MATLAB®

The Language of Technical Computing

Desktop Tools and Development Environment
Version 7
Startup and Shutdown

1

Starting MATLAB ................................................. 1-2
  Starting MATLAB on Windows Platforms ..................... 1-2
  Starting MATLAB on UNIX Platforms ......................... 1-3
  Startup Directory for MATLAB ................................. 1-3
  Startup Options ................................................. 1-4
  Toolbox Path Caching in MATLAB ............................... 1-10

Quitting MATLAB ................................................... 1-13
  Running a Script When Quitting MATLAB ..................... 1-13

Desktop

2

Overview of the Desktop .......................................... 2-2
  Example of Desktop—Default Layout ........................... 2-3
  Summary of Desktop Tools ....................................... 2-4

Arranging the Desktop—Overview ................................. 2-5
  Opening and Arranging Tools ................................... 2-6
  Opening and Arranging Documents ............................... 2-7
  Examples of Desktop Arrangements .............................. 2-11
  Saving Desktop Layouts .......................................... 2-18

Common Desktop Features ......................................... 2-19
  Start Button for Accessing Tools ............................... 2-19
  Shortcuts for MATLAB—Easily Run a Group of Statements ... 2-21
  Web Browser ..................................................... 2-29
  Menus and Context Menus ....................................... 2-31
  Toolbars ......................................................... 2-32
  Status Bar ....................................................... 2-34
  Sizing, Arranging, and Sorting Columns in Tools ............ 2-34
Contents

Keyboard Shortcuts ........................................... 2-35
Selecting Multiple Items ........................................ 2-38
Cut, Copy, and Paste ........................................... 2-39
Page Setup Options for Printing ............................... 2-40
Accessing The MathWorks on the Web ...................... 2-42

Fonts, Colors, and Other Preferences ...................... 2-44
Fonts Preferences for Desktop Tools ....................... 2-44
Colors Preferences for Desktop Tools ........................ 2-50
General Preferences for MATLAB ............................ 2-54
About Preferences ........................................... 2-56

Running Functions—Command Window and History

Opening the Command Window .............................. 3-2

Running Functions and Programs, and Entering Variables 3-3
Running Statements at the Command Line Prompt ....... 3-3
Running External Programs ................................ 3-6
Evaluating or Opening a Selection .......................... 3-6
Hyperlinks for Running Functions .......................... 3-7

Controlling Input ............................................. 3-8
Case and Space Sensitivity ................................. 3-8
Syntax Highlighting ......................................... 3-9
Cut, Copy, Paste, and Undo Features ....................... 3-10
Enter Multiple Lines Without Running Them .............. 3-10
Entering Multiple Functions in a Line ...................... 3-11
Entering Long Statements ................................... 3-11
Recalling Previous Lines .................................... 3-12
Tab Completion ............................................ 3-12
Keyboard Shortcuts in the Command Window .............. 3-16
Navigating Above the Command Line ....................... 3-17
Controlling Output .................................................. 3-18
  Suppressing Output .............................................. 3-18
  Paging of Output in the Command Window ................. 3-18
  Formatting and Spacing Numeric Output .................... 3-19
  Clearing the Command Window ............................... 3-20
  Printing Command Window Contents ....................... 3-20
  Keeping a Session Log ....................................... 3-21

Searching in the Command Window ............................. 3-22
  Find Dialog .................................................... 3-22
  Incremental Search ........................................... 3-23

Preferences for the Command Window ......................... 3-27
  Format, Display, and Accessibility Preferences ............ 3-27
  Keyboard and Indenting Preferences for the
  Command Window ............................................ 3-29

Command History ................................................. 3-30
  Viewing Statements in the Command History Window ....... 3-31
  Using Statements from the Command History ............... 3-32
  Searching in the Command History ........................... 3-33
  Printing the Command History ............................... 3-35
  Deleting Entries in the Command History Window ......... 3-35

Preferences for Command History ............................. 3-37
  Settings ....................................................... 3-37
  Saving .......................................................... 3-37

Help for Using MATLAB

Types of Documentation ........................................... 4-2
  Accessing Documentation on the Web ....................... 4-3

