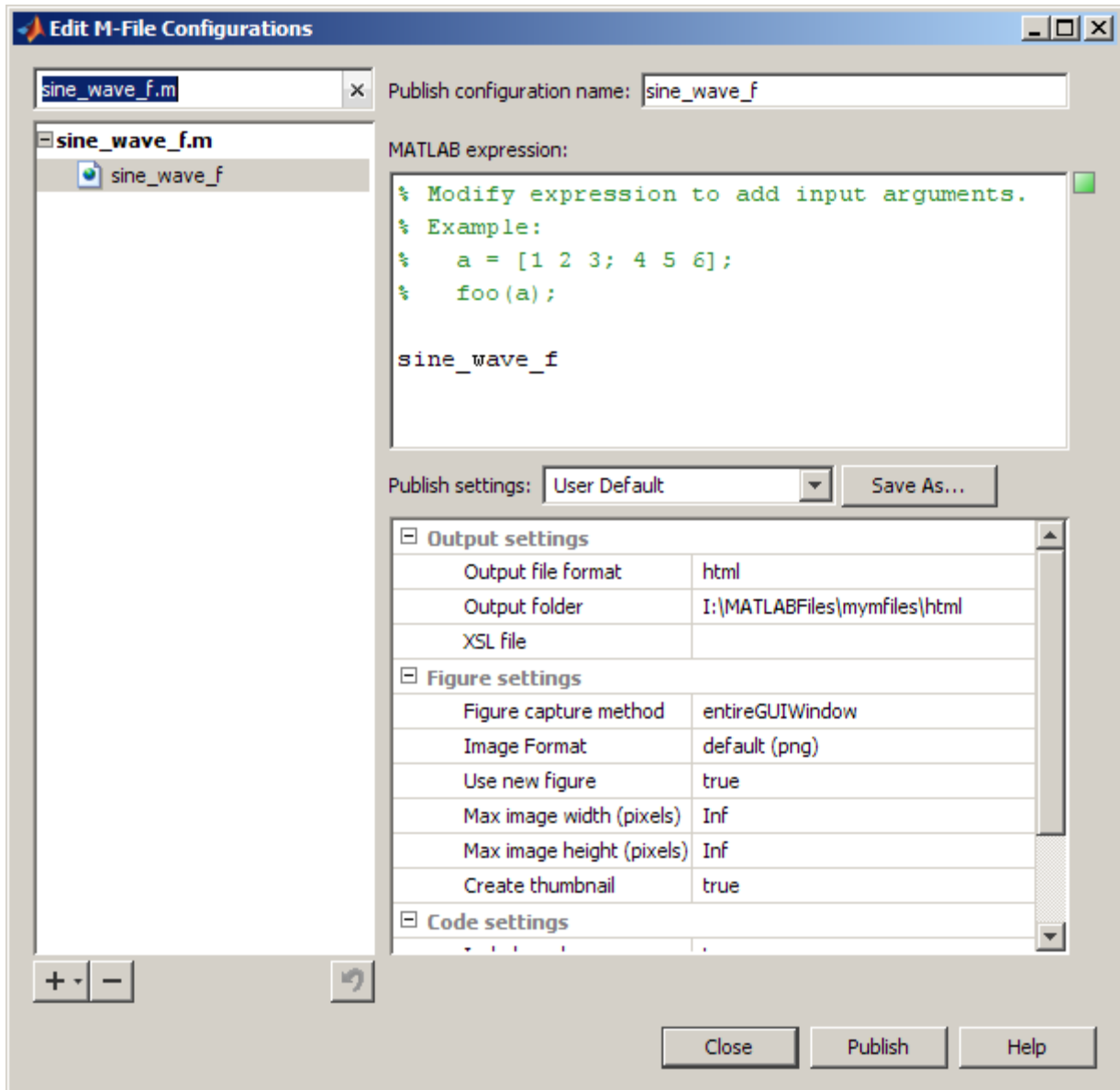
3 In the **Publish configuration name** field, type a name for the publish configuration, or accept the default name.

If you expect to create multiple configurations for an M-file, assign each a name that helps you identify the configuration. In this figure, the default name of the configuration is sine_wave_f.

4 In the **MATLAB expression** field, type the expression that you want the Editor to evaluate when it publishes the M-file. In this example, delete the commented statements and replace them as shown in the following figure.

MATLAB expression:

```
x = 0:1:6*pi;
sine_wave_f(x)
```

You can modify the statements in the **MATLAB expression** area of the dialog box, and then click **Publish** to see the results of the changes. If you clear the **MATLAB expression** area, MATLAB publishes the M-file without evaluating any code. This is equivalent to setting the **Evaluate code** property in the **Publish settings** properties table to false.

5 In the **Publish settings** properties table, change the property values if you do not want to use the current settings.

You can modify the property settings, and then click **Publish** to see the results of the changes.

See "Specify and Save Publish Configuration Settings" on page 8-74 for details.

6 Do one of the following:

- To publish the file using the settings and MATLAB expression that you have specified, click **Publish**.

  For this example MATLAB creates the following files in I:\my_matlab_files\my_mfiles\html, which is a subdirectory in the directory where sine_wave_f.m is located:

  – A published document file, `sine_wave_f.html`

  – A thumbnail file for the last image generated by the M-file code, `sine_wave_f.png`

  – Image files created by the executable M-file code, `sine_wave_f_##.png`

- To create another publish configuration for the same M-file, click the plus button ✚▾, and then select **Publish Configuration**.

  See "Create and Run Multiple Publish Configurations for an M-File" on page 8-98 for details.

- To close the Edit M-File Configurations dialog box, click **Close**. MATLAB saves the configuration and its association with the M-file.

After creating a configuration, you can view the MATLAB expression and use the configuration to publish the M-file without opening the Edit M-File Configurations dialog box. See "Run an Existing Publish Configuration" on page 8-96 for details.

Specify and Save Publish Configuration Settings

This section describes how to specify new publish settings for a configuration. Publish settings enable you to specify the directory to which a published file is saved, how images generated by the M-code are captured and included in the published document, and so on.

1 If the Edit M-File Configurations dialog box is not already open, click the down arrow on the **Publish** button 📄▾ and then click the configuration that you want to change.

  This example uses the `sine_wave_f` publish configuration as described in "Creating a Publish Configuration for an M-File" on page 8-70.

2 View the properties table below the **Publish settings** field to see the current publish property values.

3 For information about a property, click the property name. A brief description of that property displays below the publish settings property table. For example, if you click **Catch error**, the dialog box appears as shown in the following image.
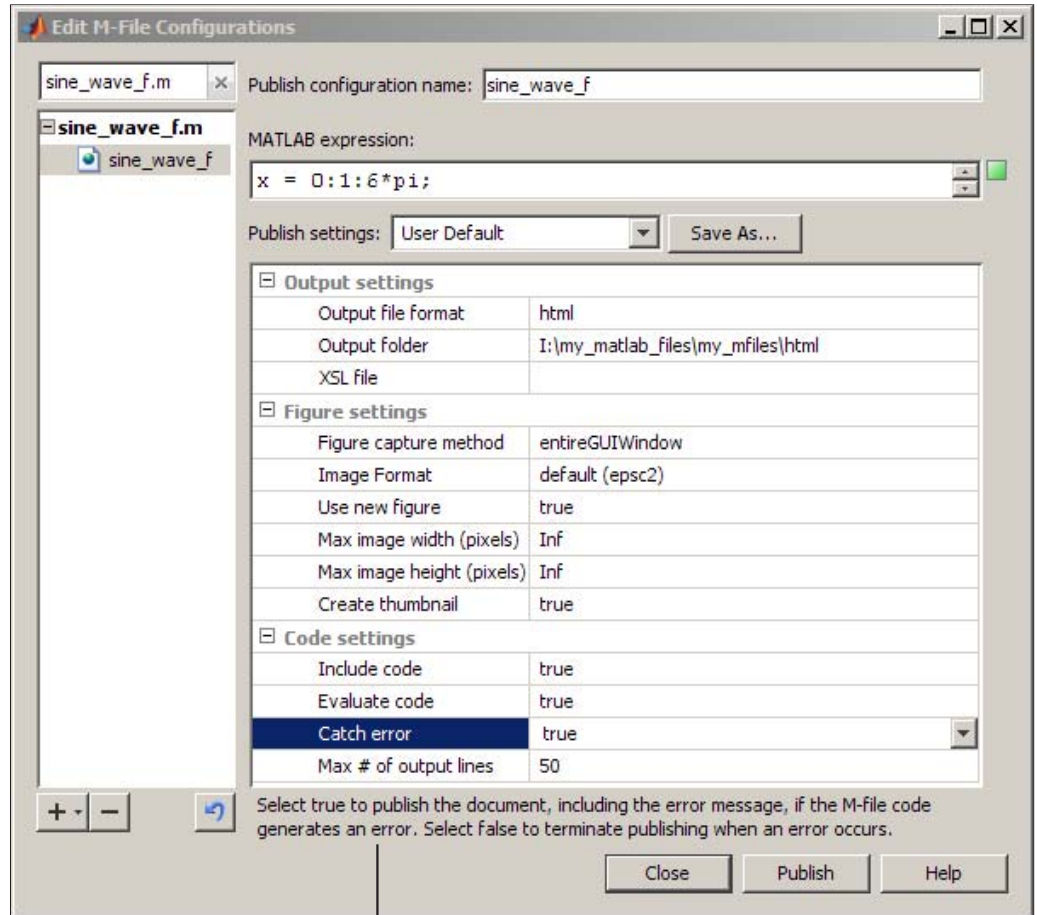
Description of Catch error property

4 Optionally, you can change publish setting values by clicking in the column to the right of the property name and then entering or selecting a property value. This example changes **Max image width** and **Max image height** to 400.

The Editor marks each property that you change with a dot (●) and adds the string, (modified), next to User Default in the **Publish settings** field.

See "Specify Values for the Publish Settings Property Table" on page 8-78 for information about the various properties you can set.

Indicates that one or more settings differ
from the saved User Default property settings

Indicates that these properties' values differ from the settings
saved in the User Default publish settings.
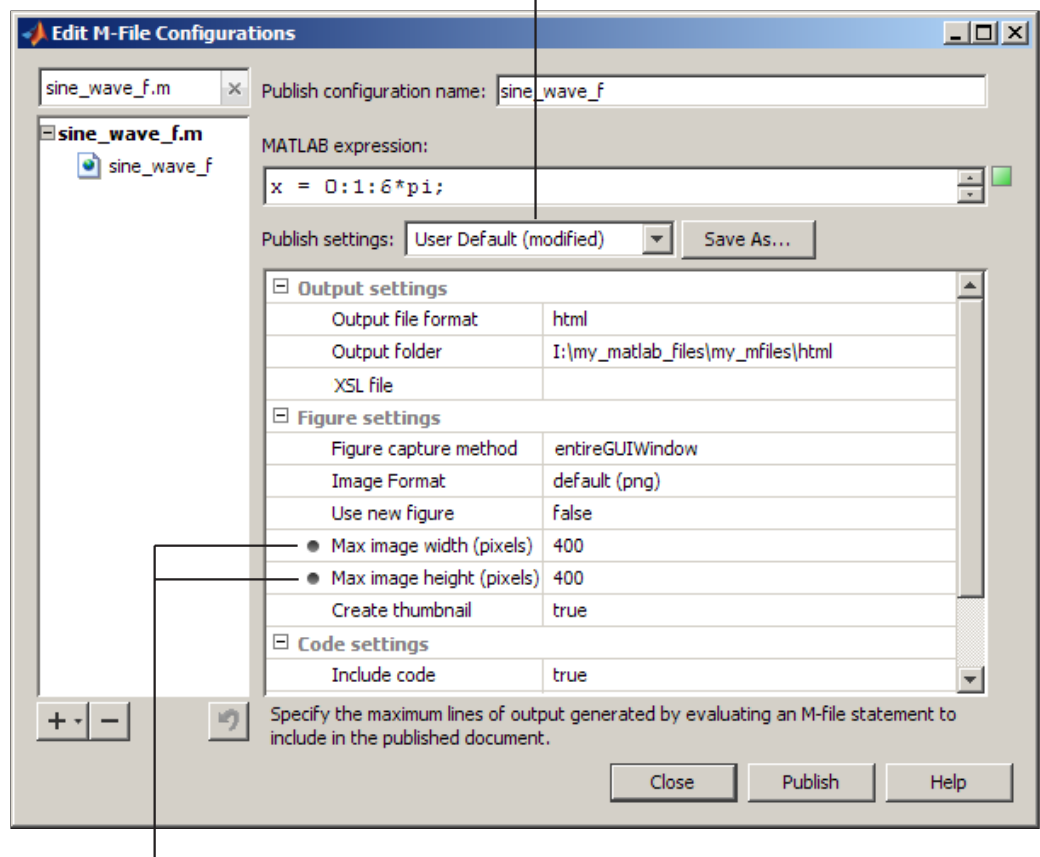
5 Click **Publish** to preview the publication of the M-file that is open in the Editor using the new settings.

6 When you are satisfied with the results, click **Save As**.

The Save Publish Options dialog box opens and displays the names of all the currently defined publish settings. By default the following publish settings are installed with MATLAB:

- `Factory Default`

  The MATLAB installation includes this named set of publishing properties for you to get started with publishing documents. It enables you to quickly publish an M-file to HTML and view the results. You can use it to test the effect of changing settings on the published output. If you determine that the test settings produce undesirable results, you can restore the `Factory Default` publish settings by selecting it from the **Publish settings** drop down list.

- `User Default`

  The MathWorks installs this named set of publishing properties in anticipation that you will have a set of publish properties that are common to most or all of your publishing configurations. Initially, `User Default` settings are identical to those in the `Factory Default`. See "Creating a Template for Typical Publish Settings" on page 8-93 for an example of changing the User Default settings to best suit your publishing needs.

7 In the **Settings Name** field, enter a meaningful name for the settings. For example, `reduce_image`. Then click **Save**.

You can now use the `reduce_image` publish settings with other publish configurations.

You can also overwrite the publishing properties saved in an existing publish settings name by selecting it from the **Publish settings** drop-down list, and then clicking **Overwrite**. However, you cannot overwrite the `Factory Default` publish settings.

Note When you overwrite a publish settings name, configurations that currently specify that publish settings name do not have their publish properties updated to use the new settings. Instead, their properties that now have different values from the updated publish settings are marked with a dot.

---

8 In the Edit M-File Configurations dialog box, do one of the following:

- Click **Publish** to publish the M-file that is open in the Editor using the new settings.

- Click **Close** to close the dialog box.

Specify Values for the Publish Settings Property Table

The sections that follow describe each of the publish settings properties that you can adjust to fit your needs when you create or update a publish configuration. To access a publish configuration open the M-file for which you want create or update a publish configuration, and then select **File > Publish Configuration for *file name* > Edit Publish Configurations for *file name***.

You can set or adjust values for the following properties:

- "Output file format Property" on page 8-78
- "Output folder Property" on page 8-79
- "XSL file Property" on page 8-79
- "Figure capture method Property" on page 8-79
- "Image format Property" on page 8-84
- "Use new figure Property" on page 8-84
- "Max image width Property" on page 8-90
- "Max image height Property" on page 8-90
- "Create thumbnail Property" on page 8-91
- "Include code Property" on page 8-91
- "Evaluate code Property" on page 8-91
- "Catch error Property" on page 8-93
- "Max # of output lines Property" on page 8-93

Output file format Property

Select one of the choices from the drop-down list to publish the document to one of the following file formats:

- `html` — Publishes an HTML document.

- `xml` — Publishes an XML document.

- `latex` — Publishes a LaTeX document, which you can use to create a PDF document, if you want.

- `doc` — Publishes a Microsoft Word document, if your system is a PC.

- `ppt` — Publishes a Microsoft PowerPoint document, if your system is a PC.

MATLAB names the published file with the same name as the publish configuration that produced it and stores it, along with images that MATLAB generates from M-file code, in the directory specified with the **Output folder** property.

Output folder Property

Type the full path of the directory to which you want MATLAB to publish the output document and its associated image files. For example, if your M-file is in `I:\my_matlab_files\my_mfiles`, you might specify `I:\my_matlab_files\my_word_files` if you are creating a publish configuration for documents that you publish to Word.

XSL file Property

Type the full path of the Extensible Stylesheet Language (XSL) file, that you want to use when you specify the **Output file format** as **HTML**, **XML**, or **LaTeX**. If you leave this field blank, MATLAB uses a default stylesheet which is installed with the MATLAB software.

Figure capture method Property

Specify a figure capture method to indicate how you want figures and dialog boxes that the M-file code creates to appear in published documents.

The following list provides some suggestions on how to set this property, depending on the type of document you are publishing. (The document shown in the images that follow was published with the **Max image width** property set to 150 pixels.)

- To create a published document that presents the Figure data and complete dialogs boxes that the M-file creates, set the **Figure capture method** to **entireGUIWindow**.

  Notice that this method sets the figure background to white, presents just the plot for the figure, but includes the window decorations (title bar) for the dialog box in the published document.
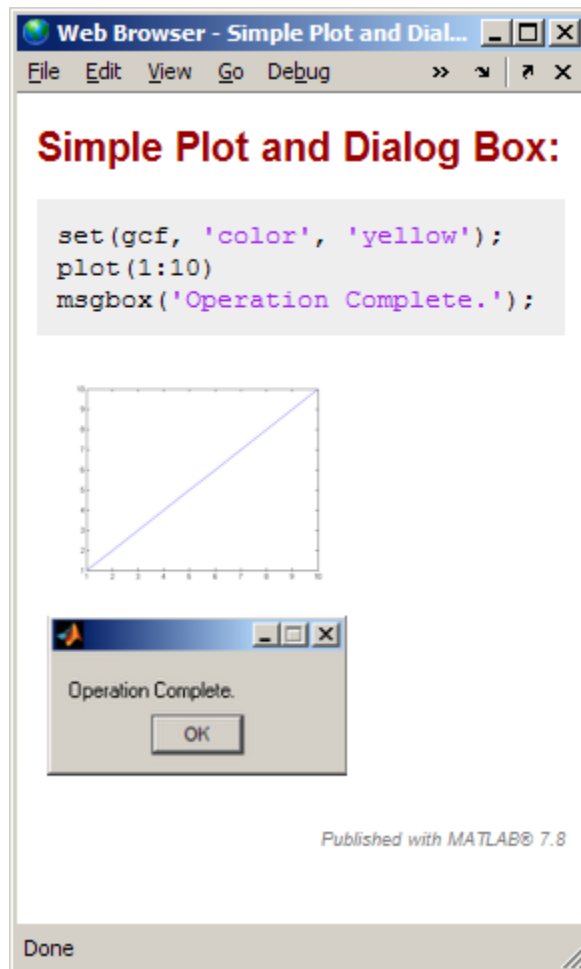
- To create a published document that presents a tutorial on using MATLAB, set the **Figure capture method** to **entireFigureWindow**.

  Notice that this method preserves the background color for figures and includes the window decorations for both figures and dialog boxes in the published document.
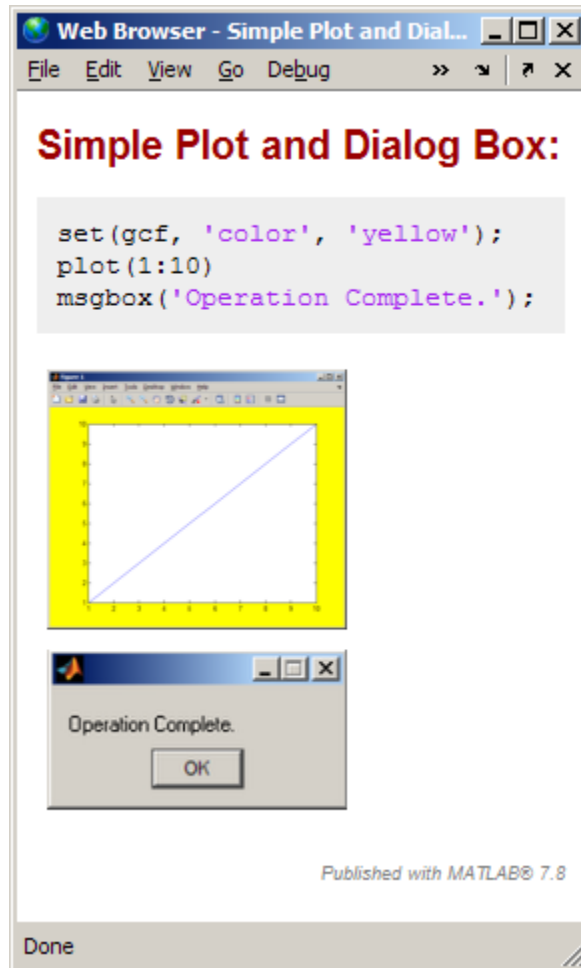
- To create a published document that presents figures with the plot background set to white and dialog boxes without window decorations, set the **Figure capture method** to **print**.
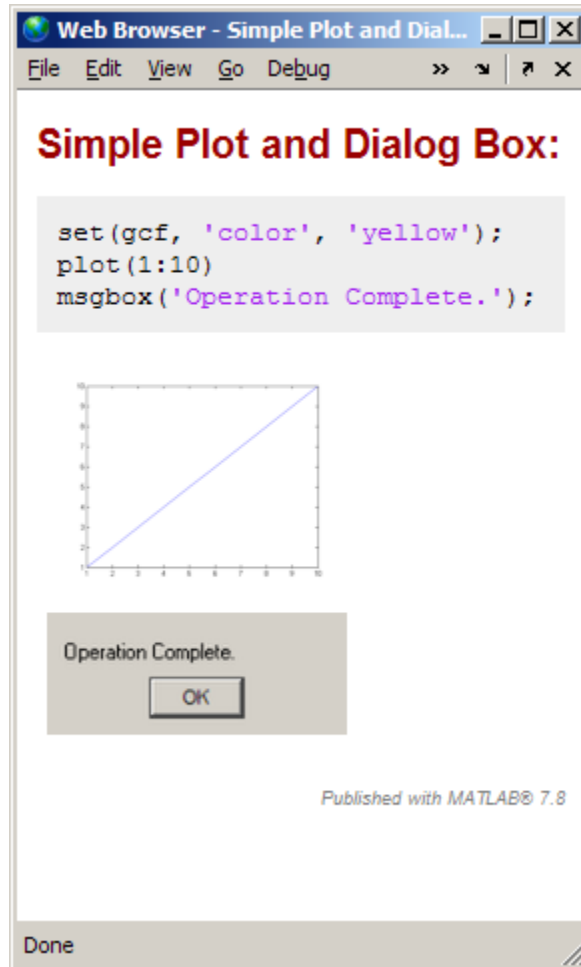
- To create a published document that presents figures and dialog boxes excluding window decorations, but including the background color for figures, set the **Figure capture method** to **getframe**.
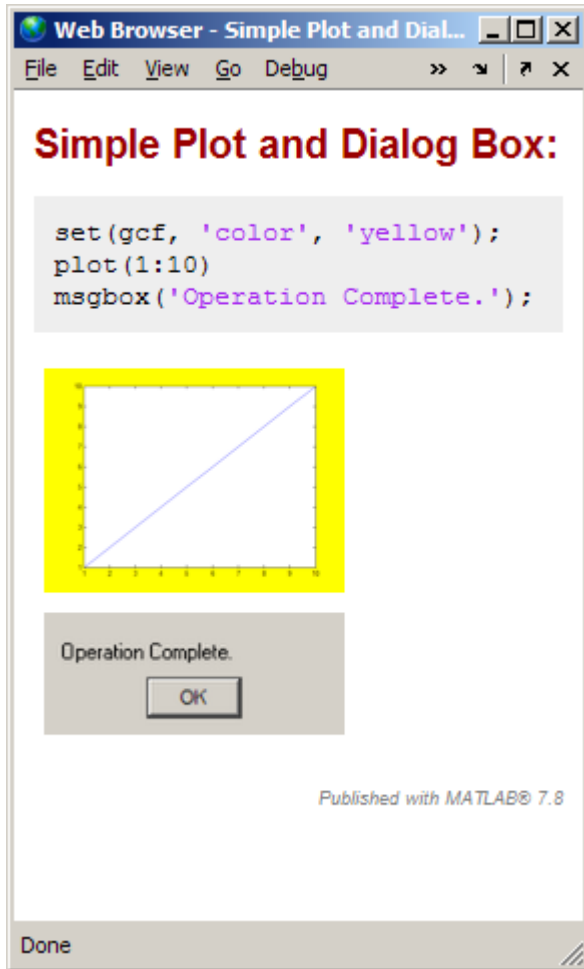
The following table summarizes the effects of the various Figure capture methods.