Help Browser ..................................................... 4-4
  Resizing the Help Browser ..................................... 4-5
  Adding Your Own Help Files to the Help Browser ........... 4-6
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Find Information with the Help Browser</strong></td>
<td>4-7</td>
</tr>
<tr>
<td>Contents Listing in the Help Browser</td>
<td>4-7</td>
</tr>
<tr>
<td>Index for Help Browser</td>
<td>4-10</td>
</tr>
<tr>
<td>Search Documentation with the Help Browser</td>
<td>4-12</td>
</tr>
<tr>
<td>Favorites</td>
<td>4-18</td>
</tr>
<tr>
<td><strong>View Documentation in the Help Browser</strong></td>
<td>4-19</td>
</tr>
<tr>
<td>Browse to Other Pages</td>
<td>4-20</td>
</tr>
<tr>
<td>Links</td>
<td>4-21</td>
</tr>
<tr>
<td>Find Text in Displayed Pages</td>
<td>4-21</td>
</tr>
<tr>
<td>Copy Information</td>
<td>4-22</td>
</tr>
<tr>
<td>Evaluate a Selection</td>
<td>4-22</td>
</tr>
<tr>
<td>View the Page Source (HTML)</td>
<td>4-22</td>
</tr>
<tr>
<td><strong>Demos in the Help Browser</strong></td>
<td>4-23</td>
</tr>
<tr>
<td>Using Demos</td>
<td>4-23</td>
</tr>
<tr>
<td>Adding Your Own Demos</td>
<td>4-26</td>
</tr>
<tr>
<td><strong>Preferences for the Help Browser</strong></td>
<td>4-27</td>
</tr>
<tr>
<td>Product Filter</td>
<td>4-27</td>
</tr>
<tr>
<td>PDF Reader—Specifying Its Location</td>
<td>4-28</td>
</tr>
<tr>
<td>General—Keep Contents Synchronized</td>
<td>4-28</td>
</tr>
<tr>
<td>Help Fonts Preferences—Specifying Font Name, Style, and Size</td>
<td>4-29</td>
</tr>
<tr>
<td><strong>Printed Documentation</strong></td>
<td>4-30</td>
</tr>
<tr>
<td>Printing a Page from the Help Browser</td>
<td>4-30</td>
</tr>
<tr>
<td>Printing the PDF Version of Documentation</td>
<td>4-30</td>
</tr>
<tr>
<td><strong>Help Functions</strong></td>
<td>4-32</td>
</tr>
<tr>
<td>View Function Reference Pages—the doc Function</td>
<td>4-33</td>
</tr>
<tr>
<td>Getting Help in the Command Window—the help Function</td>
<td>4-34</td>
</tr>
<tr>
<td><strong>Other Forms of Help</strong></td>
<td>4-37</td>
</tr>
<tr>
<td>Documentation for Other Products</td>
<td>4-37</td>
</tr>
<tr>
<td>Product-Specific Help Features</td>
<td>4-37</td>
</tr>
<tr>
<td>User-Contributed M-Files</td>
<td>4-38</td>
</tr>
<tr>
<td>Technical Support</td>
<td>4-38</td>
</tr>
<tr>
<td>Newsgroup for MathWorks Products</td>
<td>4-39</td>
</tr>
</tbody>
</table>
Contents

File Management Operations .................................. 5-29
  Current Directory Field ................................... 5-30
  Current Directory Browser ................................. 5-30
  Viewing and Making Changes to Directories ............. 5-32
  Creating, Renaming, Copying, and Removing Directories and Files ................................... 5-33
  Opening, Running, and Viewing the Content of Files . 5-37
  Finding Files and Content Within Files .................. 5-40
  Accessing Source Control Features ....................... 5-42
  Preferences for the Current Directory Browser .......... 5-42

Editing and Debugging M-Files

Begin with Existing Code .................................... 6-2
  Create M-Files from Command Window and History .... 6-2
  Use Existing M-Files and Examples ....................... 6-2

Ways to Edit and Debug Files ................................ 6-4

Starting, Customizing, and Closing the Editor/Debugger 6-6
  Creating a New File in the Editor/Debugger ................ 6-7
  Opening Existing Files in the Editor/Debugger .......... 6-8
  Opening the Editor Without Starting MATLAB ............ 6-10
  Arranging Editor/Debugger Documents ..................... 6-10
  Preferences for the Editor/Debugger ....................... 6-11
  Working with Other Text File Types ....................... 6-12
  Closing the Editor/Debugger ................................ 6-12

Creating, Editing, and Running Files ....................... 6-13
  Entering Statements ....................................... 6-13
  Appearance of an M-File ................................... 6-19
  Navigating in an M-File .................................... 6-21
  Finding Text in Files ...................................... 6-23
  Opening a Selection in an M-File ......................... 6-28
  Saving M-Files .............................................. 6-29
  Running M-Files from the Editor/Debugger ............... 6-31
Printing M-Files ........................................ 6-32
Closing M-Files ..................................... 6-32

Debugging M-Files .................................... 6-33
Finding Errors in M-Files ....................... 6-33
Debugging Example—The Collatz Problem ..... 6-36
Debugging Process and Features .............. 6-39
Preparing for Debugging ......................... 6-39
Setting Breakpoints ............................... 6-40
Running an M-File with Breakpoints .......... 6-43
Stepping Through an M-File ...................... 6-45
Examining Values .................................. 6-46
Correcting Problems and Ending Debugging .... 6-51
Conditional Breakpoints ......................... 6-58
Error Breakpoints .................................. 6-60