To Get Figure Captures with these Appearance Details		Use this Figure Capture Method
Window Decorations	**Plot Backgrounds**	
Included for dialog boxes; Excluded for figures	Set to white for figures; matches the screen for dialog boxes	**entireGUIWindow**
Excluded for dialog boxes and figures	Set to white	**print**
Excluded for dialog boxes and figures	Match the screen plot background	**getframe**
Included for dialog boxes and figures	Match the screen plot background	**entireFigureWindow**

Note Typically, MATLAB figures have the HandleVisibility property set to on. Dialog boxes are figures with the HandleVisibility property set to off or callback. If your results are different from those listed in the preceding list and table, the HandleVisibility of your figures or dialog boxes might be atypical. For more information, see "HandleVisibility Property" in the MATLAB Graphics documentation.

Image format Property

Select the file type for images produced when publishing M-files. The image file types available in the drop-down list depend on the **Figure capture method** you specify.

Use new figure Property

Set to **true** if you want MATLAB to create a new Figure window with a white background and at the default size before publishing if the M-file code generates a figure. After publishing finishes, MATLAB closes the Figure window.

To use a figure with different properties for publishing, set this property to
false. Then open a Figure window, change the size and background color,
for example, and then publish. Figures in your published document use the
characteristics of the figure you opened before publishing.

Note This preference applies to executable M-file code that generates a
figure. It does not apply to figures included in a published document using the
Cell > Insert Text Markup > Image menu option.

The following example demonstrates how to specify new Figure window
properties for published images by setting the **Use new figure** publish
settings property to **false**:

1 Create sine_wave_f.m, as described in "Creating a Publish Configuration
for an M-File" on page 8-70.

2 Create a Figure window by saving the following code in an M-file and then
running it:

```
function createfigure
%CREATEFIGURE

% Create figure
figure1 = figure('Name','purple_background',...
'Color',[0.4784 0.06275 0.8941]);
colormap('hsv');

% Create subplot
subplot(1,1,1,'Parent',figure1);
box('on');

% Create xlabel
xlabel({''});

% Create title
title({''});
```

The following figure appears.
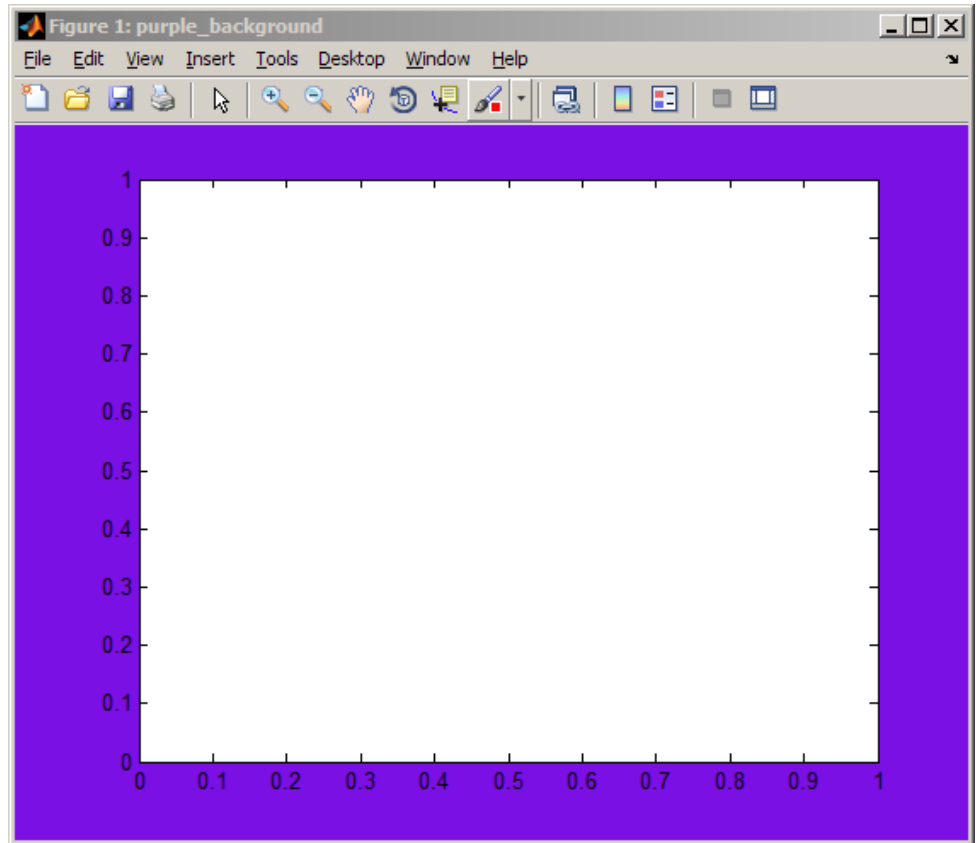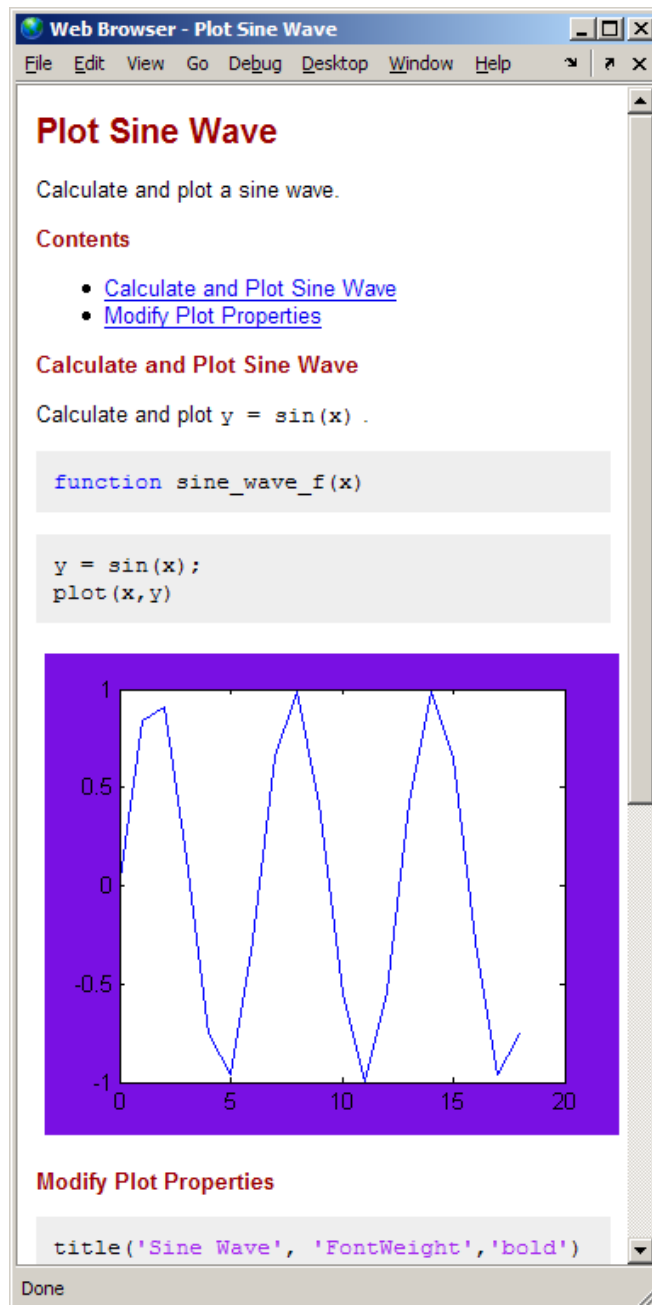
3 Reduce the size of the figure by dragging and dropping the edges. For example:
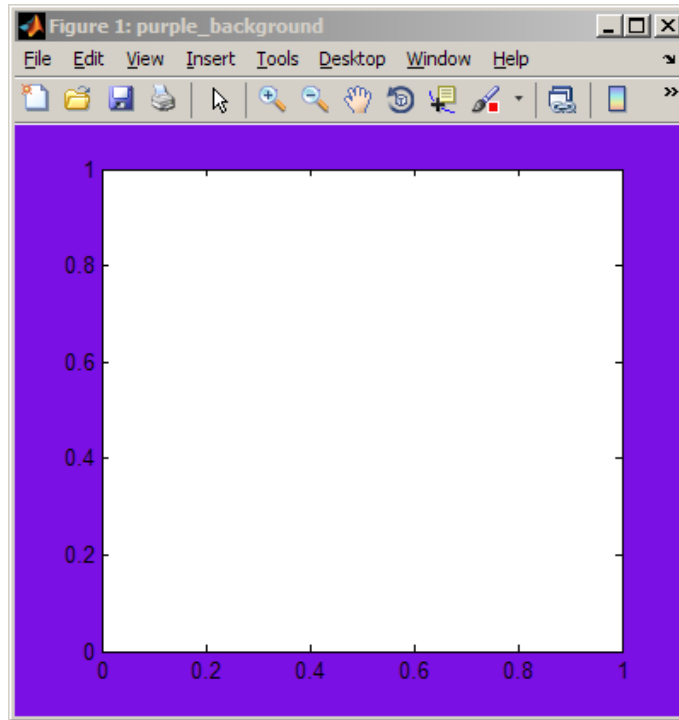
4 Do not close the window.

5 Make sine_wave_f.m the active file in the Editor, and then select
**File > Publish Configurations for sine_wave_f.m > Edit Publish
Configurations for sine_wave_f.m.**

6 In the **Publish settings** drop-down list, select Factory Default.

7 If you have previously set **Publish settings** for sine_wave_f.m, the
Change Publish Settings dialog box opens. Click **Change to Factory
Default**.

8 In the **Publish settings** properties table, set **Use new figure** to **false**.

9 Click **Publish**. MATLAB publishes `sine_wave_f.m` as shown in the following figure.
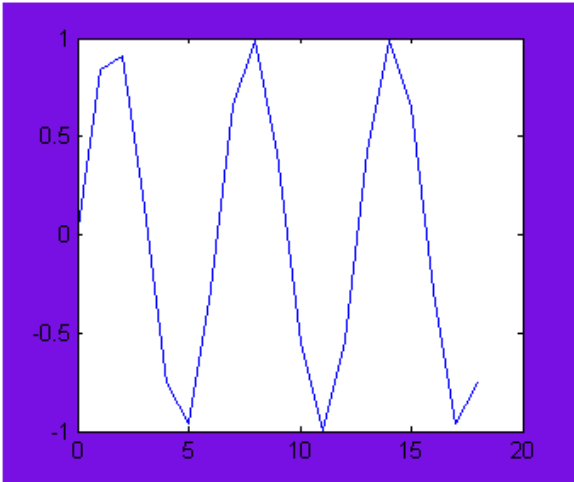
Max image width Property

Overwrite the current value to restrict the width of images in the published output. Note the following about this property:

- It applies only to images that the M-code generates. It does not apply to images you include using the method described in "Specifying Graphics in M-Files for Publishing" on page 8-31.

- It applies when you select an **Image Format** property setting that is a bitmap, such as `.png` or `.jpg`.

- It does not apply when the **Image Format** property setting is a vector format, such as `.eps`.

- The image's aspect ratio is maintained. If you restrict both height and width using **Max image width** and **Max image height** properties to resize the image, then MATLAB maintains the aspect ratio by using the maximum you specified for one dimension and something less than the maximum for the other dimension.

Max image height Property

Overwrite the current value to restrict the height of images in the published output. Note the following about this property:

- It applies only to images that the M-code generates. It does not apply to images you include using the method described in "Specifying Graphics in M-Files for Publishing" on page 8-31.

- It applies when you select an **Image Format** property setting that is a bitmap, such as `.png` or `.jpg`.

- It does not apply when the **Image Format** property setting is a vector format, such as `.eps`.

- The image's aspect ratio is maintained. If you restrict both width and height using **Max image width** and **Max image height** properties to resize the image, then MATLAB maintains the aspect ratio by using the maximum you specified for one dimension and something less than the maximum for the other dimension.

Create thumbnail Property

Set to **true** to direct MATLAB to create a thumbnail image if the **Image Format** preference is a bitmap, such as `.png` or `.jpg`. For example, you can use this thumbnail to represent your M-file in HTML pages. If you create your own demos and include them in the Help browser **Demos** pane via a `demos.xml` file, MATLAB automatically creates a list of your demos that includes the thumbnail for each.

Set to **false** to direct MATLAB to not create a thumbnail image.

Include code Property

Set to **true** to have MATLAB include the M-file code in the published document. Set to **false** to the have MATLAB exclude the code from all output types except HTML. When the output type is HTML, MATLAB inserts the M-file code in the output file as an HTML comment. Therefore, when viewed in a Web browser, for example, the M-file code is not displayed.

Use the MATLAB `grabcode` function if you want to extract the M-file code from the published HTML file.

For example, suppose you publish `I:/my_matlabfiles/my_mfiles/sine_wave_f.m` to HTML using a publish configuration with the **Include code** property set to **false**. If you share the published document with colleagues, they can view the published document in a Web browser. If your colleagues want to see the M-file code that generated the published document, they can issue the following command from the directory containing `sine_wave_f.html`:

```
grabcode('sine_wave_f.html')
```

MATLAB opens the M-file code that created `sine_wave_f.html` in the Editor.

See "Creating a Publish Configuration for an M-File" on page 8-70 for the `sine_wave_f.m` code.

Evaluate code Property

Set to **true** to direct MATLAB to evaluate the M-file code while publishing the results and include the results in the output document.

Set to **false**, to direct MATLAB to not evaluate the code nor include code results when publishing an M-file.

Because MATLAB does not evaluate the code, there might be invalid code in the M-file. Therefore, you might not want to set this property to **false** without first running the M-file.

For example, suppose you include comment text, `Label the plot`, in an M-file, but forget to preface it with the comment character. If you publish the document to HTML, and set **Evaluate code** to **true**, the published document includes the error, such as shown in the following figure.

Use this property with the **Max # of output lines** property to specify the maximum number of lines you want to include in the output. This property is

helpful when you have code that produces a lot of output and you only want to include a sample of it in the published document.

Catch error Property

Set to **true** to direct MATLAB to publish and include the error message text in the published document if an error occurs when it evaluates the code.

Set to **false** to direct MATLAB to terminate the publish operation if an error occurs when it evaluates the code.

This property has no effect if you set the **Evaluate code** property to false.

Max # of output lines Property

Type a value to specify the maximum number of output lines that you want to include after each cell break in the published document.

For example, suppose your M-file code includes a loop, such as the following:

```
for n = 1:100
  disp(x)
end;
```

If you publish the document, then by default, all 100 lines of the output generated by the preceding code is included in the published document. If you want to include a smaller representative sample of the output, set **Max # of output lines** to a small value, such as 10.

Creating a Template for Typical Publish Settings

Use the **User Default** publish settings installed with MATLAB to create a template for all or most of your publish configurations.

Initially, the User Default publish setting has the same property values as Factory Default publish settings. Update and save your most commonly used property settings to avoid having to reset the same settings each time you create a new publish configuration.

For example, suppose that you frequently publish your M-files using the factory installed `User Default` settings, with a few exceptions. You want to change the factory installed `User Default` settings to:

- Save the published document files to
  `I:\my_matlab_files\my_published_mfiles`

- Use the `getframe` figure capture method

- Terminate publishing if an error occurs while the M-file code is being evaluated

Update the `User Default` publish settings, as follows:

1 If the Edit M-File Configurations dialog box is not already open, click the down arrow on the **Publish** button , and then click the configuration for which you want to set the properties as described in the preceding list.

2 From the **Publish settings** drop-down list, select `User Default`.

   If the Change Publish Settings dialog box opens, click **Change to User Default**.

3 Adjust the values in the publish settings properties table, so that the **Publish settings** appear as shown in the following figure.

Publish settings:	User Default (modified)	▼	Save As...

⊟ **Output settings**	
Output file format	html
● Output folder	I:\my_matlab_files\my_published_mfiles
XSL file	
⊟ **Figure settings**	
● Figure capture method	getframe
Image Format	default (png)
Use new figure	true
Max image width (pixels)	Inf
Max image height (pixels)	Inf
Create thumbnail	true
⊟ **Code settings**	
Include code	true
Evaluate code	true
● Catch error	false
Max # of output lines	50

4 Click **Save As**.

The Save Publish Settings dialog box opens.

5 In the **Publish settings** drop-down list, select User Default, and then click **Overwrite**.

The User Default publish settings are now saved with the specified property values.

Now, suppose you want to create a publish configuration using all the same settings, except you want to publish your M-file to a Microsoft Word document. Follow these steps:

1 In the Editor, open the M-file that you want to publish to a Word document.

2 Click the down arrow next to the Publish button ⬇️ on the Editor toolbar and click **Edit Publish Configuration for *file name***, where the file name is the name of the file that you want to publish to a Word document.

The Edit M-File Configurations dialog box opens.

3 If you want, adjust the MATLAB expression.

4 Notice that the **Publish settings** are set to **User Default** and the publish settings properties table contains the values you set in the preceding list of steps.

5 Change the **Output file format** from **html** to **doc**.

6 Click **Save As**.

The Save Publish Settings dialog box opens.

7 In the **Settings name** box, type a name for the new group of publish settings. For example, WordDefault.

8 Click **Save**.

Now you can use any one of the following as the publish settings, or the basis for new publish settings, for the next publish configuration you create:

- Factory Default
- Your customized User Default
- Word Default
- Any other publish settings that you create and save with a unique name

Run an Existing Publish Configuration

After creating a publish configuration, you can run the configuration without opening the Edit M-File Configurations dialog box, as follows:

1 In the Editor toolbar, click the down arrow on the Publish button **Publish** button ⬇️, and position the mouse pointer on a publish configuration name. MATLAB displays a ToolTip showing the publish configuration's

MATLAB expression so you can see what will be evaluated when you publish the M-file using the named configuration.
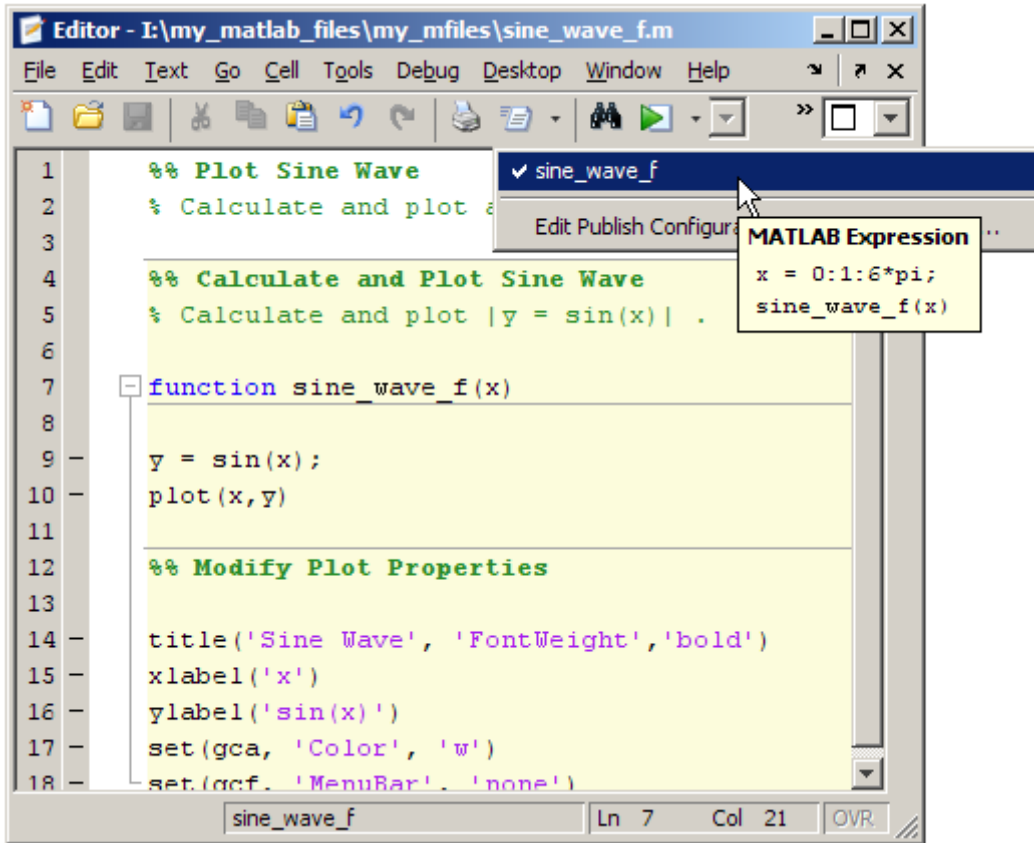
2 To use the publish configuration, select a configuration name. MATLAB publishes the M-file using the MATLAB expression you specified in the publish configuration. For example, if you select `sine_wave_f`, MATLAB sets the value of the input argument, `x`, to `0:1:6*pi` and passes it to the M-file function before evaluating and publishing it. (To see how to set the MATLAB expression, see "Creating a Publish Configuration for an M-File" on page 8-70.)

Create and Run Multiple Publish Configurations for an M-File

You can create multiple publish configurations for a given M-file. You might do this to publish the M-file with different values for input arguments, with different publish setting property values, or both. Create a named configuration for each purpose, all associated with the same M-file. Then, any time you publish the M-file, you can chose and run whichever particular publish configuration that you want. For example, for sine_wave_f(x) you might use different values for x and adjust publishing properties for these purposes:

- For reviewing with colleagues, publish the document to Word. Use publish settings to adjust the size of images generated by the code so they are not cropped in the document. Evaluate and include the code, as well as any errors generated by the code in the Word document.

- For inclusion in a blog, publish the document to HTML. Use publish settings to specify an argument value and set publishing properties to evaluate and include the code, but exclude errors generated by the code from the output published to HTML.

- For presentation at a meeting, use the same settings as used for publishing to the blog, but publish to Microsoft PowerPoint.

The following sections provide instructions for creating multiple configurations for sine_wave_f.m. Each set of steps assumes you have completed the previous set of steps. When you complete all three you will have three publish configurations, one for each output format described in the previous list.

- "Example of Publishing sine_wave_f.m to Microsoft Word" on page 8-98
- "Steps for Publishing sine_wave_f.m to HTML" on page 8-102
- "Steps for Publishing sine_wave_f.m to Microsoft® PowerPoint" on page 8-106

Example of Publishing sine_wave_f.m to Microsoft Word

The following steps provide an example of settings you might use when you want to publish an M-file to Word. This example uses the sine_wave_f.m file, the code for which is presented in "Creating a Publish Configuration for an M-File" on page 8-70.

1 In the Editor, open `sine_wave_f.m`.

2 Select **File > Publish Configuration for sine_wave_f.m > Edit Publish Configurations for sine_wave.m**.

3 Select `sine_wave_f` in the list of M-files and configurations, click the down arrow next to the **Add** button ✛▾, and then select **Publish Configuration**.
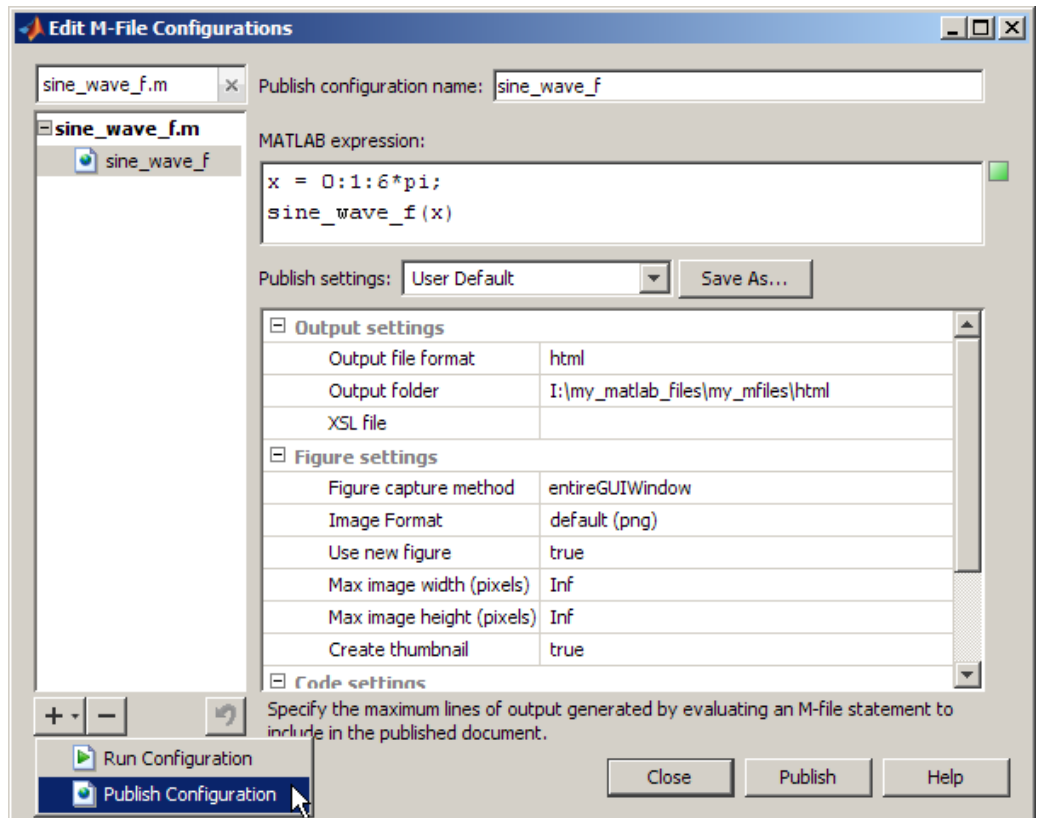
MATLAB creates a new publish configuration, `sine_wave_f_2`.
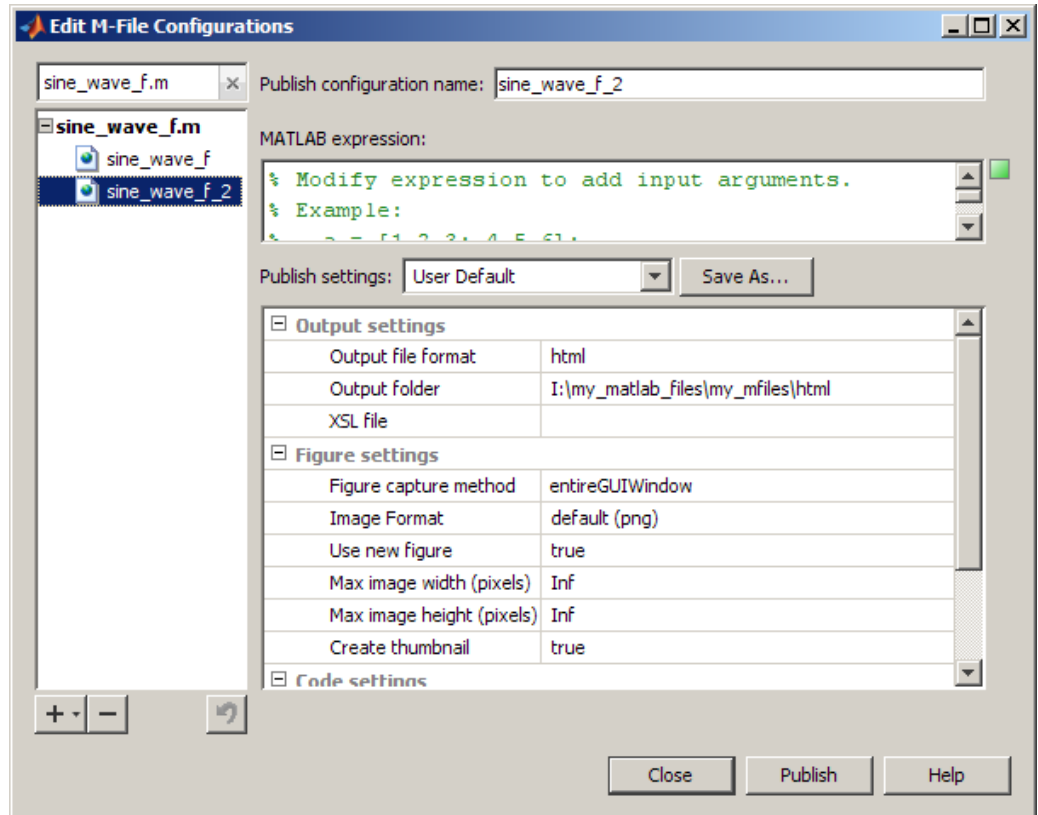
4 Rename sine_wave_f_2 to sine_wave_word, and replace the default template expression with the following code:

```
x = 0:1:rand*pi;
sine_wave_f(x)
```

5 Change the values for **Publish settings**, as follows so that the M-file is published to a Word document, including the code, its output and any errors the code may generate. The maximum values for the image height and width are set so that the images are not cropped in the Word document:

a For **Output file format**, select doc from the drop-down list.

b For **Image format**, select jpeg from the drop-down list.

c For **Max image width**, type 400.

d For **Max image height**, type 400.

6 Click **Publish** to test how the settings affect the Word document.

You can continue to test and change publish settings until you achieve the results that you want.

Tip In addition to testing that your M-file code evaluates as expected and publishes to Word as expected, you might run the spelling and grammar checker in Word to be sure that the comments in your M-file do not contain typographical or grammatical errors.

---

7 Click **Save As**. In the Save Publish Settings dialog box, in the **Settings name** field, type word_settings, and then click **Save**.
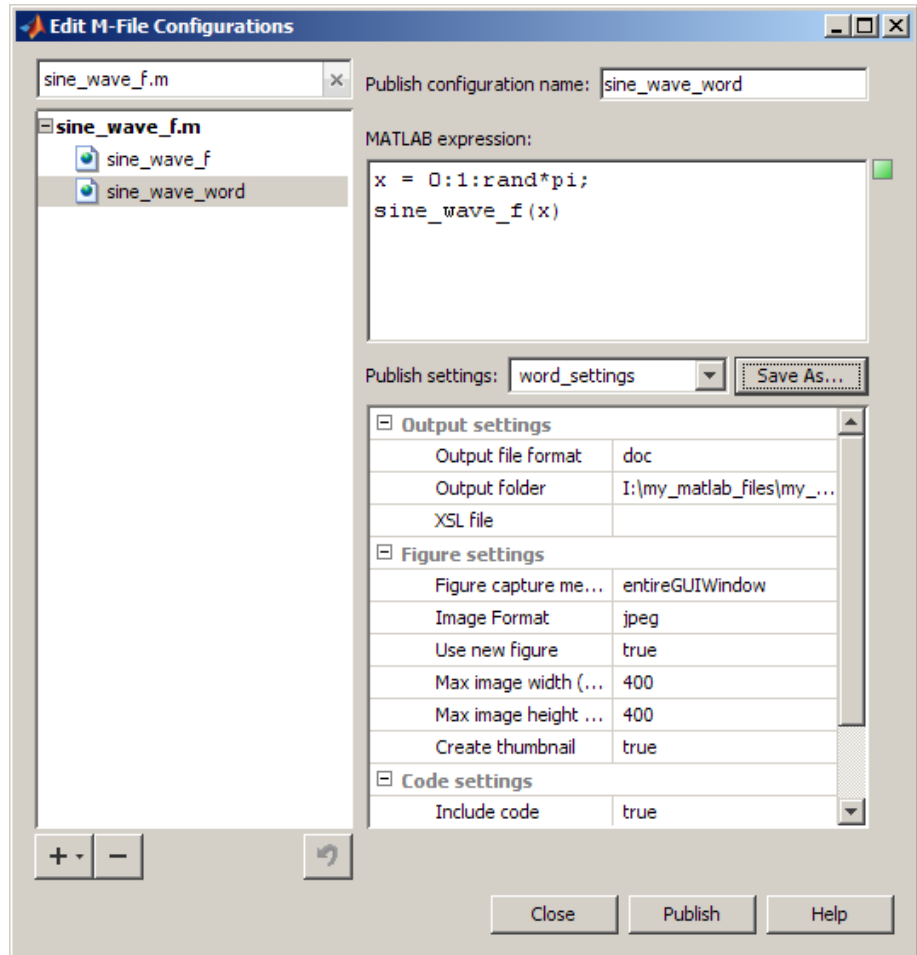
Steps for Publishing sine_wave_f.m to HTML

These steps provide an example of creating a configuration for sine_wave_f.m, that publishes the M-file to HTML. You might do this to publish output for inclusion in a blog, for example.

1 If it is not currently open, open the Edit M-File Configurations dialog box.

2 Select sine_wave_word in the list of M-files and configurations on the left side of the dialog box, click the **+** button, and then select **Publish Configurations**.

3 In the **Publish configuration name** field, replace sine_wave_f_2 with sine_wave_html.

4 In the **MATLAB expression** field, replace the default expression with the following:

```
x = 0:1:rand*pi;
sine_wave_f(x)
```

Tip To get a quick view of the expression used in a different configuration, position the mouse pointer on the name of a different publish configuration without selecting it. In the following figure, sine_wave_html is selected, but the mouse pointer is positioned on sine_wave_f. You can see the MATLAB expression specified for the sine_wave_f configuration in the ToolTip.
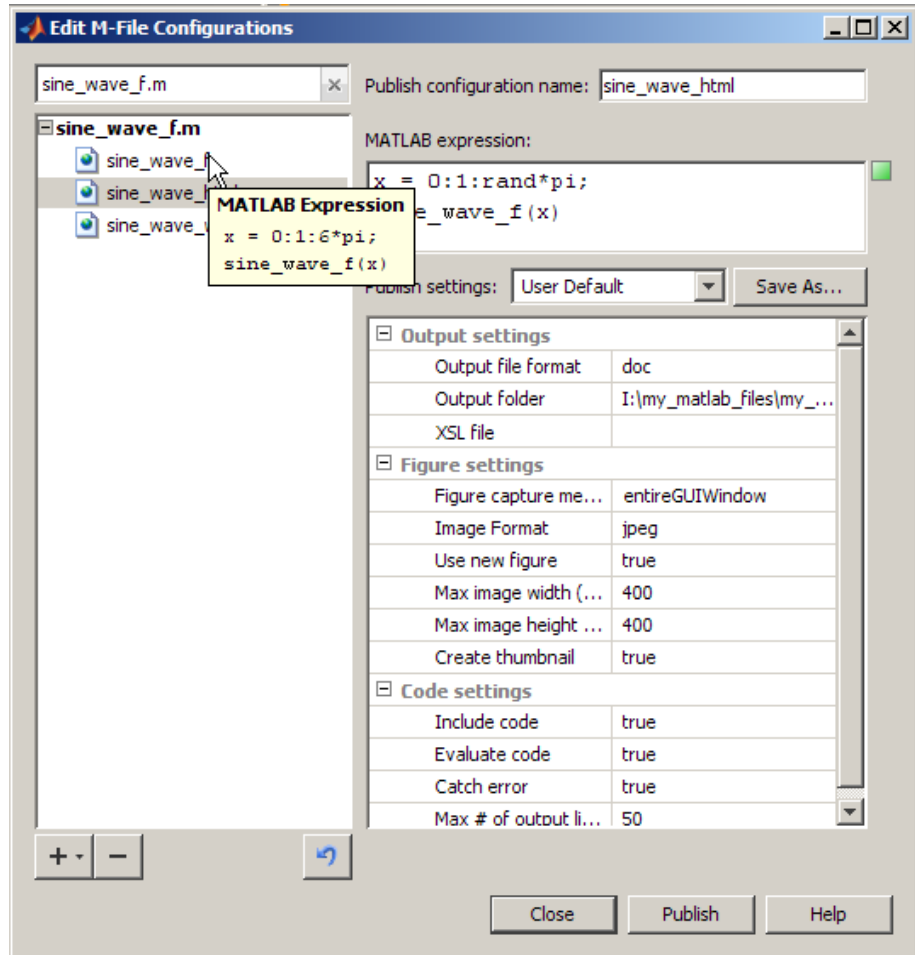
5 From the **Publish settings** drop-down list, select word_settings.

This example uses the word_settings configuration as a starting point for adjusting the publish settings. In step 8, it will be saved using a different **Publish settings** name.

6 Change the **Output file format** to **html**.
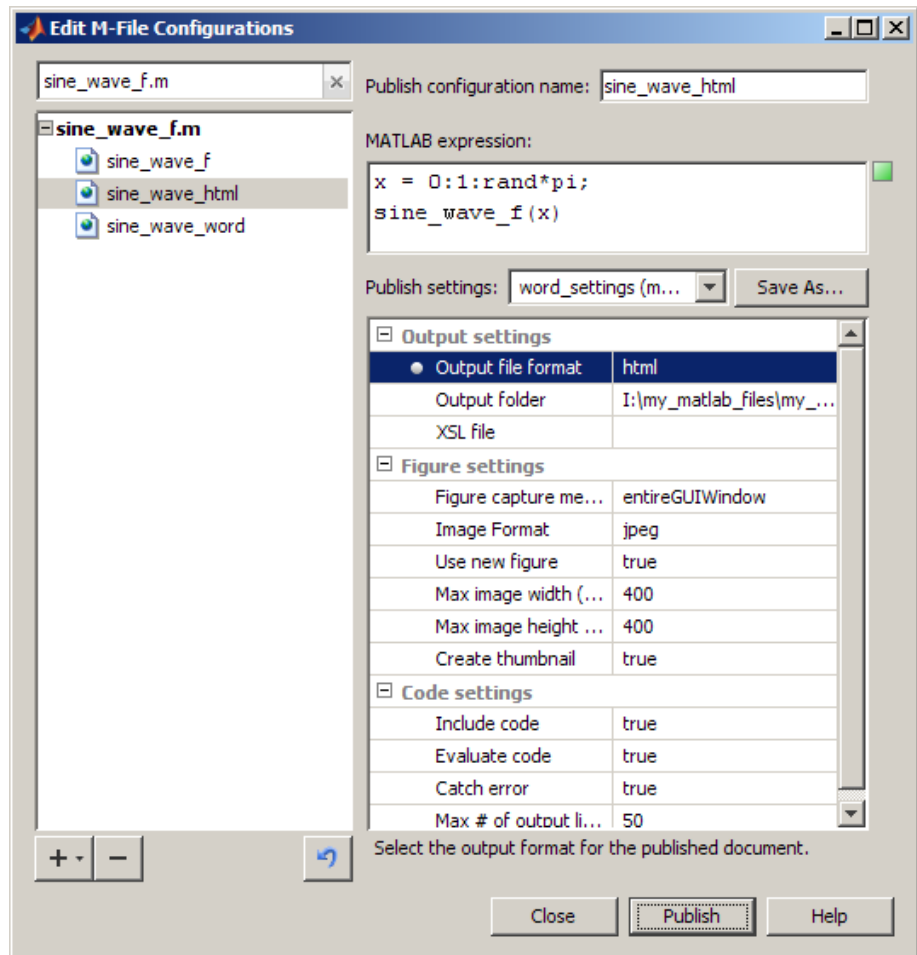
7 Click **Publish** to test how the HTML output appears.

8 Click **Save As**. In the Save Publish Settings dialog box, in the **Settings name** field, type `html_settings`, and then click **Save**.

Steps for Publishing sine_wave_f.m to Microsoft PowerPoint

These steps provide an example of creating a configuration for `sine_wave_f.m`, that publishes the M-file to Microsoft PowerPoint. You might do this to publish output for presentation in a meeting, for example.

1 If it is not currently open, open the Edit M-File Configurations dialog box.

2 Select **sine_wave_word** in the list of M-files and configurations on the left side of the dialog box, click the **+** button, and then select **Publish Configurations**.

3 In the **Publish configuration name** field, replace `sine_wave_f_2` with `sine_wave_ppt`.

4 In the **MATLAB expression** field, replace the default expression with the following:

```
x = 0:1:6*pi;
sine_wave_f(x)
```

5 From the **Publish settings** drop-down list, select `word_settings`.

This example uses the `word_settings` configuration as a starting point for adjusting the publish settings. In step 8, it will be saved using a different **Publish settings** name.