Rapid Code Iteration Using Cells ............. 6-63
Defining Cells ...................................... 6-64
Navigating and Evaluating with Cells .......... 6-69
Using Cells in Function M-Files ............... 6-73

Tuning and Managing M-Files

Visual Directory in Current Directory Browser .... 7-2
Navigate Directory Hierarchy .................... 7-4
View and Edit Files ............................ 7-5
Sort by Contents.m ................................ 7-6
Run, Make Thumbnail, Delete File (Show Actions) .... 7-7
Show File Sizes ................................... 7-9
Show Function/Script ............................ 7-9

Directory Reports in Current Directory Browser .... 7-11
Accessing and Using Directory Reports ........ 7-11
TODO/FIXME Report ............................... 7-12
Help Report ........................................ 7-13
Contents Report ................................... 7-17
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependency Report</td>
<td>7-20</td>
</tr>
<tr>
<td>File Comparison Report</td>
<td>7-22</td>
</tr>
<tr>
<td>Coverage Report</td>
<td>7-24</td>
</tr>
<tr>
<td><strong>M-Lint Code Check Report</strong></td>
<td>7-25</td>
</tr>
<tr>
<td>Accessing M-Lint</td>
<td>7-25</td>
</tr>
<tr>
<td>M-Lint Graphical User Interface (GUI)</td>
<td>7-25</td>
</tr>
<tr>
<td>Making Changes Based on M-Lint Messages</td>
<td>7-28</td>
</tr>
<tr>
<td><strong>Profiling for Improving Performance</strong></td>
<td>7-35</td>
</tr>
<tr>
<td>What Is Profiling?</td>
<td>7-35</td>
</tr>
<tr>
<td>The Profiling Process: Guidelines</td>
<td>7-36</td>
</tr>
<tr>
<td>The Profiler</td>
<td>7-38</td>
</tr>
<tr>
<td>Profile Summary Report</td>
<td>7-42</td>
</tr>
<tr>
<td>Profile Detail Report</td>
<td>7-45</td>
</tr>
<tr>
<td>The profile Function</td>
<td>7-51</td>
</tr>
<tr>
<td><strong>Publishing Results</strong></td>
<td>8</td>
</tr>
</tbody>
</table>

**Publishing to HTML, XML, LaTeX, Word, and PowerPoint**

**Using Cells** 8-2
- Overview of Publishing 8-2
- Example of Publishing Without Text Markup 8-3
- Example of Publishing with Text Markup 8-4

**Marking Up Text in Cells for Publishing** 8-9

**Publishing M-Files Using Cells** 8-15
- How to Publish an M-File 8-15
- About Published M-Files 8-16
- Modifying Published Output Via Preferences 8-17
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notebook for Publishing to Word</td>
<td>8-18</td>
</tr>
<tr>
<td>Creating an M-Book</td>
<td>8-18</td>
</tr>
<tr>
<td>Entering MATLAB Commands in an M-Book</td>
<td>8-21</td>
</tr>
<tr>
<td>Protecting the Integrity of Your Workspace in M-Books</td>
<td>8-22</td>
</tr>
<tr>
<td>Ensuring Data Consistency in M-Books</td>
<td>8-22</td>
</tr>
<tr>
<td>Debugging and Notebook</td>
<td>8-22</td>
</tr>
<tr>
<td>Defining MATLAB Commands as Input Cells for Notebook</td>
<td>8-23</td>
</tr>
<tr>
<td>Defining Cell Groups for Notebook</td>
<td>8-23</td>
</tr>
<tr>
<td>Defining Autoinit Input Cells for Notebook</td>
<td>8-25</td>
</tr>
<tr>
<td>Defining Calc Zones for Notebook</td>
<td>8-25</td>
</tr>
<tr>
<td>Converting an Input Cell to Text with Notebook</td>
<td>8-26</td>
</tr>
<tr>
<td>Evaluating MATLAB Commands with Notebook</td>
<td>8-27</td>
</tr>
<tr>
<td>Evaluating Cell Groups with Notebook</td>
<td>8-28</td>
</tr>
<tr>
<td>Evaluating a Range of Input Cells with Notebook</td>
<td>8-29</td>
</tr>
<tr>
<td>Evaluating a Calc Zone with Notebook</td>
<td>8-30</td>
</tr>
<tr>
<td>Evaluating an Entire M-Book</td>
<td>8-30</td>
</tr>
<tr>
<td>Using a Loop to Evaluate Input Cells Repeatedly with Notebook</td>
<td>8-31</td>
</tr>
<tr>
<td>Converting Output Cells to Text with Notebook</td>
<td>8-32</td>
</tr>
<tr>
<td>Deleting Output Cells with Notebook</td>
<td>8-32</td>
</tr>
<tr>
<td>Printing and Formatting an M-Book</td>
<td>8-33</td>
</tr>
<tr>
<td>Printing an M-Book</td>
<td>8-33</td>
</tr>
<tr>
<td>Modifying Styles in the M-Book Template</td>
<td>8-33</td>
</tr>
<tr>
<td>Choosing Loose or Compact Format for Notebook</td>
<td>8-34</td>
</tr>
<tr>
<td>Controlling Numeric Output Format for Notebook</td>
<td>8-35</td>
</tr>
<tr>
<td>Controlling Graphic Output Format for Notebook</td>
<td>8-35</td>
</tr>
<tr>
<td>Configuring Notebook</td>
<td>8-39</td>
</tr>
<tr>
<td>Notebook Feature Reference</td>
<td>8-41</td>
</tr>
</tbody>
</table>
Source Control