6 Assume for the purposes of a PowerPoint presentation, you do not want to include the code.

Change the **Output file format** to **ppt** and **Include code** to **false**.
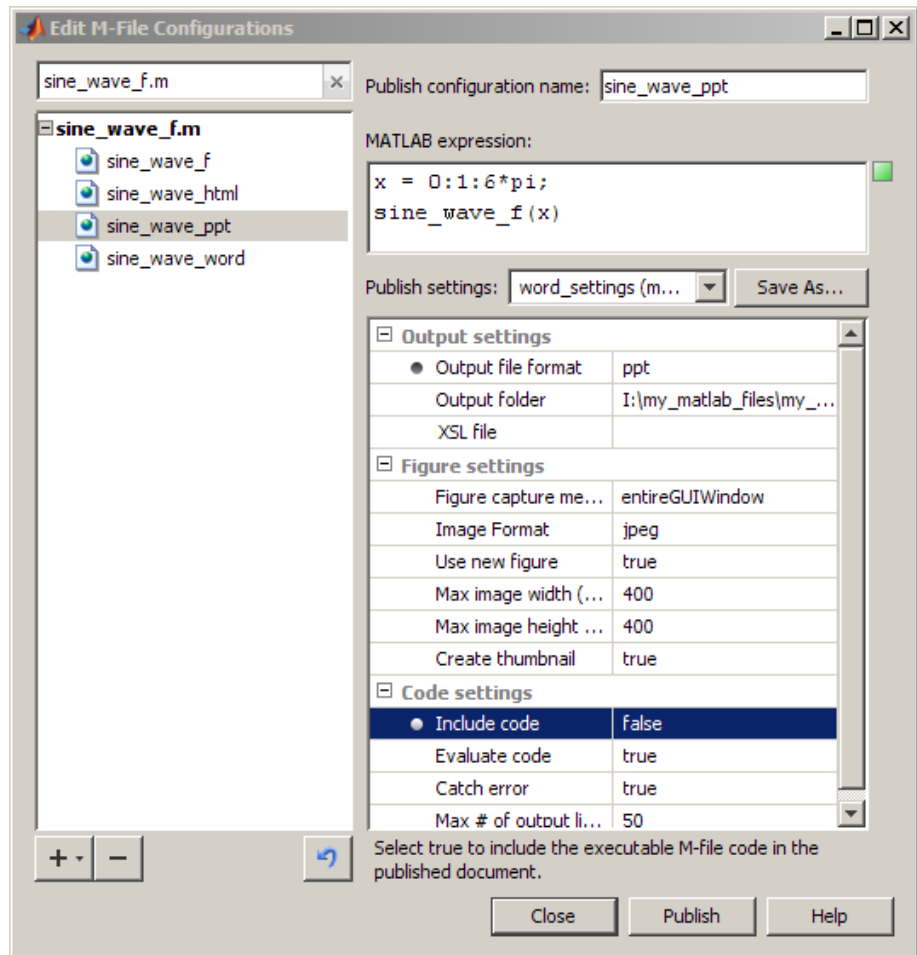
7 Click **Publish** to test how the PowerPoint output appears.

8 Click **Save As**. In the Save Publish Settings dialog box, in the **Settings name** field, type ppt_settings, and then click **Save**.
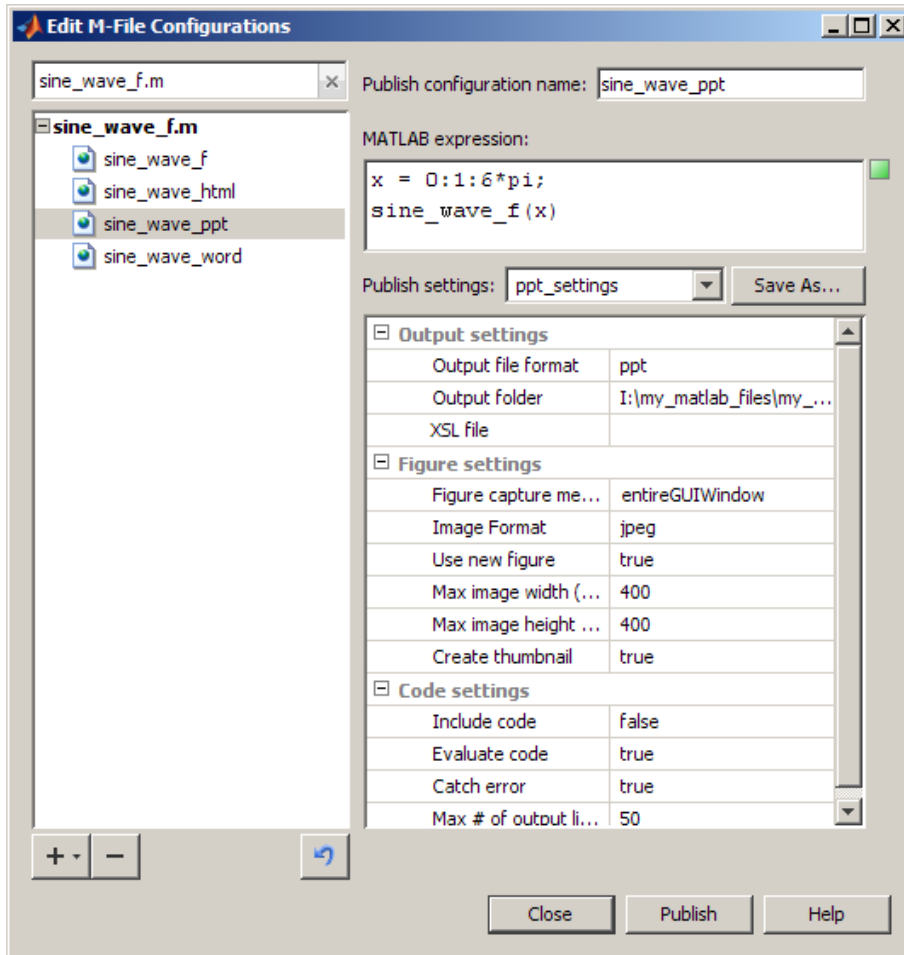
About the publish_configurations.m File

When you create one or more publish configurations using the Edit M-File Configurations dialog box, the Editor updates the publish_configurations.m file in your preferences directory. (This is the directory that MATLAB returns when you run the MATLAB prefdir function.)

Although you can port this file from the preferences directory on one system to another, there can only be one publish_configurations.m file on a

system. Therefore, you should only do this if you have not already created configurations on the second system. In addition, because this file may contain references to file paths, you need to be sure the specified M-files and paths exist on the second system.

The MathWorks recommends that you not update `publish_configurations.m` in the MATLAB Editor or a text editor. Changes that you make using tools other than the Edit M-File Configurations dialog box may be overwritten later. Each time you save a configuration using the Edit M-File Configurations dialog box, MATLAB updates the `publish_configurations.m` file, as well as the `run_configurations.m` file. For more information, see "About the run_configurations.m File" on page 6-112.

Find Publish Configurations

The method you use to find publish configurations is the same as the one you use to find run configurations. For details, see "Find Configurations" on page 6-112.

Remove Publish Configurations

If you no longer need a publish configuration because you do not use it or because you deleted the M-file with which it is associated, it is a good practice to delete the publish configuration. The method you use to delete publish configurations is the same as the one you use to delete run configurations. For details, see "Remove Configurations" on page 6-114 for details.

Reassociate and Rename Publish Configurations

Each publish configuration is associated with a specific M-file. If you move or rename the M-file, you need to redefine the association. If you delete an M-file, you might want to delete the associated configurations, or associate them with a different M-file. You might also need to modify the statements in the configurations so they will run. The method you use to reassociate and rename publish configurations is the same as the one you use to reassociate and rename run configurations. See "Reassociate and Rename Configurations" on page 6-115 for details.

9

Using Notebook to Publish to Microsoft Word

Notebook is useful for creating electronic or printed records of MATLAB sessions, class notes, textbooks or technical reports to Microsoft Word. As an alternative to Notebook, consider using cells to publish to Microsoft Word. For more information, see Chapter 8, "Publishing M-Files".

Note Notebook is available only on Windows systems that have Microsoft Word installed. For supported versions of Word, see "Configuring Notebook" on page 9-27.

About Using Notebook to Publish to Word

In this section...
"Using Notebook to Create an M-book" on page 9-2
"Creating or Opening an M-Book" on page 9-2
"Entering MATLAB Commands in an M-Book" on page 9-9
"Protecting the Integrity of Your Workspace in M-Books" on page 9-9
"Ensuring Data Consistency in M-Books" on page 9-10
"Debugging and Notebook" on page 9-10

Using Notebook to Create an M-book

Using Notebook, you can create a document, called an *M-book*, that contains text, MATLAB commands, and the output from MATLAB commands.

You can think of an M-book as a record of an interactive MATLAB session annotated with text, or as a document embedded with live MATLAB commands and output.

Creating or Opening an M-Book

This section includes information on performing the following tasks:

- "Creating an M-Book from the MATLAB Desktop" on page 9-2
- "Creating an M-Book While Running Notebook" on page 9-5
- "Opening an Existing M-Book" on page 9-6
- "Converting a Word Document to an M-Book" on page 9-7

Creating an M-Book from the MATLAB Desktop

To create a new M-book from within MATLAB desktop, type the following in the Command Window:

```
notebook
```

If you are running Notebook for the first time, you might need to configure it. See "Configuring Notebook" on page 9-27 for more information.

Notebook starts Microsoft Word on your system and creates a new M-book, called `Document1`.

When Word is opening, if a dialog box appears asking you to enable or disable macros, choose to enable macros. Notebook defines Microsoft Word macros that enable MATLAB to interpret the different types of cells that hold MATLAB commands and their output. For more information on macro security, see "Configuring Notebook" on page 9-27.

Depending on the version of Word you are using, one of the following occurs:

- In Word 2002, and 2003, Notebook adds the **Notebook** menu to the Word menu bar, as shown in the following illustration. Use this menu to access Notebook features.
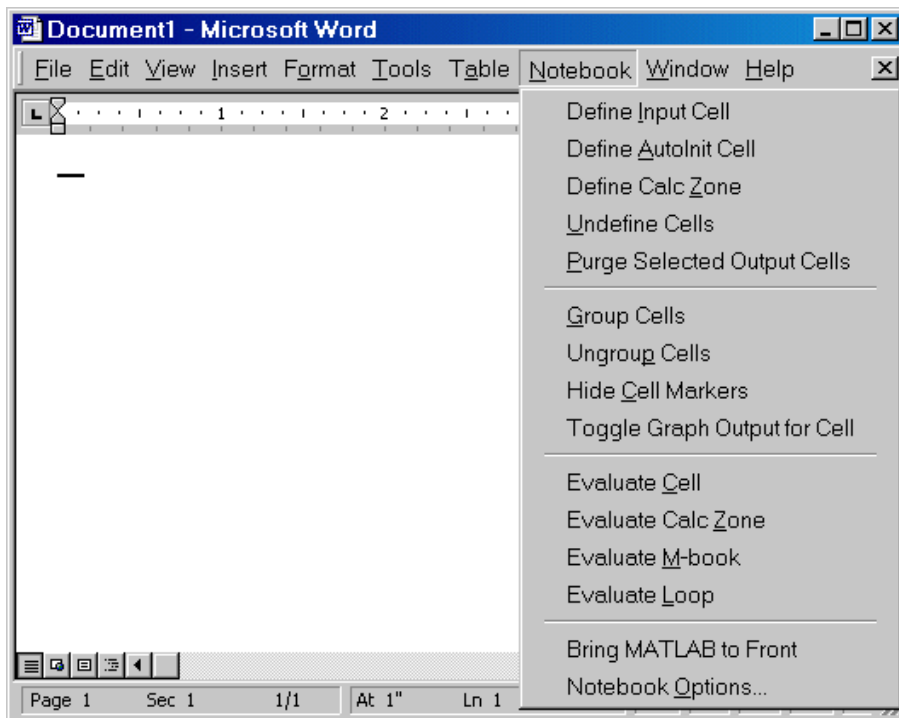
- In Word 2007, Notebook adds the **Notebook** menu to the Word **Add-Ins** tab, as shown in the following illustration. Use this menu to access Notebook features.
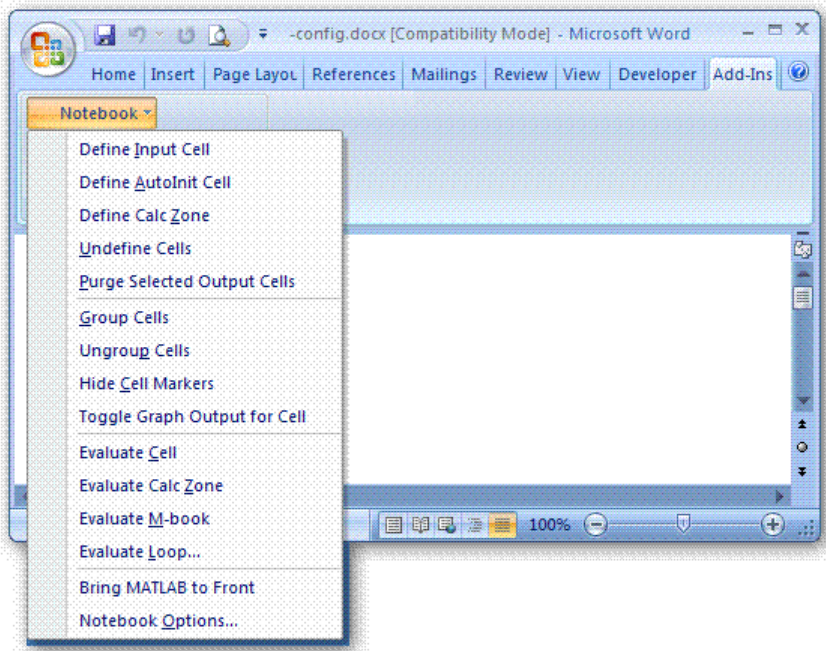
Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Creating an M-Book While Running Notebook

With Notebook running, you can create a new M-book as follows:

- In Word 2002, and 2003, select **File > New M-book**

- In Word 2007, select **Add-Ins > New M-book**, as shown in the following figure.

New M-book

Add-Ins tab
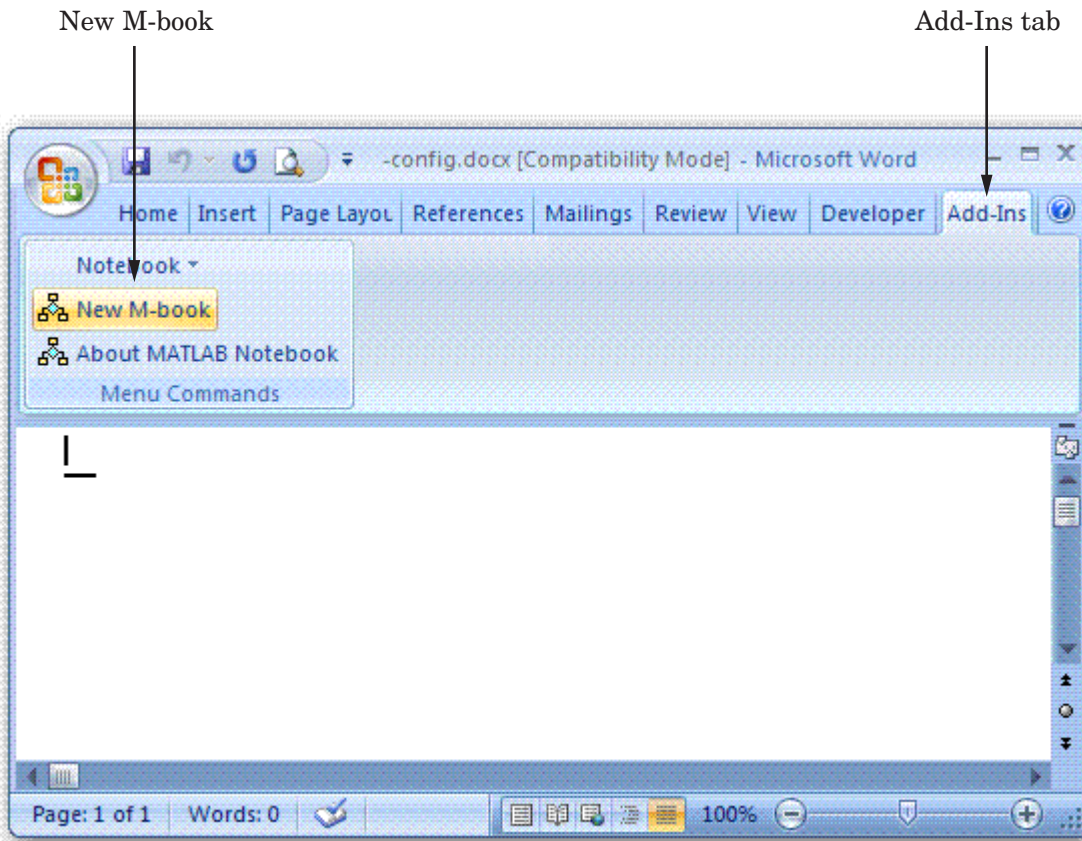
Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Opening an Existing M-Book

You can use the notebook command to open an existing M-book, as shown in the following code, where *filename* is the M-book you want to open.

```
notebook filename
```

Alternatively, you can double-click an M-book file in a Windows file management tool, such as Explorer.

When you double-click an M-book, Microsoft Word opens the M-book and starts MATLAB if it is not already running. Notebook adds the **Notebook** menu to the Word menu bar and adds **New M-book** to the **File** menu, as shown in the figure that follows.
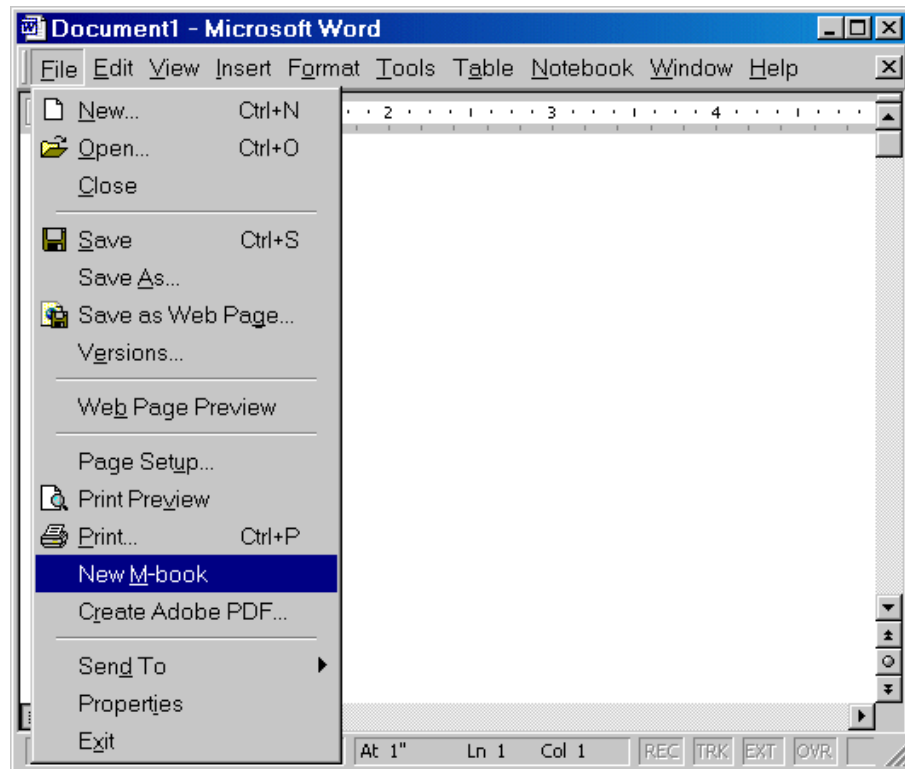
Converting a Word Document to an M-Book

To convert a Word document to an M-book, follow the steps provided in one of the following sections, depending on which version of Word you are using:

- "Microsoft Word 2002, or 2003" on page 9-8

- "Microsoft Word 2007" on page 9-8

Microsoft Word 2002, or 2003.

1 Create a new M-book.

2 From the **Insert** menu, select **File**.

3 Select the file you want to convert.

4 Click **OK**.

Microsoft Word 2007.

1 Create a new M-book.

2 From the **Insert** tab, in the **Text** group, click the arrow next to **Object** and then click **Text from File**, as shown in the image that follows.

  The Insert File dialog box opens.

3 In the Insert File dialog box, select the file that you want to convert, and then click **OK**.
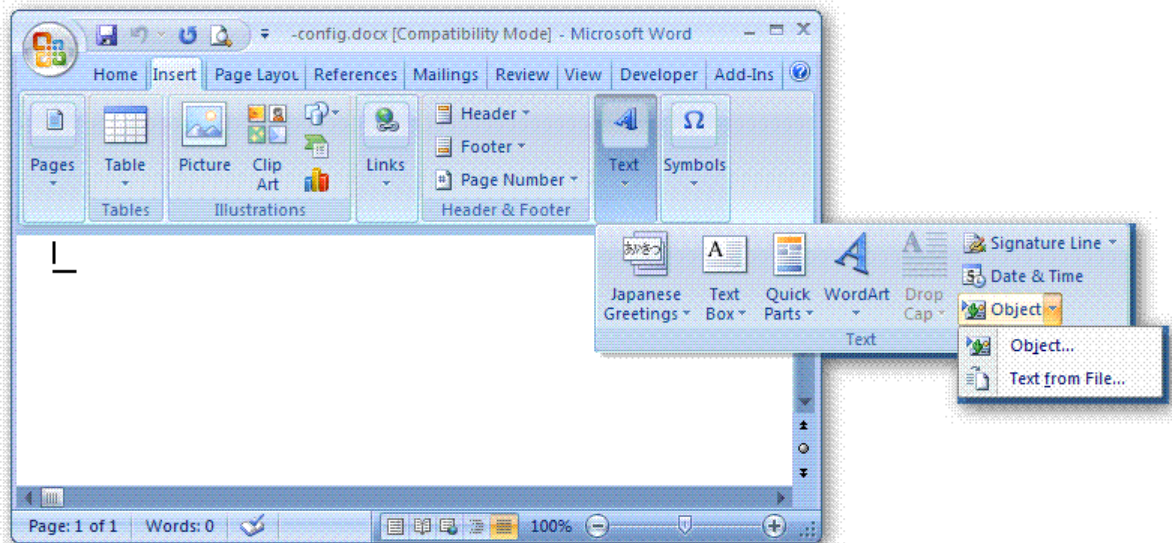
Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Entering MATLAB Commands in an M-Book

Note A good way to learn how to use Notebook is to open the sample M-book, `Readme.doc`, and try out the various techniques described in this section. You can find this file in the *matlabroot*/`notebook`/`pc` directory.

You enter MATLAB commands in an M-book the same way you enter text in any other Word document. For example, you can enter the following text in a Word document. The example uses text in Courier Font but you can use any font:

```
Here is a sample M-book.

a = magic(3)
```

To execute the MATLAB `magic` command in this document, you must follow the steps described in these sections:

- "Defining MATLAB Commands as Input Cells for Notebook" on page 9-11
- "Evaluating MATLAB Commands with Notebook" on page 9-16

MATLAB displays the output of the command in the Word document in an output cell.

Protecting the Integrity of Your Workspace in M-Books

When you work on more than one M-book in a single word processing session, note that:

- Each M-book uses the same "copy" of MATLAB.
- All M-books share the same workspace.

If you use the same variable names in more than one M-book, data used in one M-book can be affected by another M-book. You can protect the integrity of your workspace by specifying the `clear` command as the first autoinit cell in the M-book.

Ensuring Data Consistency in M-Books

An M-book can be thought of as a sequential record of a MATLAB session. When executed in order, from the first MATLAB command to the last, the M-book accurately reflects the relationships among these commands.

If, however, you change an input cell or output cell as you refine your M-book, Notebook does not automatically recalculate input cells that depend on either the contents or the results of the changed cells. As a result, the M-book may contain inconsistent data.

When working on an M-book, you might find it useful to select **Evaluate M-book** periodically to ensure that your M-book data is consistent. You can also use calc zones to isolate related commands in a section of the M-book, and then use **Evaluate Calc Zone** to execute only those input cells contained in the calc zone.

Debugging and Notebook

Do not use debugging functions or the Editor while evaluating cells with Notebook. Instead, debug M-files from within MATLAB, and then after completing debugging, clear all the breakpoints and access the M-file using Notebook. If you debug while evaluating from Notebook, you might experience problems with MATLAB.

Defining MATLAB Commands as Input Cells for Notebook

In this section...
"Defining Commands as Input Cells for Notebook" on page 9-11
"Defining Cell Groups for Notebook" on page 9-12
"Defining Autoinit Input Cells for Notebook" on page 9-13
"Defining Calc Zones for Notebook" on page 9-13
"Converting an Input Cell to Text with Notebook" on page 9-14

For information about evaluating the input cells you define, see "Evaluating MATLAB Commands with Notebook" on page 9-16.

Defining Commands as Input Cells for Notebook

To define a MATLAB command in a Word document as an input cell, follow these steps:

1 Type the command into the M-book as text. For example,

```
This is a sample M-book.

a = magic(3)
```

2 Position the cursor anywhere in the command and select
Notebook > Define Input Cell or press **Alt+D**. If the command is
embedded in a line of text, use the mouse to select it. Notebook defines the
MATLAB command as an input cell:

```
This is a sample M-book.

[a = magic(3)]
```

Note how Notebook changes the character font of the text in the input cell to a
bold, dark green color and encloses it within *cell markers*. Cell markers are
bold, gray brackets. They differ from the brackets used to enclose matrices by
their size and weight. For information about changing these default formats,
see "Modifying Styles in the M-Book Template" on page 9-22.

Defining Cell Groups for Notebook

You can collect several input cells into a single input cell. This is called a *cell group*. Because all the output from a cell group appears in a single output cell that Notebook places immediately after the group, cell groups are useful when several MATLAB commands are needed, such as to fully define a graphic.

For example, if you define all the MATLAB commands that produce a graphic as a cell group and then evaluate that cell group, Notebook generates a single graphic that includes all the graphic components defined in the commands. If instead you define all the MATLAB commands that generate the graphic as separate input cells, evaluating the cells generates multiple graphic output cells.

See "Evaluating Cell Groups with Notebook" on page 9-17 for information about evaluating a cell group. For information about ungrouping a cell group, see "Ungroup Cells" on page 9-36.

Creating a Cell Group for Notebook

To create a cell group, follow these steps:

1 Use the mouse to select the input cells that are to make up the group.

2 Select **Notebook > Group Cells** or press **Alt+G**.

Notebook converts the selected cells into a cell group and replaces cell markers with a single pair that surrounds the group:

```
This is a sample cell group.

[date
a = magic(3) ]
```

Note the following:

- A cell group cannot contain output cells. If the selection includes output cells, Notebook deletes them.

- A cell group cannot contain text. If the selection includes text, Notebook places the text after the cell group. However, if the text precedes the first input cell in the selection, Notebook leaves it where it is.

- If you select part or all of an output cell, but not its input cell, Notebook includes the input cell in the cell group.

When you create a cell group, Notebook defines it as an input cell unless its first line is an autoinit cell, in which case Notebook defines the group as an autoinit cell.

Defining Autoinit Input Cells for Notebook

You can use *autoinit cells* to specify MATLAB commands to be automatically evaluated each time an M-book is opened. This is a quick and easy way to initialize the workspace. *Autoinit cells* are input cells with the following additional characteristics:

- Notebook evaluates the autoinit cells when it opens the M-book.
- Notebook displays the commands in autoinit cells using dark blue characters.

Autoinit cells are otherwise identical to input cells.

Creating an Autoinit Cell for Notebook

You can create an autoinit cell in one of the following two ways:

- Enter the MATLAB command as text, then convert the command to an autoinit cell by selecting **Notebook > Define AutoInit Cell**.
- If you already entered the MATLAB command as an input cell, you can convert the input cell to an autoinit cell. Either select the input cell or position the cursor in the cell, then select **Notebook > Define AutoInit Cell**.

See "Evaluating MATLAB Commands with Notebook" on page 9-16 for information about evaluating autoinit cells.

Defining Calc Zones for Notebook

You can partition an M-book into self-contained sections, called *calc zones*. A calc zone is a contiguous block of text, input cells, and output cells. Notebook inserts Microsoft Word section breaks before and after the section to define

the calc zone. The section break indicators include bold, gray brackets to distinguish them from standard Word section breaks.

You can use calc zones to prepare problem sets, making each problem a separate calc zone that can be created and tested on its own. An M-book can contain any number of calc zones.

Note Using calc zones does not affect the scope of the variables in an M-book. Variables used in one calc zone are accessible to all calc zones.

Creating a Calc Zone
After you create the text and cells that you want to include in the calc zone, define the calc zone by following these steps:

1 Select the input cells and text to be included in the calc zone.

2 Select **Notebook > Define Calc Zone**.

Note You must select an input cell and its output cell in their entirety to include them in the calc zone.

See "Evaluating a Calc Zone with Notebook" on page 9-19 for information about evaluating a calc zone.

Converting an Input Cell to Text with Notebook
To convert an input cell (or an autoinit cell or a cell group) to text, follow these steps:

1 Select the input cell with the mouse or position the cursor in the input cell.

2 Select **Notebook > Undefine Cells** or press **Alt+U**.

When Notebook converts the cell to text, it reformats the cell contents according to the Microsoft Word Normal style. For more information about M-book styles, see "Modifying Styles in the M-Book Template" on page

9-22. When you convert an input cell to text, Notebook also converts the corresponding output cell to text.

Evaluating MATLAB Commands with Notebook

In this section...
"Evaluating Input Commands with Notebook" on page 9-16
"Evaluating Cell Groups with Notebook" on page 9-17
"Evaluating a Range of Input Cells with Notebook" on page 9-18
"Evaluating a Calc Zone with Notebook" on page 9-19
"Evaluating an Entire M-Book" on page 9-19
"Using a Loop to Evaluate Input Cells Repeatedly with Notebook" on page 9-20
"Converting Output Cells to Text with Notebook" on page 9-21
"Deleting Output Cells with Notebook" on page 9-21

Evaluating Input Commands with Notebook

After you define a MATLAB command as an input cell, or as an autoinit cell, you can evaluate it in your M-book. Use the following steps to define and evaluate a MATLAB command:

1 Type the command into the M-book as text. For example:

```
This is a sample M-book

a = magic(3)
```

2 Position the cursor anywhere in the command. If the command is embedded in a line of text, use the mouse to select it. Then select **Notebook > Define Input Cell** or press **Alt+D**.

Notebook defines the MATLAB command as an input cell. For example:

```
This is a sample M-book

[a = magic(3)]
```

3 Specify the input cell to be evaluated by selecting it with the mouse or by placing the cursor in it. Then select **Notebook > Evaluate Cell** or press **Ctrl+Enter**.

Notebook evaluates the input cell and displays the results in a output cell immediately following the input cell. If there is already an output cell, Notebook replaces its contents, wherever it is in the M-book. For example:

```
This is a sample M-book.

[a = magic(3) ]

[a =
    8    1    6
    3    5    7
    4    9    2 ]
```

The text in the output cell is blue and is enclosed within cell markers. Cell markers are bold, gray brackets. They differ from the brackets used to enclose matrices by their size and weight. Error messages appear in red. For information about changing these default formats, see "Modifying Styles in the M-Book Template" on page 9-22.

Evaluating Cell Groups with Notebook

You evaluate a cell group the same way you evaluate an input cell (because a cell group is an input cell), as follows:

1 Position the cursor anywhere in the cell or in its output cell.

2 Select **Notebook > Evaluate Cell** or press **Ctrl+Enter**.

For information about creating a cell group, see "Defining Cell Groups for Notebook" on page 9-12.

When MATLAB evaluates a cell group, the output for all commands in the group appears in a single output cell. By default, Notebook places the output cell immediately after the cell group the first time the cell group is evaluated. If you evaluate a cell group with an existing output cell, Notebook places the results in the output cell wherever the output cell is located in the M-book.

Note Text or numeric output always comes first, regardless of the order of the commands in the group.

The following illustration shows a cell group and the figure created when you evaluate the cell group.

This is a sample M-book with a cell group.

```
[t = 0:pi/100:2*pi;
y = sin(t);
plot(t,y) ]
```
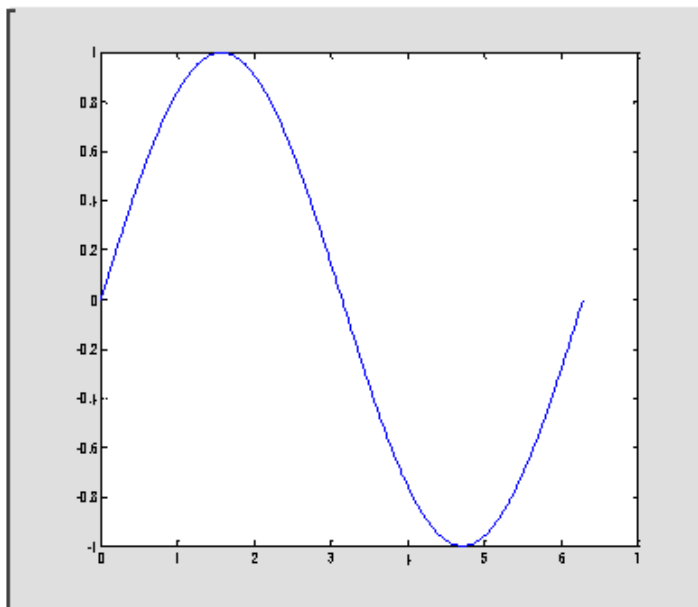

Evaluating a Range of Input Cells with Notebook

To evaluate more than one MATLAB command contained in different but contiguous input cells, follow these steps:

1 Select the range of cells that includes the input cells you want to evaluate. You can include text that surrounds input cells in your selection.

2 Select **Notebook > Evaluate Cell** or press **Ctrl+Enter**.

Notebook evaluates each input cell in the selection, inserting new output cells or replacing existing ones.

Evaluating a Calc Zone with Notebook

To evaluate a calc zone, follow these steps:

1 Position the cursor anywhere in the calc zone.

2 Select **Notebook > Evaluate Calc Zone** or press **Alt+Enter**.

For information about creating a calc zone, see "Defining Calc Zones for Notebook" on page 9-13.

By default, Notebook places the output cell immediately after the calc zone the first time the calc zone is evaluated. If you evaluate a calc zone with an existing output cell, Notebook places the results in the output cell wherever it is located in the M-book.

Evaluating an Entire M-Book

To evaluate the entire M-book, either select **Notebook > Evaluate M-book** or press **Alt+R**.

Notebook begins at the top of the M-book regardless of the cursor position and evaluates each input cell in the M-book. As it evaluates the M-book, Notebook inserts new output cells or replaces existing output cells.

Controlling Execution of Multiple Commands

When you evaluate an entire M-book, and an error occurs, evaluation continues. If you want to stop evaluation if an error occurs, follow this procedure:

1 Select **Notebook > Notebook Options**.

The **Notebook Options** dialog box opens.

2 Select the **Stop evaluating on error** check box and click **OK**.

Using a Loop to Evaluate Input Cells Repeatedly with Notebook

To evaluate a sequence of MATLAB commands repeatedly, follow these steps:

1 Use the mouse to select the input cells, including any text or output cells located between them.

2 Select **Notebook > Evaluate Loop** or press **Alt+L**. Notebook displays the **Evaluate Loop** dialog box.
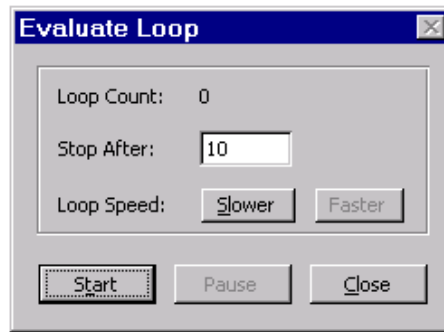
3 Enter the number of times you want MATLAB to evaluate the selected commands in the **Stop After** field, then click **Start**. The button changes to **Stop**. Notebook begins evaluating the commands and indicates the number of completed iterations in the **Loop Count** field.

You can increase or decrease the delay at the end of each iteration by clicking **Slower** or **Faster**. Slower increases the delay. Faster decreases the delay.

To suspend evaluation of the commands, click **Pause**. The button changes to **Resume**. Click **Resume** to continue evaluation.

To stop processing the commands, click **Stop**. To close the **Evaluate Loop** dialog box, click **Close**.

Converting Output Cells to Text with Notebook

You can convert an output cell to text by undefining cells. If the output is numeric or textual, Notebook removes the cell markers and converts the cell contents to text according to the Microsoft Word Normal style. If the output is graphical, Notebook removes the cell markers and dissociates the graphic from its input cell, but does not alter its contents.

Note Undefining an output cell does not affect the associated input cell.

To undefine an output cell, follow these steps:

1 Select the output cell you want to undefine.

2 Select **Notebook > Undefine Cells** or press **Alt+U**.

Deleting Output Cells with Notebook

To delete output cells, follow these steps:

1 Select an output cell, using the mouse, or place the cursor in the output cell.

2 Select **Notebook > Purge Selected Output Cells** or press **Alt+P**.

If you select a range of cells, Notebook deletes all the output cells in the selected range, but any associate input cells remain intact.

Printing and Formatting an M-Book

In this section...
"Printing an M-Book" on page 9-22
"Modifying Styles in the M-Book Template" on page 9-22
"Choosing Loose or Compact Format for Notebook" on page 9-23
"Controlling Numeric Output Format for Notebook" on page 9-24
"Controlling Graphic Output for Notebook" on page 9-24

Printing an M-Book

You can print all or part of an M-book by doing one of the following, depending on the version of Microsoft Word you are using:

- In Microsoft Word 2002, 2003 — Select **File > Print**.

- In Microsoft Word 2007 — Select **Microsoft Office Button > Print**

Word follows these rules when printing M-book cells and graphics:

- Cell markers are not printed.

- Input cells, autoinit cells, and output cells (including error messages) are printed according to their defined styles. If you prefer to print these cells using black type instead of colors or shades of gray, you can modify the styles.

Modifying Styles in the M-Book Template

You can control the appearance of the text in your M-book by modifying the predefined styles stored in the M-book template, `m-book.dot`. These styles control the appearance of text and cells. By default, M-books use the Word Normal style for all other text.

For example, if you print an M-book on a color printer, input cells appear dark green, output and autoinit cells appear dark blue, and error messages appear red. If you print the M-book on a grayscale printer, these cells appear as

shades of gray. To print these cells using black type, you need to modify the color of the Input, Output, AutoInit, and Error styles in the M-book template.

The table below describes the default styles used by Notebook. If you modify styles, you can use the information in the tables below to help you return the styles to their original settings. For general information about using styles in Word documents, see the Word documentation.

Style	Font	Size	Weight	Color
Normal	Times New Roman	10 points	N/A	Black
AutoInit	Courier New	10 points	Bold	Dark blue
Error	Courier New	10 points	Bold	Red
Input	Courier New	10 points	Bold	Dark green
Output	Courier New	10 points	N/A	Blue

When you change a style, Word applies the change to all characters in the M-book that use that style and gives you the option to change the template. Be cautious about making changes to the template. If you choose to apply the changes to the template, you will affect all new M-books that you create using the template. See the Word documentation for more information.

Choosing Loose or Compact Format for Notebook

You can specify whether a blank line appears between the input and output cells by selecting the loose or compact format, as follows:

1 Select **Notebook > Notebook Options**.

2 In the **Notebook Options** dialog box, select either **Loose** or **Compact**. Loose format adds an empty line. Compact format does not.

3 Click **OK**.

Note Changes you make using the **Notebook Options** dialog box take effect for output generated *after* you click **OK**. To affect existing input or output cells, you must reevaluate the cells.

Controlling Numeric Output Format for Notebook

To change how Notebook displays numeric output, follow these steps:

1 Select **Notebook > Notebook Options**.

2 In the **Notebook Options** dialog box, select a format from the **Numeric Format** list. These settings correspond to the choices available with the MATLAB `format` command.

3 Click **OK**.

Note Changes you make using the **Notebook Options** dialog box take effect for output generated *after* you click **OK**. To affect existing input or output cells, you must reevaluate the cells.

Controlling Graphic Output for Notebook

This section describes how to control several aspects of the graphic output produced by MATLAB commands in an M-book, including

- "Embedding Graphic Output in the M-Book" on page 9-24
- "Suppressing Graphic Output for Individual Input Cells in Notebook" on page 9-25
- "Adjusting Graphic Output in Notebook" on page 9-26

Embedding Graphic Output in the M-Book

By default, graphic output is embedded in an M-book. To display graphic output in a separate figure window, follow these steps:

1 Select **Notebook > Notebook Options**.

2 In the **Notebook Options** dialog box, clear the **Embed Figures in M-book** check box.
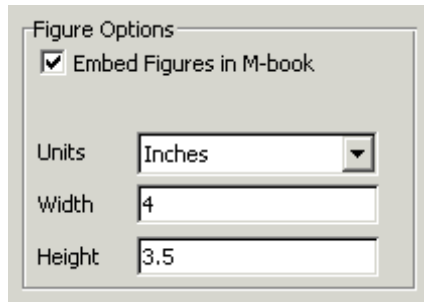
3 Click **OK**.

Note Embedded figures do not include Handle Graphics objects generated by the `uicontrol` and `uimenu` functions.

Notebook determines whether to embed a figure in the M-book by examining the value of the figure object's `Visible` property. If the value of the property is `off`, Notebook embeds the figure. If the value of this property is `on`, Notebook directs all graphic output to the current figure window.

Suppressing Graphic Output for Individual Input Cells in Notebook

If an input or autoinit cell generates figure output that you want to suppress, follow these steps:

1 Place the cursor in the input cell.

2 Select **Notebook > Toggle Graph Output for Cell**.

Notebook suppresses graphic output from the cell, inserting the string (`no graph`) after the input cell.

To allow graphic output for a cell, repeat the procedure. Notebook removes the (`no graph`) marker and allows graphic output from the cell.

Note **Toggle Graph Output for Cell** overrides the **Embed Figures in M-book** option, if that option is set.

Adjusting Graphic Output in Notebook

To set the default size of embedded graphics in an M-book, follow these steps:

1 Select **Notebook > Notebook Options**.

2 In the **Notebook Options** dialog box, use the **Units**, **Width** and **Height** fields to set the size of graphics generated by the M-book.

3 Click **OK**.

Note Changes you make using the **Notebook Options** dialog box take effect for graphic output generated *after* you click **OK**. To affect existing input or output cells, you must reevaluate the cells.

You change the size of an existing embedded figure by selecting the figure, clicking the left mouse button anywhere in the figure, and dragging the resize handles of the figure. If you resize an embedded figure using its resize handles and then regenerate the figure, its size reverts to its original size.

To crop graphic output, or add white space around it, follow the instructions for performing these tasks in Microsoft Word. See the Microsoft Word help for details.

Configuring Notebook

After you install Notebook, but before you begin using it, you must specify that Word can use the Notebook macros, and then configure Notebook. (Notebook is installed as part of the MATLAB installation process on Microsoft Windows platforms. For more information, see the MATLAB installation documentation for your platform.)

To specify that Word can use the Notebook macros:

- In Word 2002, and 2003 do either of the following:
  - Set the macro security level to medium: in Word, select **Tools > Macros > Security**, and in the resulting dialog box, choose **Medium**.
  - After starting Notebook, when Word first opens, a security warning dialog box appears. In the dialog box, select **Always trust macros from this source**. This allows you to use Notebook, but still maintain a high security level for other macros you use in Word.
- In Word 2007, follow the Word help instructions in the topic entitled "Enable or disable macros in Office documents."

To configure Notebook:

1 Type `notebook` in the MATLAB Command Window.

  MATLAB opens a dialog box that indicates Notebook has not been configured and asks if you want to configure it now.

2 Click **Yes**.

  MATLAB configures Notebook and issues the following messages in the Command Window:

```
Welcome to the utility for setting up the MATLAB Notebook
for interfacing MATLAB to Microsoft Word

Setup complete
Warning: MATLAB is now an automation server
```

When MATLAB configures Notebook, it accesses the Microsoft Windows system registry to locate Microsoft Word and the Word templates directory, and to identify the version of Word. MATLAB then copies Notebook's `m-book.dot` template to the Word templates directory. MATLAB Notebook supports Word versions 2002, 2003, and 2007.

If you have previously configured Notebook, typing `notebook` in the MATLAB Command Window, starts Microsoft Word and creates a new M-book. The Command Window displays the message `Warning:  MATLAB is now an automation server`.

If you suspect a problem with the current configuration, you can explicitly configure notebook by typing:

```
notebook ('-setup')
```

Notebook Feature Reference

In this section...

This section provides reference information about each of the Notebook features, listed alphabetically. To use these features, select them from the **Notebook** menu in Microsoft Word. (In Word 2007, the **Notebook** menu is on the **Add-Ins** tab.)

Bring MATLAB to Front

Bring MATLAB to Front brings the MATLAB Command Window to the foreground.

Define Autoinit Cell

Define AutoInit Cell creates an autoinit cell by converting the current paragraph, selected text, or input cell. An autoinit cell is an input cell that is automatically evaluated whenever you open an M-book.