Source Control Interface on PC Platforms .......... 9-2
Selecting and Viewing the Source Control System ....... 9-3
Adding Files to the Source Control System ............ 9-4
Checking Files Out of the Source Control System ....... 9-8
Checking Files Into the Source Control System ........ 9-11
Getting the Latest Version of Files from the Source
Control System ............................................ 9-15
Undoing .................................................. 9-18
Removing Files from the Source Control System ....... 9-19
Showing File History ....................................... 9-20
Comparing the Working Copy of a File to the Latest
Version in Source Control ................................. 9-22
Displaying Source Control Properties of a File .......... 9-24
Starting the Source Control System Client .......... 9-26

Source Control Interface on UNIX Platforms .......... 9-27
Selecting and Viewing the Source Control System ....... 9-27
Checking Files Into the Source Control System ....... 9-29
Checking Files Out of the Source Control System ....... 9-31
Undoing the Check-Out .................................... 9-32

Index
Startup and Shutdown

Starting MATLAB (p. 1-2)  General information about starting a MATLAB® session.
Starting MATLAB on Windows Platforms (p. 1-2)  Start MATLAB on Windows. Includes troubleshooting tips.
Starting MATLAB on UNIX Platforms (p. 1-3)  Start MATLAB on UNIX. Includes troubleshooting tips.
Startup Directory for MATLAB (p. 1-3)  View and change the startup directory for different platforms.
Startup Options (p. 1-4)  Instruct MATLAB to perform specified operations upon startup, including using a startup.m file.
Toolbox Path Caching in MATLAB (p. 1-10)  Reduce startup time if you run MATLAB from a network server.
Quitting MATLAB (p. 1-13)  End a MATLAB session. Instruct MATLAB to perform specified operations upon shutdown.
Starting MATLAB

Instructions for starting MATLAB depend on your platform. For a list of supported platforms, see the system requirements in the installation documentation, or the Products section of the MathWorks Web site, http://www.mathworks.com.

- “Starting MATLAB on Windows Platforms” on page 1-2
- “Starting MATLAB on UNIX Platforms” on page 1-3
- “Startup Directory for MATLAB” on page 1-3
- “Startup Options” on page 1-4
- “Toolbox Path Caching in MATLAB” on page 1-10

Starting MATLAB on Windows Platforms

To start MATLAB on a Microsoft Windows platform, select the Start -> Programs -> MATLAB 7.0 -> MATLAB 7.0, or double-click the MATLAB shortcut icon on your Windows desktop. The shortcut was automatically created by the installer.

If you start MATLAB from a DOS window, type `matlab` at the DOS prompt.

After starting MATLAB, the MATLAB desktop opens—see Chapter 2, “Desktop.” All of the desktop components that were open when you last shut down MATLAB will be opened on startup.

Starting MATLAB on UNIX Platforms

To start MATLAB on a UNIX platform, type `matlab` at the operating system prompt.

After starting MATLAB, the MATLAB desktop opens—see Chapter 2, “Desktop.” On UNIX platforms, if the `DISPLAY` environment variable is not set or is invalid, the desktop will not display. To troubleshoot other startup problems, see the Technical Support Web page for MATLAB Startup Solutions on the MathWorks Web site, http://www.mathworks.com/support/product/ML/solution/startup/page1.shtml.

Startup Directory for MATLAB

The startup directory is the current directory in MATLAB when it first starts, and depends on your platform and installation. You can specify a different startup directory.

Startup Directory on Windows Platforms

On Windows platforms, when you installed MATLAB, the default startup directory was set to `$matlabroot/work`, where `$matlabroot` is the directory where MATLAB files are installed.

Startup Directory on UNIX Platforms

On UNIX platforms, the initial current directory is the directory you are in on your UNIX file system when you start MATLAB.

Changing the Startup Directory

You can start MATLAB in a different directory from the default. The directory you specify will be the current working directory when MATLAB starts.

For Windows Platforms Only. To change the startup directory on Windows platforms:

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

   The **Properties** dialog box for `matlab.exe` opens to the **Shortcut** page.
2 Enter the new startup directory in the **Start in** field and click **OK**.

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.

**For All Platforms**. To change the startup directory:

1. Create a `startup.m` file—see “Using the Startup File for MATLAB, `startup.m`” on page 1-4.
2. In the `startup.m` file, include the `cd` function to change to the new directory.
3. Put the `startup.m` file in the current startup directory.

**Startup Options**

You can define startup options for MATLAB that instruct MATLAB to perform certain operations when you start it. There are two ways to specify startup options for MATLAB:

- “Using the Startup File for MATLAB, `startup.m`” on page 1-4
- “Adding Startup Options for Windows Platforms” on page 1-5 or “Adding Startup Options for UNIX Platforms” on page 1-8

**Using the Startup File for MATLAB, `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 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 Handle Graphics® defaults. Creating a `startup.m` file with the line

```matlab
addpath /home/me/mytools
cd /home/me/mytools
```

adds `/home/me/mytools` to your default search path and makes `mytools` the current directory upon startup.
**Location of startup.m.** Place the `startup.m` file in the current startup directory, which is where MATLAB first looks for it. For more information, see “Startup Directory for MATLAB” on page 1-3. You can instead place it in `$matlabroot/toolbox/local`, which is the next place MATLAB looks for `startup.m`, where `$matlabroot` is the directory in which MATLAB is installed.

**Adding Startup Options for Windows Platforms**

You can add selected startup options (also called command flags or command line switches) to the target path for your Windows shortcut for MATLAB. To do so:

1. Right-click the MATLAB shortcut icon

   ![Shortcut Icon]

   and select **Properties** from the context menu. The **Properties** dialog box for `matlab.exe` opens to the **Shortcut** panel.

2. In the **Target** field, after the target path for `matlab.exe`, add `/r results` to the end of the file path. This instructs MATLAB to run the results file automatically after startup. The statement in the **Target** field might appear as

   ```
   H:\Programs\matlab.exe /r results
   ```
3 Click OK.

The following table contains the available MATLAB startup options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/automation</td>
<td>Start MATLAB as an automation server, minimized and without the MATLAB splash screen. For more information, see “COM and DDE Support” in the External Interfaces documentation.</td>
</tr>
<tr>
<td>/c licensefile</td>
<td>Set LM_LICENSE_FILE to licensefile. It can have the form port@host.</td>
</tr>
<tr>
<td>/logfile logfile</td>
<td>Automatically write output from MATLAB to the specified log file.</td>
</tr>
<tr>
<td>/nosplash</td>
<td>Start MATLAB without displaying the MATLAB splash screen.</td>
</tr>
<tr>
<td>/r MATLAB_command</td>
<td>Automatically run the specified MATLAB command or M-file immediately after MATLAB starts. This is also referred to as calling MATLAB in batch mode. Separate multiple commands with commas or semicolons (;). See also “Notes About Startup Options” on page 1-7.</td>
</tr>
<tr>
<td>/regserver</td>
<td>Modify the Windows registry with the appropriate COM entries for MATLAB. For more information, see “COM and DDE Support” in the External Interfaces documentation.</td>
</tr>
<tr>
<td>/unregserver</td>
<td>Modify the Windows registry to remove the COM entries for MATLAB. Use this option to reset the registry. For more information, see “COM and DDE Support” in the External Interfaces documentation.</td>
</tr>
</tbody>
</table>
Notes About Startup Options.

• You can use a hyphen (-) instead of a slash (/), for example, -nosplash.
• When automatically running M-files or commands at startup with the /r option, all M-files or commands used must be on the MATLAB path or in the MATLAB startup directory. For example

    ... matlab /r myownfile

    runs myownfile, where myownfile is in the MATLAB startup directory. Use only the filename, not the file extension or pathname. For example, MATLAB produces an error when you run

    ... matlab /r C:\results.m.

• You can also enter MATLAB startup options with a DOS command by including the commands in quotation marks. For example, in the DOS window, run

    matlab /r "hf=figure;peaks;print(hf);exit"

    When MATLAB starts, it automatically creates a figure, runs peaks, prints the figure, and exits.
Adding Startup Options for UNIX Platforms
Include startup options (also called command flags or command line switches) after the `matlab` startup command.

For example, to start MATLAB without the splash screen, type

```
matlab -nosplash
```