Result

If you select this feature while the cursor is in a paragraph of text, Notebook converts the entire paragraph to an autoinit cell. If you select this feature while text is selected, Notebook converts the text to an autoinit cell. If you select this feature while the cursor is in an input cell, Notebook converts the input cell to an autoinit cell.

Format

Notebook formats the autoinit cell using the AutoInit style, defined as bold, dark blue, 10-point Courier New.

See Also

For more information about autoinit cells, see "Defining Autoinit Input Cells for Notebook" on page 9-13.

Define Calc Zone

Define Calc Zone defines the selected text, input cells, and output cells as a calc zone. A calc zone is a contiguous block of related text, input cells, and output cells that describes a specific operation or problem.

Result

Notebook defines a calc zone as a Word document section, placing section breaks before and after the calc zone. However, Word does not display section breaks at the beginning or end of a document.

See Also

For information about evaluating calc zones, see "Evaluating a Calc Zone with Notebook" on page 9-19. For more information about document sections, see the Microsoft Word documentation.

Define Input Cell

Define Input Cell creates an input cell by converting the current paragraph, selected text, or autoinit cell. An input cell contains a MATLAB command.

Result

If you select this feature while the cursor is in a paragraph of text, Notebook converts the entire paragraph to an input cell. If you select this feature while text is selected, Notebook converts the text to an input cell. If you select this feature while the cursor is in an autoinit cell, Notebook converts the autoinit cell to an input cell.

Format

Notebook encloses the text in cell markers and formats the cell using the Input style, defined as bold, dark green, 10-point Courier New.

See Also

For more information about creating input cells, see "Defining MATLAB Commands as Input Cells for Notebook" on page 9-11. For information about evaluating input cells, see "Evaluating MATLAB Commands with Notebook" on page 9-16.

Evaluate Calc Zone

Evaluate Calc Zone sends the input cells in the current calc zone to MATLAB to be evaluated. The current calc zone is the Word section that contains the cursor.

Result

As Notebook evaluates each input cell, it generates an output cell. When you evaluate an input cell for which there is no output cell, Notebook places the output cell immediately after the input cell that generated it. If you evaluate an input cell for which there is an output cell, Notebook replaces the results in the output cell wherever it is in the M-book.

See Also

For more information, see "Evaluating a Calc Zone with Notebook" on page 9-19.

Evaluate Cell

Evaluate Cell sends the current input cell or cell group to MATLAB to be evaluated. An input cell contains a MATLAB command. A cell group is a single, multiline input cell that contains more than one MATLAB command. Notebook displays the output or an error message in an output cell.

Result

If you evaluate an input cell for which there is no output cell, Notebook places the output cell immediately after the input cell that generated it. If you evaluate an input cell for which there is an output cell, Notebook replaces the results in the output cell wherever it is in the M-book. If you evaluate a cell group, all output for the cell appears in a single output cell.

An input cell or cell group is the current input cell or cell group if

- The cursor is in the input cell or cell group.

- The cursor is at the end of the line that contains the closing cell marker for the input cell or cell group.

- The cursor is in the output cell for the input cell or cell group.

- The input cell or cell group is selected.

Note Evaluating a cell that involves a lengthy operation may cause a time-out. If this happens, Word displays a time-out message and asks whether you want to continue waiting for a response or terminate the request. If you choose to continue, Word resets the time-out value and continues waiting for a response. Word sets the time-out value; you cannot change it.

See Also

For more information, see "Evaluating MATLAB Commands with Notebook" on page 9-16. For information about evaluating the entire M-book, see "Evaluating an Entire M-Book" on page 9-19.

Evaluate Loop

Evaluate Loop evaluates the selected input cells repeatedly.

For more information, see "Using a Loop to Evaluate Input Cells Repeatedly with Notebook" on page 9-20.

Evaluate M-Book

Evaluate M-book evaluates the entire M-book, sending all input cells to MATLAB to be evaluated. Notebook begins at the top of the M-book regardless of the cursor position.

Result

As Notebook evaluates each input cell, it generates an output cell. When you evaluate an input cell for which there is no output cell, Notebook places the output cell immediately after the input cell that generated it. If you evaluate an input cell for which there is an output cell, Notebook replaces the results in the output cell wherever it is in the M-book.

See Also

For more information, see "Evaluating an Entire M-Book" on page 9-19.

Group Cells

Group Cells converts the input cells in the selection into a single multiline input cell called a cell group. You evaluate a cell group using **Evaluate Cell**. When you evaluate a cell group, all of its output follows the group and appears in a single output cell.

Result

If you include text in the selection, Notebook moves it after the cell group. However, if text precedes the first input cell in the group, the text will remain before the group.

If you include output cells in the selection, Notebook deletes them. If you select all or part of an output cell before selecting this feature, Notebook includes its input cell in the cell group.

If the first line in the cell group is an autoinit cell, the entire group acts as a sequence of autoinit cells. Otherwise, the group acts as a sequence of input cells. You can convert an entire cell group to an autoinit cell by using **Define AutoInit Cell**.

See Also

For more information, see "Defining Cell Groups for Notebook" on page 9-12. For information about converting a cell group to individual input cells, see "Ungroup Cells" on page 9-36.

Hide Cell Markers

Hide Cell Markers hides cell markers in the M-book.

When you select this feature, it changes to **Show Cell Markers**.

Note Notebook does not print cell markers whether you choose to hide them or show them on the screen.

Notebook Options

Notebook Options allows you to examine and modify display options for numeric and graphic output.

See Also

See "Printing and Formatting an M-Book" on page 9-22 for more information.

Purge Selected Output Cells

Purge Selected Output Cells deletes all output cells from the current selection.

See Also

For more information, see "Deleting Output Cells with Notebook" on page 9-21.

Toggle Graph Output for Cell

Toggle Graph Output for Cell suppresses or allows graphic output from an input cell.

If an input or autoinit cell generates figure output that you want to suppress, place the cursor in the input cell and choose this feature. The string (`no graph`) will be placed after the input cell to indicate that graph output for that cell will be suppressed.

To allow graphic output for that cell, place the cursor inside the input cell and choose **Toggle Graph Output for Cell** again. The (`no graph`) marker will be removed. This feature overrides the **Embed Figures in M-book** option, if that option is set in the **Notebook Options** dialog box.

See Also

See "Embedding Graphic Output in the M-Book" on page 9-24 and "Suppressing Graphic Output for Individual Input Cells in Notebook" on page 9-25 for more information.

Undefine Cells

Undefine Cells converts the selected cells to text. If no cells are selected but the cursor is in a cell, Notebook undefines that cell. Notebook removes the cell markers and reformats the cell according to the Normal style.

If you undefine an input cell, Notebook automatically undefines its output cell. However, if you undefine an output cell, Notebook does not undefine its input cell. If you undefine an output cell containing an embedded graphic, the graphic remains in the M-book but is no longer associated with an input cell.

See Also

For information about the Normal style, see "Modifying Styles in the M-Book Template" on page 9-22. For information about deleting output cells, see "Purge Selected Output Cells" on page 9-35.

Ungroup Cells

Ungroup Cells converts the current cell group into a sequence of individual input cells or autoinit cells. If the cell group is an input cell, Notebook converts the cell group to input cells. If the cell group is an autoinit cell, Notebook converts the cell group to autoinit cells. Notebook deletes the output cell for the cell group.

A cell group is the current cell group if

- The cursor is in the cell group.
- The cursor is at the end of a line that contains the closing cell marker for the cell group.
- The cursor is in the output cell for the cell group.
- The cell group is selected.

See Also

For information about creating cell groups, see the description of "Defining Cell Groups for Notebook" on page 9-12.

10

Source Control Interface

The source control interface provides access to your source control system from the MATLAB desktop. Source control systems, also known as version control, revision control, configuration management, and file management systems, are platform dependent—the topics for the Microsoft Windows platforms appear first, followed by the topics for the UNIX platforms.

Source Control Interface on Microsoft Windows

If you use source control systems to manage your files, you can interface with the systems to perform source control actions from within the MATLAB, Simulink, and Stateflow® products. Use menu items in the MATLAB, Simulink, or Stateflow products, or run functions in the MATLAB Command Window to interface with your source control systems.

The source control interface on Windows works with any source control system that conforms to the Microsoft Common Source Control standard, Version 1.1. If your source control system does not conform to the standard, use a Microsoft Source Code Control API wrapper product for your source control system so that you can interface with it from the MATLAB, Simulink, and Stateflow products.

This documentation uses the Microsoft® Visual SourceSafe® software as an example. Your source control system might use different terminology and not support the same options or might use them in a different way. Regardless, you should be able to perform similar actions with your source control system based on this documentation.

Perform most source control interface actions from the Current Directory browser. You can also perform many of these actions for a single file from the MATLAB Editor, a Simulink model window, or a Stateflow chart window—for more information, see "Performing Source Control Actions from the Editor, Simulink, or Stateflow File Menu on Microsoft Windows" on page 10-23. Another way to access many of the source control actions is with the `verctrl` function.

Setting Up the Source Control Interface on Microsoft Windows

In this section...
"Create Projects in Source Control System" on page 10-3
"Specify Source Control System with MATLAB Software" on page 10-5
"Register Source Control Project with MATLAB Software" on page 10-7
"Add Files to Source Control" on page 10-9

Create Projects in Source Control System

In your source control system, create the projects that your directories and files will be associated with.

All files in a directory must belong to the same source control project. Be sure the working directory for the project in the source control system specifies the correct path to the directory on disk.

Example of Creating Source Control Project

This example uses the project my_thesis_files in Microsoft Visual SourceSafe. This illustration of the Current Directory browser shows the path to the directory on disk, I:\my_thesis_files.
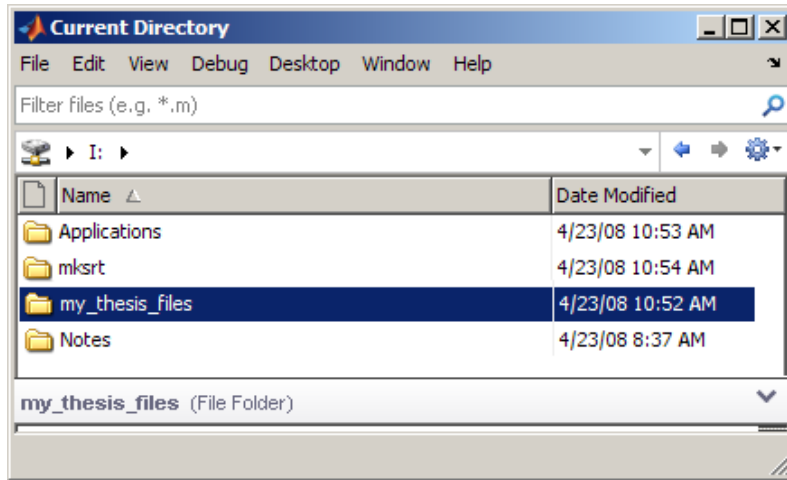
The following illustration shows the example project in the source control system.
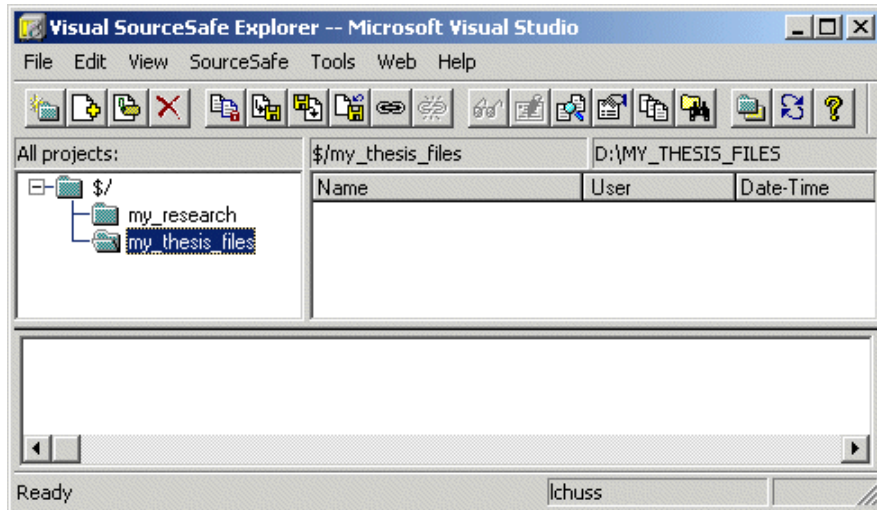
To set the working directory in Microsoft Visual SourceSafe for this example, select my_thesis_files, right-click, select **Set Working Folder** from the context menu, and specify D:\my_thesis_files in the resulting dialog box.
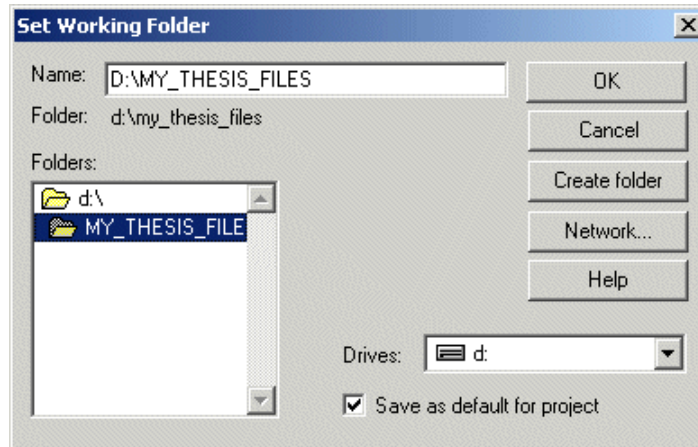
Specify Source Control System with MATLAB Software

In MATLAB, specify the source control system you want to access. Select **File > Preferences > General > Source Control**.

The currently selected system is shown in the Preferences dialog box. The list includes all installed source control systems that support the Microsoft Common Source Control standard.

Select the source control system you want to interface with and click **OK**.
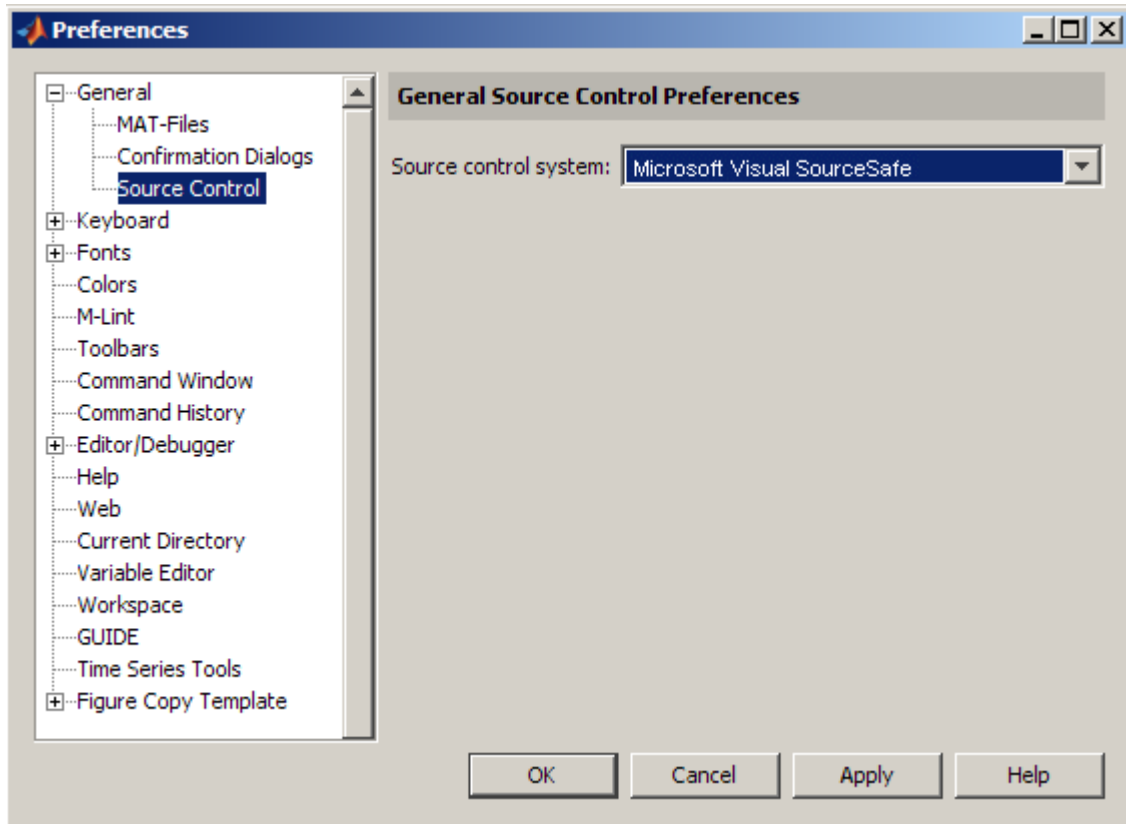
MATLAB remembers preferences between sessions, so you only need to perform this action again when you want to access a different source control system.

Source Control with 64-Bit Versions of MATLAB

If you run a 64-bit version of MATLAB and want MATLAB to interface with your source control system, your source control system must be 64-bit compliant. If you have a 32-bit source control system, or if you have a 64-bit source control system running in 32-bit compatibility mode, MATLAB cannot use it. In that event, MATLAB displays a warning about the problem in the Source Control preference pane.

Function Alternative for Specifying Source Control System

A function alternative to select a source control system is not available, but you can list all available source control systems using `verctrl` with the `all_systems` argument. Use `cmopts` to display the name of the currently selected source control system.

Register Source Control Project with MATLAB Software

Register a source control system project with a directory in MATLAB, that is, associate a source control system project with a directory and all files in that directory. Do this only one time for any file in the directory, which registers all files in that directory:

1 In the MATLAB Current Directory browser, select a file that is in the directory you want to associate with a project in your source control system. For example, select `D:\my_thesis_files\wind.m`. This will associate all files in the `my_thesis_files` directory.

2 Right-click, and from the context menu, select **Source Control > Register Name_of_Source_Control_System Project with MATLAB**. The **Name_of_Source_Control_System** is the source control system you selected using preferences as described in "Specify Source Control System with MATLAB Software" on page 10-5.

The following example shows Microsoft Visual SourceSafe.
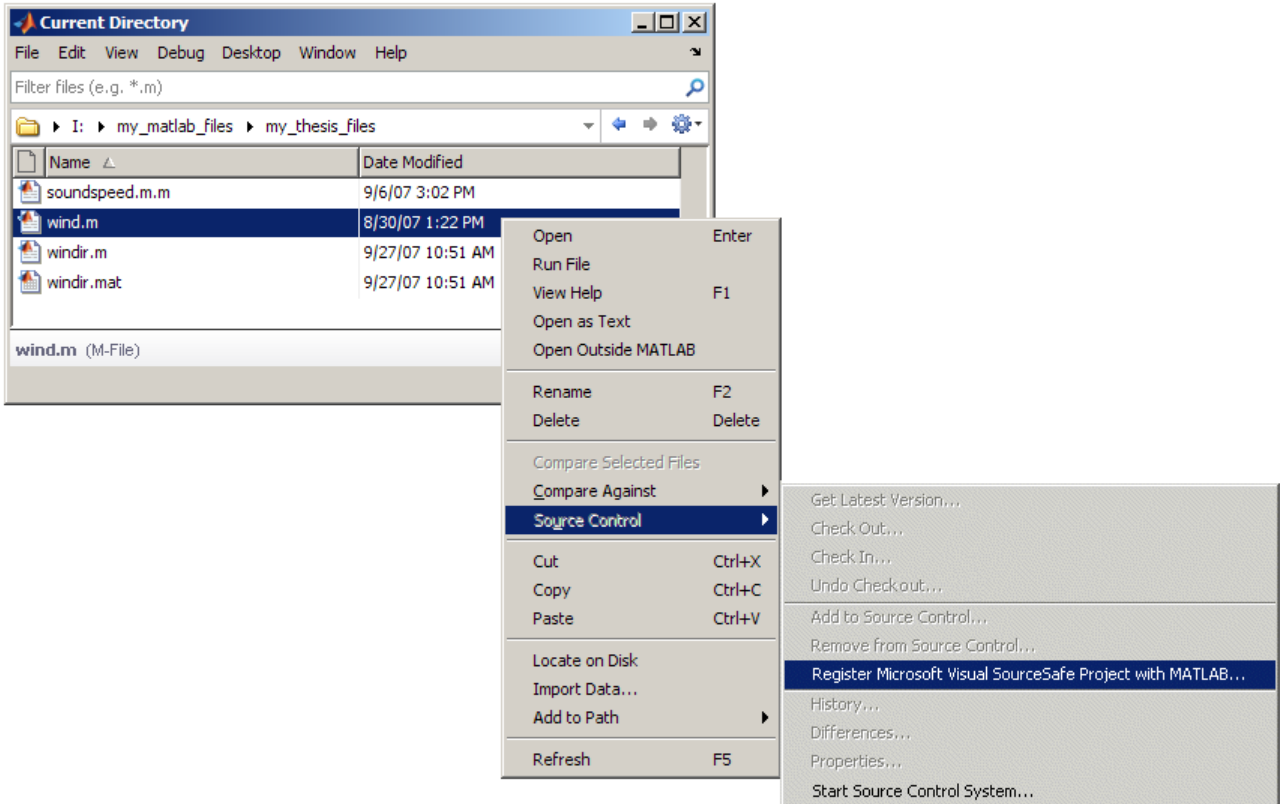
3 In the resulting **Name_of_Source_Control_System Login** dialog box, provide the username and password you use to access your source control system, and click **OK**.
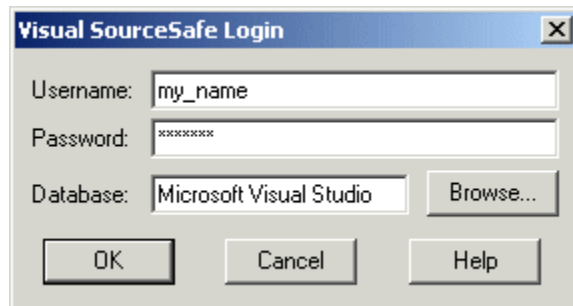
4 In the resulting **Choose project from Name_of_Source_Control_System** dialog box, select the source control system project to associate with the directory and click **OK**. This example shows my_thesis_files.
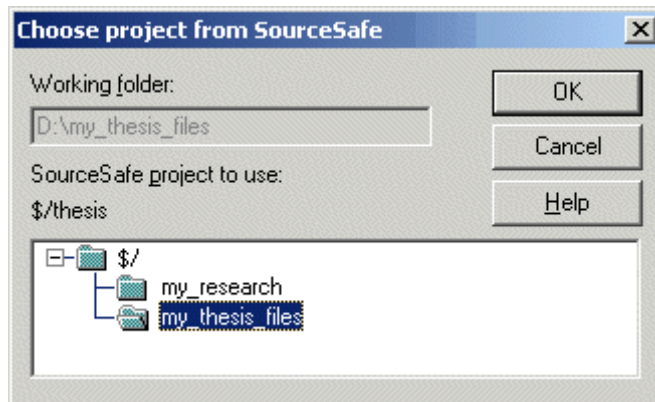
The selected file, its directory, and all files in the directory, are associated with the source control system project you selected. For the example, MATLAB associates all files in D:\my_thesis_files with the source control project my_thesis_files.