For more details, see the `matlab` reference page.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-arch</code></td>
<td>Run MATLAB assuming architecture <code>arch</code>.</td>
</tr>
<tr>
<td><code>-arch/ext</code></td>
<td>Run the version of MATLAB with the extension <code>ext</code>, if it exists, assuming architecture <code>arch</code>.</td>
</tr>
<tr>
<td><code>-c licensefile</code></td>
<td>Set <code>LM_LICENSE_FILE</code> to <code>licensefile</code>. It can have the form <code>port@host</code>.</td>
</tr>
<tr>
<td><code>-check_malloc</code></td>
<td>Set the <code>MATLAB_MEM_MGR</code> environment variable to <code>debug</code>. This starts MATLAB memory integrity checking.</td>
</tr>
<tr>
<td><code>-D debugger [options]</code></td>
<td>Start MATLAB with the specified debugger.</td>
</tr>
<tr>
<td><code>-debug</code></td>
<td>Turn on MATLAB internal debugging.</td>
</tr>
<tr>
<td><code>-display Xserver</code></td>
<td>Send X commands to <code>Xserver</code>.</td>
</tr>
<tr>
<td><code>-ext</code></td>
<td>Run the version of MATLAB with the extension <code>ext</code>, if it exists.</td>
</tr>
<tr>
<td><code>-h or -help</code></td>
<td>Display startup options (without starting MATLAB).</td>
</tr>
<tr>
<td><code>-logfile log</code></td>
<td>Automatically write output from MATLAB to the specified log file.</td>
</tr>
<tr>
<td><code>-mwvisual visualid</code></td>
<td>Specify the default X visual to use for figure windows.</td>
</tr>
</tbody>
</table>
**Option** | **Description (Continued)**
--- | ---
-n | Display final values of environment variables and arguments passed to MATLAB (without starting MATLAB).
-nodesktop | Start MATLAB without bringing up the MATLAB desktop. Use this option to run without an X-window, for example, in VT100 mode, or in batch processing mode. Note that if you pipe to MATLAB using the > constructor, the nodesktop option is used automatically. With nodesktop, you can still use most development environment tools by starting them via a function. Specifically use:
- `commandhistory` to open the Command History
- `edit` to open the Editor/Debugger
- `helpbrowser` to open the Help browser
- `filebrowser` to open the Current Directory browser
- `workspace` to open the Workspace browser
- `openvar` to open the Array Editor
- `profile viewer` to open the Profiler
- `uiimport` to open the Import Wizard
Don’t use nodesktop to provide a command line interface. If you prefer a command line interface, select **Desktop -> Desktop Layout -> Command Window Only**.
-nodisplay | MATLAB ignores the `display` argument.
Toolbox Path Caching in MATLAB

For performance reasons, MATLAB 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.

Startup Using Cache File

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

How the Toolbox Path Cache Works. MATLAB caches (essentially, stores in a known files list) the names and locations of files in \texttt{$\matlabroot/toolbox$} directories. These directories are for MathWorks supplied files 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 MathWorks-supplied M-files that you edit in a directory that is \textit{not} in the \texttt{$\matlabroot/toolbox$} directory tree. If you keep your files in \texttt{$\matlabroot/toolbox$} directories, they may be overwritten when you install a new version of MATLAB.

When to Update the Cache. When you add files to \texttt{$\matlabroot/toolbox$} 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 MATLAB installer. 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 \texttt{$\matlabroot/toolbox$} directories.

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

- Save new files in \texttt{$\matlabroot/toolbox$} directories using an external editor
- Use operating system features and commands to add or remove files in \texttt{$\matlabroot/toolbox$} directories

MATLAB displays this message

\begin{verbatim}
Undefined function or variable
\end{verbatim}

You need to update the cache so MATLAB will recognize the changes you made in \texttt{$\matlabroot/toolbox$} directories.

Steps to Update the Cache. To update the cache and the cache file

1. Select \texttt{File} $\rightarrow$ \texttt{Preferences} $\rightarrow$ \texttt{General}.

   The \texttt{General Preferences} panel is displayed.
2 Click **Update Toolbox Path Cache** and click **OK**.

![Image of Preferences window with General Preferences selected]

**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**.
Quitting MATLAB

To quit MATLAB 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.

Unless unsaved files are open, MATLAB closes immediately. If you want to see a warning that allows you to confirm quitting, use the finishdlg.m script, as described in the next paragraph.

Running a Script When Quitting MATLAB

When MATLAB quits, it runs the script finish.m, if finish.m exists in the current directory or anywhere on the MATLAB search path. You create the file finish.m. It contains functions 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 finish.
Desktop

The easiest way to learn to use the desktop is just by working with it. Refer to this information if you have problems or questions.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview of the Desktop (p. 2-2)</td>
<td>Basic summary of the desktop and its tools.</td>
</tr>
<tr>
<td>Arranging the Desktop—Overview (p. 2-5)</td>
<td>Open and arrange desktop tools and documents to suit your needs. Scan the examples and follow the instructions to arrange your desktop.</td>
</tr>
<tr>
<td>Common Desktop Features (p. 2-19)</td>
<td>Details regarding the <strong>Start</strong> button, MATLAB shortcuts, toolbars, menus and context menus, status bar, and keyboard shortcuts and accelerators. Also includes selecting multiple items, cut, copy, and paste, using page setup for printing, and accessing the MathWorks Web site from MATLAB.</td>
</tr>
<tr>
<td>Fonts, Colors, and Other Preferences (p. 2-44)</td>
<td>Specify options for desktop tools, including fonts and colors.</td>
</tr>
</tbody>
</table>
Overview of the Desktop

When you start MATLAB, the first thing you see is the MATLAB desktop, consisting of tools (GUIs or graphical user interfaces) 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. You can change the desktop arrangement to meet your needs, including resizing, moving, and closing tools. For details, see “Arranging the Desktop—Overview” on page 2-5.