Add Files to Source Control

Add files to the source control system. Do this only once for each file:

1 In the Current Directory browser, select files you want to add to the source control system.

2 Right-click, and from the context menu, select **Source Control > Add to Source Control**.

3 The resulting **Add to source control** dialog box lists files you selected to add. You can add text in the **Comments** field. If you expect to use the files soon, select the **Keep checked out** check box (which is selected by default). Click **OK**.
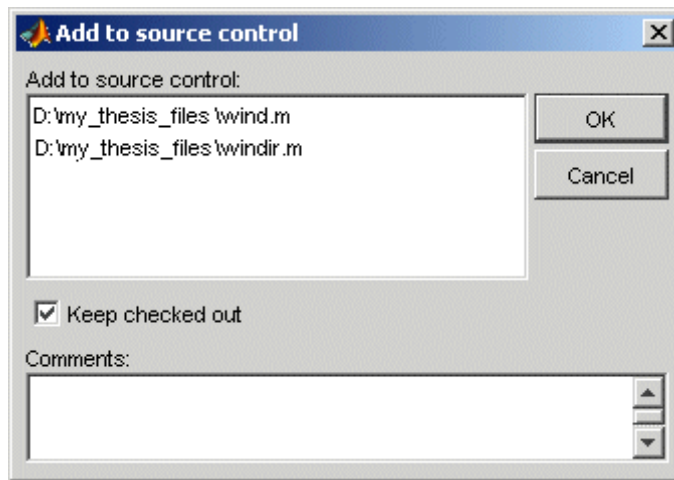
If you try to add an unsaved file, the file is automatically saved upon adding.

Function Alternative

The function alternative is `verctrl` with the `add` argument.

Checking Files Into and Out of Source Control from the MATLAB Desktop on Microsoft Windows

In this section...
"Check Files Into Source Control" on page 10-11
"Check Files Out of Source Control" on page 10-12
"Undoing the Checkout" on page 10-13

Before checking files into and out of your source control system from the MATLAB desktop, be sure to set up your system for use with MATLAB as described in "Setting Up the Source Control Interface on Microsoft Windows" on page 10-3.

Check Files Into Source Control

After creating or modifying files using MATLAB software or related products, check the files into the source control system by performing these steps:

1 In the Current Directory browser, select the files to check in. A file can be open or closed when you check it in, but it must be saved, that is, it cannot contain unsaved changes.

2 Right-click, and from the context menu, select **Source Control > Check In**.

3 In the resulting **Check in file(s)** dialog box, you can add text in the **Comments** field. If you want to continue working on the files, select the check box **Keep checked out**. Click **OK**.

If a file contains unsaved changes when you try to check it in, you will be prompted to save the changes to complete the checkin. If you did not keep the file checked out and you keep the file open, note that it is a read-only version.

Function Alternative

The function alternative is `verctrl` with the `checkin` argument.

Check Files Out of Source Control

From MATLAB, to check out the files you want to modify, perform these steps:

1 In the Current Directory browser, select the files to check out.

2 Right-click, and from the context menu, select **Source Control > Check Out**.

3 The resulting **Check out file(s)** dialog box lists files you selected to check out. Enter comment text in the **Comments** field, which appears if your source control system supports comments on checkout. Click **OK**.
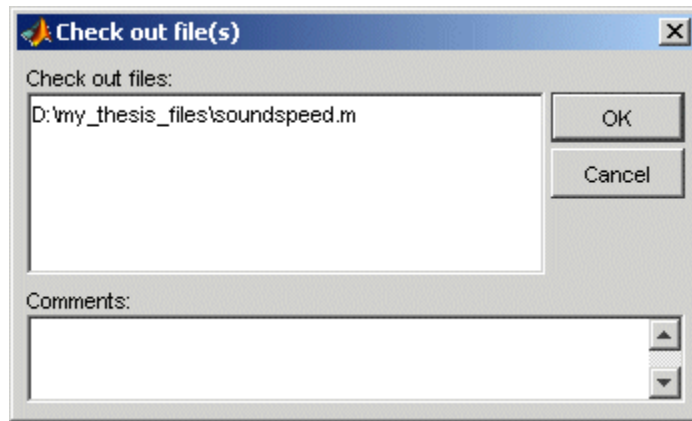
After checking out a file, make changes to it in MATLAB or another product, and save the file. For example, edit an M-file in the Editor.

If you try to change a file without first having checked it out, the file is read-only, as seen in the title bar, and you will not be able to save any changes. This protects you from accidentally overwriting the source control version of the file.

If you end the MATLAB session, the file remains checked out. You can check in the file from within MATLAB during a later session, or directly from your source control system.

Function Alternative

The function alternative is `verctrl` with the `checkout` argument.

Undoing the Checkout

You can undo the checkout for files. The files remain checked in, and do not have any of the changes you made since you last checked them out. To save any changes you have made since checking out a particular file select **File > Save As**, and supply a different file name before you undo the checkout.

To undo a checkout, follow these steps:

1 In the MATLAB **Current Directory** browser, select the files for which you want to undo the checkout.

2 Right-click, and from the context menu, select **Source Control > Undo Checkout**.

The MATLAB **Undo checkout** dialog box opens, listing the files you selected.
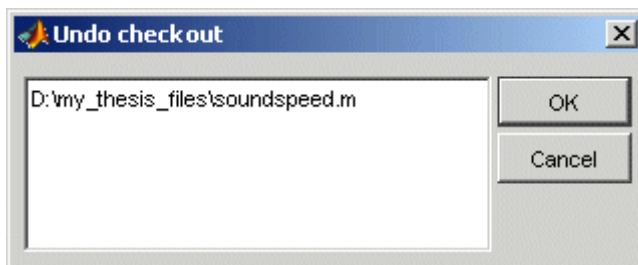
3 Click **OK**.

Function Alternative

The function alternative is `verctrl` with the `undocheckout` argument.

Additional Source Control Actions on Microsoft Windows

In this section...
"Getting the Latest Version of Files for Viewing or Compiling" on page 10-14
"Removing Files from the Source Control System" on page 10-15
"Showing File History" on page 10-16
"Comparing the Working Copy of a File to the Latest Version in Source Control" on page 10-18
"Viewing Source Control Properties of a File" on page 10-20
"Starting the Source Control System" on page 10-21

Getting the Latest Version of Files for Viewing or Compiling

You can get the latest version of a file from the source control system for viewing or running. Getting a file differs from checking it out. When you get a file, it is write protected so you cannot edit it, but when you check out a file, you can edit it.

To get the latest version, follow these steps:

1 In the MATLAB **Current Directory** browser, select the directories or files that you want to get. If you select files, you cannot also select directories.

2 Right-click, and from the context menu, select **Source Control > Get Latest Version**.

The MATLAB **Get latest version** dialog box opens, listing the files or directories you selected.
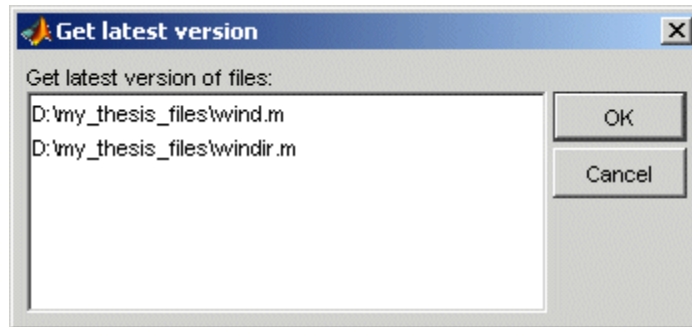
3 Click **OK**.

You can now open the file to view it, run the file, or check out the file for editing.

Function Alternative

The function alternative is `verctrl` with the `get` argument.

Removing Files from the Source Control System

To remove files from the source control system, follow these steps:

1 In the MATLAB **Current Directory** browser, select the files you want to remove.

2 Right-click, and from the context menu, select **Source Control > Remove from Source Control**.

The MATLAB **Remove from source control** dialog box opens, listing the files you selected.
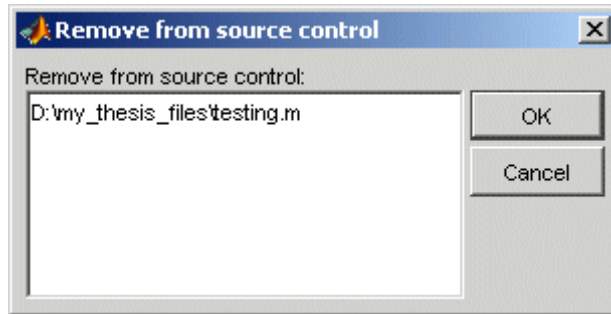
3 Click **OK**.

Function Alternative

The function alternative is `verctrl` with the `remove` argument.

Showing File History

To show the history of a file in the source control system, follow these steps:

1 In the MATLAB **Current Directory** browser, select the file for which you want to view the history.

2 Right-click, and from the context menu, select **Source Control > History**.

A dialog box, which is specific to your source control system, opens. For Microsoft Visual SourceSafe, the **History Options** dialog box opens, as shown in the following example illustration.
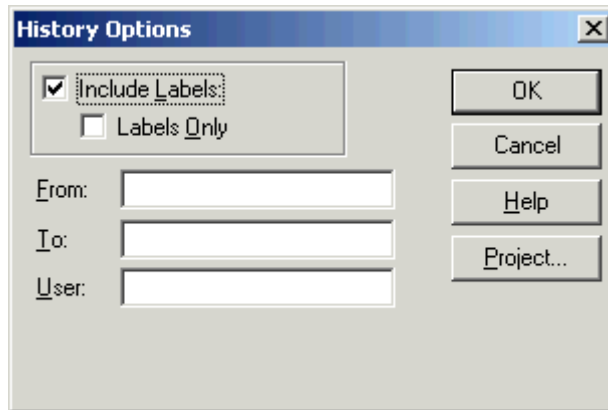
3 Complete the dialog box to specify the range of history you want for the selected file and click **OK**. For example, enter my_name for **User**.

The history presented depends on your source control system. For Microsoft Visual SourceSafe, the **History** dialog box opens for that file, showing the file's history in the source control system.
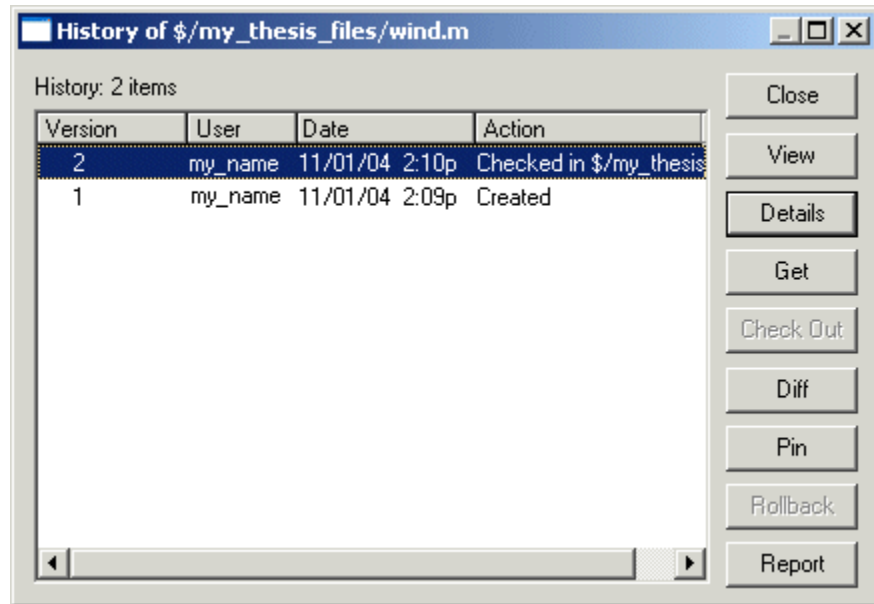
Function Alternative
The function alternative is verctrl with the history argument.

Comparing the Working Copy of a File to the Latest Version in Source Control

You can compare the current working copy of a file with the latest checked-in version of the file in the source control system. This highlights the differences between the two files, showing the changes you made since you checked out the file.

To view the differences, follow these steps:

1 In the MATLAB **Current Directory** browser, select the file for which you want to view differences. This is a file that has been checked out and edited.

2 Right-click, and from the context menu, select **Source Control > Differences**.

A dialog box, which is specific to your source control system, opens. For Microsoft Visual SourceSafe, the **Difference Options** dialog box opens.

3 Review the default entries in the dialog box, make any needed changes, and click **OK**. The following example is for Microsoft Visual SourceSafe.
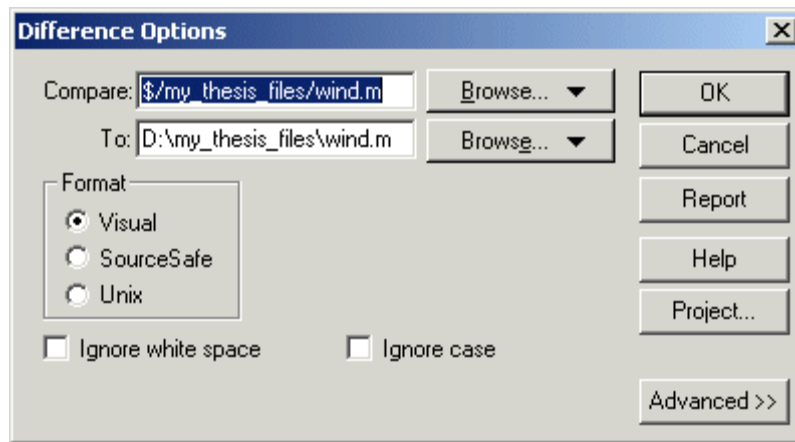
The method of presenting differences depends on your source control system. For Microsoft Visual SourceSafe, the **Differences for** dialog box opens. This highlights the differences between the working copy of the file and the latest checked-in version of the file.
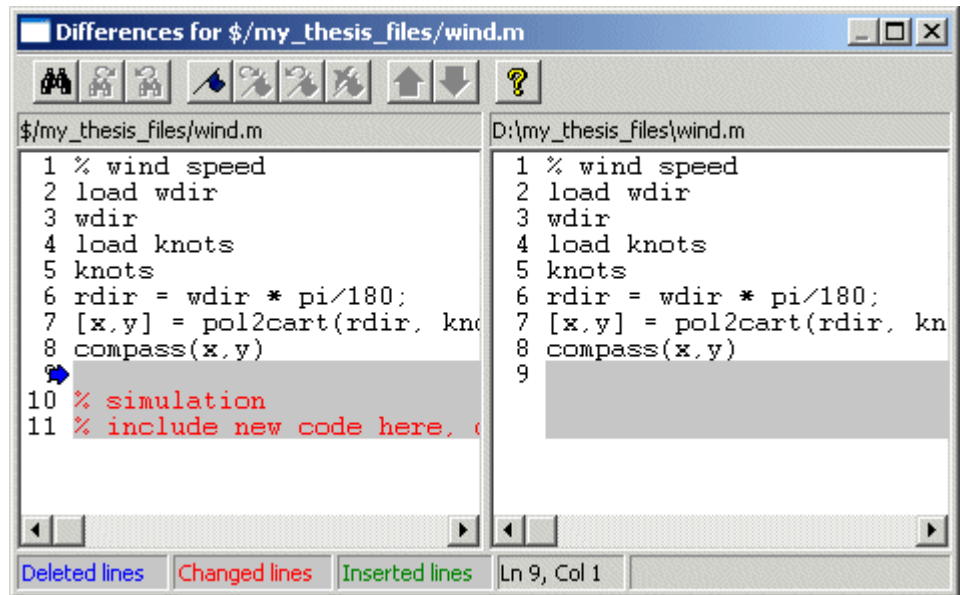
Function Alternative

The function alternative is verctrl with the showdiff or isdiff argument.

Viewing Source Control Properties of a File

To view the source control properties of a file, follow these steps:

1 In the MATLAB **Current Directory** browser, select the file for which you want to view properties.

2 Right-click, and from the context menu, select **Source Control > Properties**.

A dialog box, which is specific to your source control system, opens. The following example shows the Microsoft Visual SourceSafe properties dialog box.
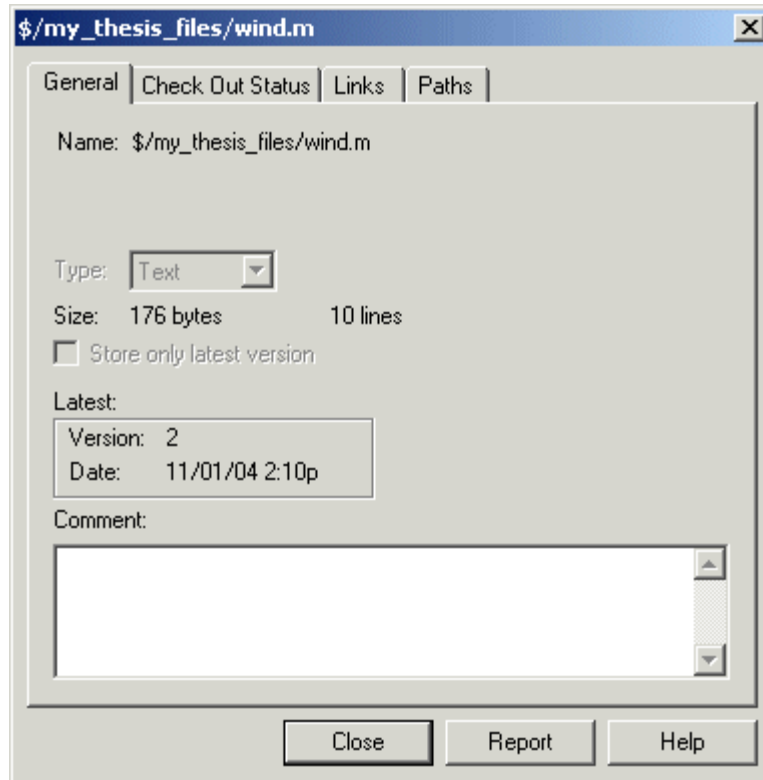
Function Alternative

The function alternative is `verctrl` with the `properties` argument.

Starting the Source Control System

All the MATLAB source control actions automatically start the source control system to perform the action, if the source control system is not already open. If you want to start the source control system from MATLAB without performing a specific action source control action,

1 Right-click any directory or file in the MATLAB Current Directory browser

2 From the context menu, select **Source Control > Start Source Control System**.

The interface to your source control system opens, showing the source control project associated with the current directory in MATLAB. The following example shows the Microsoft Visual SourceSafe Explorer interface.
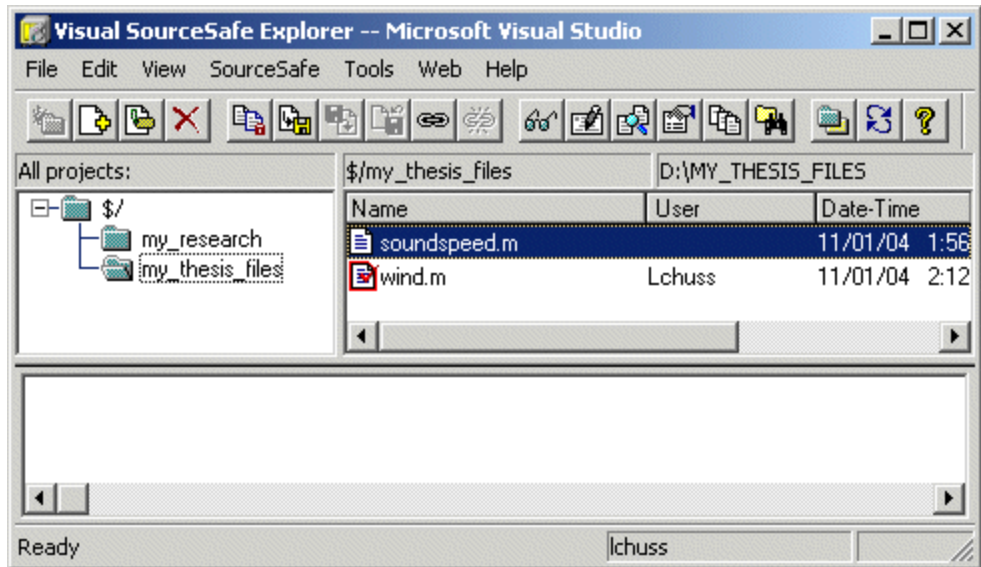
Function Alternative

The function alternative is `verctrl` with the `runscc` argument.

Performing Source Control Actions from the Editor, Simulink, or Stateflow File Menu on Microsoft Windows

You can create or open a file in the Editor, the Simulink or Stateflow products and perform most source control actions from their **File > Source Control** menus, rather than from the Current Directory browser as described in previous sections. Following are some differences in the source control interface process when you use the Editor, Simulink, or Stateflow:

- You can perform actions on only one file at time.