The Editor/Debugger and Array Editor support multiple document windows within them, and similarly, you can group multiple figure windows together. For information about working with documents in the desktop, see “Opening and Arranging Documents” on page 2-7 for more information.

If you are using the Help browser, watch the Desktop and Command Window video demo for an overview of the major functionality.
Overview of the Desktop

Example of Desktop—Default Layout

Menus change, depending on the tool you are currently using.

Use tab to go to Workspace browser.

View or change current directory.

Move Command Window outside of desktop (undock).

Click for quick access to tools and more.

View or execute previously run functions from the Command History.

Drag the separator bar to resize windows.

Enter MATLAB functions at command line prompt.

Menu bar

Shortcut bar

Current Directory

Workspace browser

Command History

Command Window

File Edit Debug Desktop Window Help

Copyright 1984-2004 The MathWorks, Inc.

Version 7.0.0.333 (R14)

To get started, select MATLAB Help or Draw from the

>>
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 result as using features in desktop tools. You must use these equivalent functions to perform the operations in M-files. Instructions for using equivalent functions are provided with the documentation for each tool.

<table>
<thead>
<tr>
<th>Desktop Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array Editor</td>
<td>View array contents in a table format and edit the values.</td>
</tr>
<tr>
<td>Command Window</td>
<td>Run MATLAB functions.</td>
</tr>
<tr>
<td>Command History</td>
<td>View a log of the functions you entered in the Command Window, copy them, execute them, and more.</td>
</tr>
<tr>
<td>Current Directory Browser</td>
<td>View files, perform file operations such as open, find files and file content, and manage and tune your files.</td>
</tr>
<tr>
<td>Editor/Debugger</td>
<td>Create, edit, and debug M-files (files containing MATLAB functions).</td>
</tr>
<tr>
<td>Figures</td>
<td>Create, modify, view, and print MATLAB figures.</td>
</tr>
<tr>
<td>Help Browser</td>
<td>View and search the documentation for all your MathWorks products.</td>
</tr>
<tr>
<td>Profiler</td>
<td>Improve the performance of your M-files using this graphical interface.</td>
</tr>
<tr>
<td>Start Button</td>
<td>Run tools and access documentation for all of your MathWorks products, and create and use MATLAB shortcuts.</td>
</tr>
<tr>
<td>Web Browser</td>
<td>View HTML and related information produced by MATLAB.</td>
</tr>
<tr>
<td>Workspace Browser</td>
<td>View and make changes to the contents of the workspace.</td>
</tr>
</tbody>
</table>
You can modify the desktop configuration to best meet your needs. Because the desktop uses many standard user interface conventions, it is easy to learn about arranging the desktop just by using it. If you are not familiar with any of the interface elements, refer to the overview information and examples in this section.

The desktop manages tools and documents differently. The Command History and Editor/Debugger are examples of tools, and an M-file is an example of a document that appears in the Editor/Debugger tool.

These are the main actions you perform in arranging your desktop tools and documents:

- “Opening and Arranging Tools” on page 2-6
- “Opening and Arranging Documents” on page 2-7
- “Saving Desktop Layouts” on page 2-18

See also “Examples of Desktop Arrangements” on page 2-11.
Opening and Arranging Tools

This table summarizes actions for arranging desktop tools. For further information, click the “See more details online” links, which provide additional information, available only online.

<table>
<thead>
<tr>
<th>Tool Action</th>
<th>Steps to Perform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opening Desktop Tools</td>
<td>To maximize your work area, keep open only those tools you use. To open a tool, select the tool name from the Desktop menu. Opened tools have a check mark before them in the menu. The tool appears in the location it occupied the last time it was open. The sizes of other tools adjust to accommodate the newly opened tool. See more details online. Documents and the tools they are associated with can be part of the desktop, but you do not open them via the Desktop menu. Instead, you open a document and it opens the associated tool. See “Opening and Arranging Documents” on page 2-7.</td>
</tr>
<tr>
<td>Navigating Among Desktop Tools</td>
<td>The Window menu displays all open desktop tools and documents, as well as tools for other MathWorks products. Select an entry in the Window menu to go directly to that tool or document. Another way to access an undocked desktop tool is by selecting its entry in the Windows task bar, or the equivalent for your platform. See more details online.</td>
</tr>
<tr>
<td>Closing Desktop Tools</td>
<td>To close a desktop tool, select the item in the Desktop menu, which clears the check mark in the menu and closes the tool. Or click the close box (X) in the title bar for the tool. See more details online.</td>
</tr>
<tr>
<td>Resizing Tools</td>
<td>To resize tools in the MATLAB desktop, drag the separator bar, which is the bar between two tools. You can hide the title bars for tools in the desktop so the tools use less space—select Desktop -&gt; Titles. See more details online.</td>
</tr>
<tr>
<td>Moving Tools Within the Desktop</td>
<td>To move a tool in the MATLAB desktop, drag the title bar of the tool toward where you want the tool to be located. As you drag the tool, an outline of it appears. When the outline nears a position where you can keep it, the outline snaps to that location. Release the mouse button. The tool stays at the new location. Other tools in the desktop resize to accommodate the new configuration. The inside edges of the desktop container and tools all act as if they are “sticky,” so you can position a tool along any inside edge. See more details online.</td>
</tr>
</tbody>
</table>
Opening and Arranging Documents