- Some of the dialog boxes have a different icon in the title bar. For example, the **Check out file(s)** dialog box uses an M-file Editor document icon instead of the MATLAB icon.

- You cannot add a new (Untitled) file, but must instead first save the file.

- You cannot register projects from the Simulink or Stateflow products. Instead, register a project using the Current Directory browser, as described in "Register Source Control Project with MATLAB Software" on page 10-7.

Troubleshooting Source Control Problems on Microsoft Windows

In this section...
"Source Control Error: Provider Not Present or Not Installed Properly" on page 10-24
"Restriction Against @ Character" on page 10-25
"Add to Source Control Is the Only Action Available" on page 10-25
"More Solutions for Source Control Problems" on page 10-25

Source Control Error: Provider Not Present or Not Installed Properly

In some cases, MATLAB software recognizes your source control system but you cannot use source control features for MATLAB. Specifically, when you select **File > Preferences > General > Source Control**, or run cmopts, MATLAB lists your source control system, but you cannot perform any source control actions. Only the **File > Source Control > Start Source Control System** menu item is available, and when you select it, MATLAB displays this error:

```
Source control provider is not present or not installed properly.
```

Often, this error occurs because a registry key that MATLAB requires from the source control application is not present. Make sure this registry key is present:

```
HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider\
InstalledSCCProviders
```

The registry key refers to another registry key that is similar to

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SourceSafe\SccServerPath
```

This registry key has a path to a DLL-file in the file system. Make sure the DLL-file exists in that location. If you are not familiar with registry keys, ask your system administrator for help.

If this does not solve the problem and you use Microsoft Visual SourceSafe, try running a client setup for your source control application. When SourceSafe is installed on a server for a group to use, each machine client can run a setup but is not required to do so. However, some applications that interface with SourceSafe, including MATLAB, require you to run the client setup. Run the client setup, which should resolve the problem.

If the problem persists, access source control outside of MATLAB.

Restriction Against @ Character

Some source control systems, such as Perforce® and Synergy™, reserve the @ character. Perforce, for example, uses it as a revision specifier. Therefore, you might experience problems if you use these source control systems with MATLAB files and directories that include the @ character in the directory or file name.

You might be able to work around this restriction by quoting nonstandard characters in file names, such as with an escape sequence, which some source control systems allow. Consult your source control system documentation or technical support resources for a workaround.

Add to Source Control Is the Only Action Available

To use source control features for a file in the Simulink or Stateflow products, the file's source control project must first be registered with MATLAB. When a file's source control project is *not* registered with MATLAB, all **File > Source Control** menu items are disabled except **Add to Source Control**. You can select **Add to Source**, which registers the project with MATLAB, or you can register the project using the Current Directory browser, as described in "Register Source Control Project with MATLAB Software" on page 10-7. You can then perform source control actions for all files in that project (directory).

More Solutions for Source Control Problems

The latest solutions for problems interfacing MATLAB with a source control system appear on the MathWorks Web page for support at http://www.mathworks.com/support/. Search Solutions and Technical Notes for "source control."

Source Control Interface on UNIX Platforms

If you use a source control system to manage your files, you can check M-files and Simulink models, and Stateflow charts into and out of the source control system from within the MATLAB, Simulink, and Stateflow products.

The source control interface supports four popular source control systems, as well as a custom option:

- ClearCase® software from IBM® Rational®
- Concurrent Version System (CVS)
- ChangeMan® and PVCS® software from Serena®
- Revision Control System (RCS)
- Custom option — Allows you to build your own interface if you use a different source control system. For details, see the reference page for customverctrl.

Perform source control interface actions for a single file using menu items in the MATLAB Editor, a Simulink model window, or a Stateflow chart window. To perform source control actions on multiple files, use the Current Directory browser. Alternatively, run source control functions in the Command Window, which provide some options not supported with the menu items.

Specifying the Source Control System on UNIX Platforms

In this section...

"MATLAB Desktop Alternative" on page 10-27

"Function Alternative" on page 10-28

"Setting a View and Checking Out a Directory with ClearCase Software on UNIX Platforms" on page 10-29

MATLAB Desktop Alternative

To specify the source control system you want to access, select
File > Preferences > General > Source Control.

The currently selected system is shown in the Preferences dialog box. The default selection is None.

Select the source control system with which you want to interface and click **OK**.
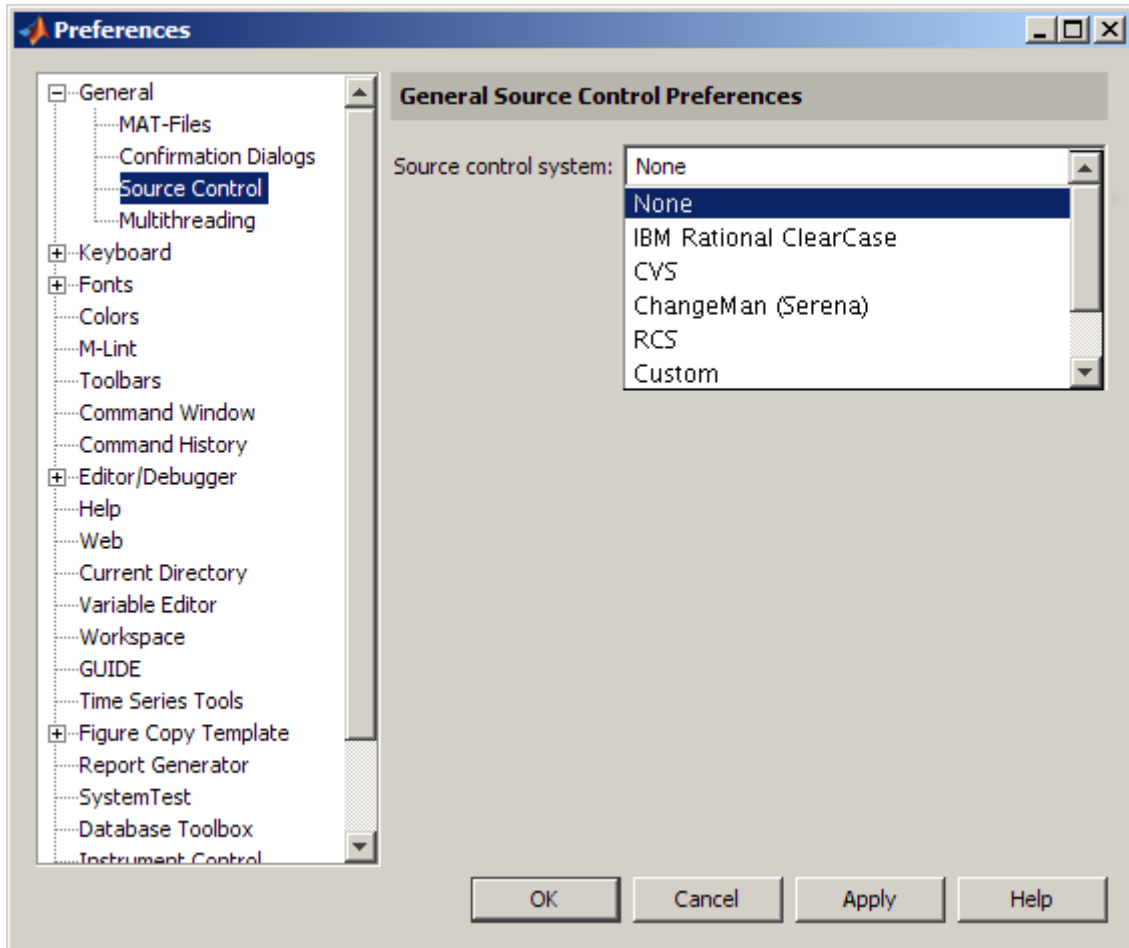
MATLAB remembers preferences between sessions, so you only need to perform this action when you want to access a different source control system.

Function Alternative

A function alternative to select a source control system is not available, but you can list the currently selected source control system by running `cmopts`.

Setting a View and Checking Out a Directory with ClearCase Software on UNIX Platforms

If you use ClearCase software on a UNIX platform, perform the following from ClearCase:

1 Set a view.

2 Check out the directory that contains files you want to save, check in, or check out.

You can now use the MATLAB, Simulink, or Stateflow source control interfaces to ClearCase software.

Checking Files Into the Source Control System on UNIX Platforms

In this section...

"Checking In One or More Files Using the Current Directory Browser" on page 10-30

"Checking In One File Using the Editor, or the Simulink or Stateflow Products" on page 10-31

"Function Alternative" on page 10-32

Checking In One or More Files Using the Current Directory Browser

1 From the Current Directory browser, select the file or files to check in. A file can be open or closed when you check it in, but it must be saved, that is, it cannot contain unsaved changes.

2 Right-click, and from the context menu, select **Source Control > Check In**.

3 In the resulting **Check in file(s)** dialog box, you can add text in the **Comments** field. If you want to continue working on the files, select the check box **Keep checked out**. Click **OK**.
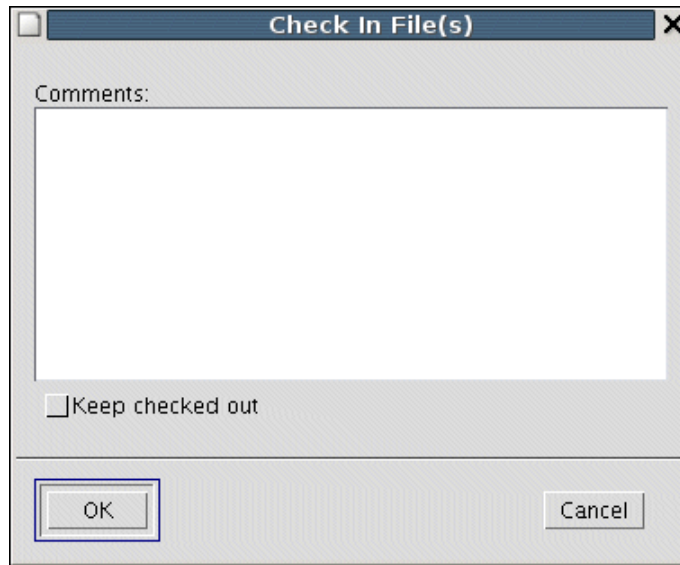
The files are checked into the source control system. If any file contains unsaved changes when you try to check it in, you will be prompted to and must then save the changes to complete the checkin.

An error appears in the Command Window if a file is already checked in.

If you did not keep a file checked out and you keep that file open, note that it is a read-only version.

Checking In One File Using the Editor, or the Simulink or Stateflow Products

1 From the Editor, or the Simulink or Stateflow products, with the file open and saved, select **File > Source Control > Check In**.

2 In the resulting **Check in file(s)** dialog box, you can add text in the **Comments** field. If you want to continue working on the files, select the check box **Keep checked out**. Click **OK**.

Function Alternative

Use `checkin` to check files into the source control system. The files can be open or closed when you use `checkin`. The `checkin` function takes this form:

```
checkin({'file1','file2', ...},'comments','comment_text',...
'option','value')
```

For `filen`, use the complete path and include the file extension. You must supply the `comments` argument and a comments string with `checkin`.

Use the `option` argument to

- Check in a file and keep it checked out — set the `lock` option value to `on`.

- Check in a file even though it has not changed since the previous check in — set the `force` option value to `on`.

The `comments` argument and the `lock` and `force` options apply to all files checked in.

Example Using checkin Function

To check in the file `clock.m` with the comment `Adjustment for leap year`, type

```
checkin('\myserver\mymfiles\clock.m','comments', ...
'Adjustment for leap year')
```

For other examples, see the reference page for `checkin`.

Checking Files Out of the Source Control System on UNIX

Checking Out One or More Files Using the Current Directory Browser

1 In the Current Directory browser, select the file or files to check out.

2 Right-click, and from the context menu, select **Source Control > Check Out**. The **Check out file(s)** dialog box opens.
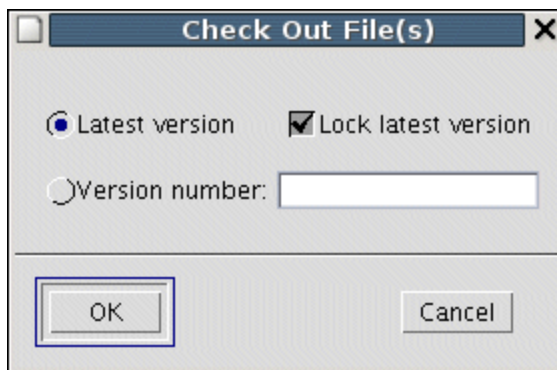
3 Complete the dialog box:

   a To check out the versions that were most recently checked in, select the **Latest version** option.

   b To check out a specific version of the files, select the **Version number** option and type the version number in the field.

    c To prevent others from checking out the files while you have them
       checked out, select **Lock latest version**. To check out read-only
       versions of the file, clear **Lock latest version**.

4 Click **OK**.

An error appears in the Command Window if a file is already checked out.

After checking out files, make changes to them using MATLAB software or
another software product, and save the files. For example, edit an M-file
in the Editor.

If you try to change a file without first having checked it out, the file is
read-only, as seen in the title bar, and you will not be able to save any
changes. This protects you from accidentally overwriting the source control
version of the file.

If you end the MATLAB session, the file or files remain checked out. You can
check in files from within MATLAB during a later session, or directly from
your source control system.

Checking Out a Single File Using the Editor, or the Simulink or Stateflow Products

1 Open the M-file, Simulink model, or Stateflow chart you want to check out.
The title bar indicates the file is read-only.

2 Select **File > Source Control > Check Out**. The **Check out file(s)**
dialog box opens.

3 Complete the dialog box as described in step of "Checking Out One or More
Files Using the Current Directory Browser" on page 10-33, and click **OK**.

Function Alternative

Use checkout to check out a file from the source control system. You can
check out multiple files at once and specify checkout options. The checkout
function takes this form:

```
checkout({'file1','file2', ...},'option','value')
```

For `filen`, use the complete path and include the file extension.

Use the `option` argument to

- Check out a read-only version of the file — set the `lock` option value to `off`.

- Check out the file even if you already have it checked out — set the `force` option value to `on`.

- Check out a specific version of the file — use the `revision` option, and assign the version number to the `value` argument.

The options apply to all files being checked out. The files can be open or closed when you use `checkout`.

Example Using checkout Function—Check Out a Specific Version of a File

To check out the 1.1 version of the file `clock.m`, type

```
checkout('\myserver\mymfiles\clock.m','revision','1.1')
```

For other examples, see the reference page for `checkout`.

Undoing the Checkout on UNIX Platforms

In this section...

Impact of Undoing a File Checkout

When you undo the checkout for a file, the file remains checked in, and does not have any of the changes you made since you checked it out. To save any changes you have made since checking out a file, select **File > Save As**, and supply a different file name before you undo the checkout. Undo the checkout using the Current Directory browser for one or more files. For only one file, you can also use the Editor, or the Simulink or Stateflow products.

Undoing the Checkout for One or More Files Using the Current Directory Browser

1 In the MATLAB **Current Directory** browser, select the file or files for which you want to undo the checkout.

2 Right-click, and from the context menu, select **Source Control > Undo Checkout**. MATLAB undoes the checkout.

An error appears in the Command Window if the file is not checked out.

Undoing the Checkout for a Single File Using the Editor, or the Simulink or Stateflow Products

1 Open the M-file, Simulink model, or Stateflow chart for which you want to undo the checkout.

2 Select **File > Source Control > Undo Checkout**. MATLAB undoes the checkout.

Function Alternative

The undocheckout function takes this form:

```
undocheckout({'file1','file2', ...})
```

Use the complete path for filen and include the file extension. For example, to undo the checkout for the files clock.m and calendar.m, type

```
undocheckout({'\myserver\mymfiles\clock.m',...
'\myserver\mymfiles\calendar.m'})
```

11

Internationalization

How the MATLAB Process Uses Locale Settings

A *locale* is part of the user environment definition. It defines language, territory, and *codeset*, which is a coded character set. The MATLAB process uses the user-specified locale name on all platforms. MATLAB also reads the user-specified *UI language name*, and uses it to select localized resources in the specified language. By using this feature, a user can select localized resources in US-English. The user-specified UI language setting also controls language and country settings of the Sun™ Java Virtual Machine (JVM) software.

Consider the following when choosing your locale settings. To see what your current settings are, use the instructions in "Setting Locale on Windows Platforms" on page 11-4, "Setting Locale on Linux and Solaris Platforms" on page 11-5, or "Setting Locale on Macintosh Platforms" on page 11-6..

- **Default Locale Setting** — If the user-specified locale is not supported, MATLAB uses the default locale en_US.US-ASCII.

- **UI Language Setting** — The UI language setting should be set to either the same language as the user-specified locale or to US-English. Otherwise, non-7-bit ASCII characters might not display properly.

- **Supported Encoding Scheme** — MATLAB might not properly handle character codes greater than 2 bytes.

- **Supported Character Set** — MATLAB supports the character set specified by the user locale setting.

- **M-File Compatibility** — Non-7-Bit ASCII characters in M-files created on one platform might not be compatible on other platforms using different locale settings.

- **Platform-Specific Localized Formats** — MATLAB usually uses platform-neutral localized formats and rules.

- **On Windows Platforms** — User locale and system locale must be the same value on the Microsoft Windows platform. If these values are not the same, users might see garbled text or incorrect characters. For information on controlling these settings, see "Setting Locale on Windows Platforms" on page 11-4.

- **On Macintosh Platforms** — MATLAB automatically chooses a codeset for each combination of language and territory on the Apple Macintosh OS X platform. In Version 10.5 of the OS X operating system, MATLAB ignores the `LANG` environment variable.

- **Running nodesktop Option On Macintosh OS X Version 10.5 Platforms** — When you run MATLAB software with the `-nodesktop` startup option on the Macintosh OS X Version 10.5 platform, the MATLAB locale setting is not the Macintosh locale setting for the Terminal application. For example, for users selecting the `Japanese_Japan` region on the **Formats** tab, the MATLAB locale setting is `ja_JP.sjis`. The Macintosh locale setting is `ja_JP.UTF-8`.

Setting the Locale

Setting Locale on Windows Platforms

MATLAB software uses the *system locale* and *user locale* on Windows platforms:

- "Setting User Locale on Windows Vista Platforms" on page 11-4
- "Setting System Locale on Windows Vista Platforms" on page 11-4
- "Setting User Locale on Windows XP Platforms" on page 11-5
- "Setting System Locale on Windows XP Platforms" on page 11-5

Setting User Locale on Windows Vista Platforms

1 Select **Start** -> **Control Panel** -> **Regional and Language Options**.

2 Open **Formats** tab.

3 Select an item from the drop-down list.

Setting System Locale on Windows Vista Platforms

1 Select **Start** -> **Control Panel** -> **Regional and Language Options**.

2 Open **Administrative** tab.

3 Click **Change system locale...** button.

4 Select an item from the drop-down list.

5 Reboot the system.

Note When you change the system locale, you must reboot your system; otherwise, you might see unexpected locale-setting behaviors.

Setting User Locale on Windows XP Platforms

1 Select **Start** -> **Control Panel** -> **Regional and Language Options**.

2 Open **Regional Options** tab.

3 Select an item from the drop-down list.

Setting System Locale on Windows XP Platforms

1 Select **Start** -> **Control Panel** -> **Regional and Language Options**.

2 Open **Advanced** tab.

3 Select an item from the drop-down list.

4 Reboot the system.

Note When you change the system locale, you must reboot your system; otherwise, you might see unexpected locale-setting behaviors.

Setting Locale on Linux and Solaris Platforms

Linux[19] and Sun Solaris platforms manage locale settings with six *locale categories*. These are the same categories used by C standard library functions.

The following locale categories are available:

- LC_CTYPE controls character data manipulations.
- LC_COLLATE controls character collation/sorting operations.

---

19. Linux is a registered trademark of Linus Torvalds.

- `LC_TIME` controls date/time data formatting or parsing.

- `LC_NUMERIC` controls numeric data formatting or parsing.

- `LC_MONETARY` controls monetary data formatting or parsing.

- `LC_MESSAGES` controls the user UI language.

Setting User Locale and User UI Language

Use the `LANG` environment variable to specify a single locale for all locale categories. The locale specified with this variable might be partially or entirely over-written by other environment variables.

Use the environment variables `LC_CTYPE`, `LC_COLLATE`, `LC_TIME`, `LC_NUMERIC`, and `LC_MONETARY` to specify a locale for a particular category.

Use the `LC_ALL` environment variable to over-write all locales specified with other environment variables. If a single locale has to be set to all locale categories, use `LANG` instead of `LC_ALL`.

Setting Locale on Macintosh Platforms

The Macintosh OS X platform manages the user locale setting and the user UI language setting.

Setting User Locale

1 Select **System Preferences** ->**International**

2 Open **Formats** tab

3 Select an item from the **Region** pop-up menu

Setting UI Language

1 Select **System Preferences** ->**International**

2 Open **Language** tab

3 Drag an item to the top of the **Languages** list

Calculating Dates in Programs

To ensure the correct calculation of functions using date values, replace `datenum` function calls with the use of the `dir` function `datenum` field.

For example, look at the modification date of your MATLAB `license.txt` file:

```
cd(matlabroot)
f=dir('license.txt')
```

MATLAB displays information similar to:

```
f =
        name: 'license.txt'
        date: '10-May-2007 17:48:22'
       bytes: 5124
       isdir: 0
     datenum: 7.3317e+005
```

If your code uses a command similar to:

```
n=datenum(f.date);
```

you must replace it with:

```
n=f.datenum;
```

Numeric Format Uses C Locale

MATLAB software reads the user locale for all categories except for the `LC_NUMERIC` category. This category controls numeric data formatting and parsing. MATLAB always sets `LC_NUMERIC` to the `C` locale.

For example, some users expect a comma in a number while other users expect a decimal. The value of pi can be displayed as `3.1415` or `3,1415`, depending on the format used by a locale. MATLAB always uses `3.1415`, regardless of the format specified by the user locale.

Index

Symbols and Numerics

A

B

D

F