Open a document, such as an M-file or a variable, and it opens in its tool, for example, the Editor/Debugger or Array Editor. The following illustration shows a desktop arrangement that includes Editor/Debugger and Array Editor documents. See instructions in “Summary of Actions for Arranging Documents” on page 2-9.
Example of Documents in the Desktop

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

- Use tabs to go to open tools. Use the document bar to go to open documents.
- Use the Window menu or toolbar icons 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 or displaying the unsaved document dialog box.

See also “Examples of Desktop Arrangements” on page 2-11.
Summary of Actions for Arranging Documents

This table summarizes actions for arranging documents in their tool. For further information, click the “See more details online” links, which provide additional information, available only online.

<table>
<thead>
<tr>
<th>Document Action</th>
<th>Overview</th>
</tr>
</thead>
</table>
| Opening Documents | When you open a MATLAB document, it opens in the associated tool. If the tool is not already open, it opens when you open the document, and appears in the position it occupied when last used. Figures and Web browsers open undocked, regardless of the last position occupied. How to open a document depends on the document type:  
  * M-file: Select File -> Open and select the M-file. It opens in the Editor/Debugger.  
  * Workspace variable: In the Workspace browser, double-click the variable. It opens in the Array Editor.  
  * HTML document: In the Current Directory browser, double-click the file. It opens in the Web browser.  
  * Figure: Type plot or use another graphics function. The plot appears in a figure window.  

There are many additional ways to open documents. See more details online. |
<p>| Navigating Among Documents—The document bar | When more than one document is open within a tool, each document is either maximized (the default), or arranged so that multiple documents are visible at once. Click a document that is in view to make it the active document. Use the document bar to go to a document that is not in view. The names of all open documents appear in the document bar. Select a document name in the document bar to make that document active. To show the document bar if it is not open, select Desktop -&gt; Document Bar, and select a position for its location, for example, Right. See more details online. Entries for undocked documents appear in the Windows task bar, or the equivalent for your platform. Click the task bar entry for a document to make that document active. |</p>
<table>
<thead>
<tr>
<th>Document Action</th>
<th>Overview (Continued)</th>
</tr>
</thead>
</table>
| Positioning, Moving, and Resizing Documents | To position open documents within their tool, select an arrangement from the Window menu when the tool is active, or by using an icon on the toolbar for Maximize, Float, Left/Right Split, Top/Bottom Split, and Tile. On the Macintosh platform, the tile option is not available in the Window menu so use the tile icon instead.  
With the split and tile arrangements, you refine the document position by moving the cursor over the handle (●) on the separator bar. A close box then appears. When you click the close box between two open documents, both documents stay open, but one moves over the top of the other. When you click the close box between a document and an empty tile, the empty tile closes. To move a document in a tiled or split arrangement, drag the title bar of a document to another tile. If the documents use the tiled or split arrangement, drag the separator bar that is between the documents to resize them.  
To move or resize maximized documents, you move or resize the tool. Or, right-click a document name in the document bar to move it. See more details online. |
| Closing Documents        | To close a document, click the close box in that document’s title bar. After closing all the documents in a tool, the tool remains open with no documents in it. If you select the close box for the tool, all documents in that tool close. Upon closing an M-file with unsaved changes, a prompt appears asking if you want to save the document. To close the document without saving changes and without seeing the prompt, use Ctrl with the document’s close box. See more details online. |
| Moving Documents and Tools Out of the Desktop (Undocking) | To undock all documents in a tool from the desktop, click the undock button ⌘ in the tool’s title bar. The tool and its documents move outside of the desktop. See more details online.  
To undock a document from its tool, click the undock button ⌘ for the document. The undock button is either in the document’s title bar, menu bar, or toolbar, depending on the document type and whether or not the document is within the desktop or is in its tool outside of the desktop.  
Undocked tools and documents have entries in the Windows task bar (or the equivalent for your platform) and each document type has a unique icon. |
Examples of Desktop Arrangements

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 and instructions present just one way. Depending on how your desktop looks before you start, the instructions might not apply exactly.

- “Tool Outside of Desktop and Other Tools Tabbed Inside Desktop Example” on page 2-12
- “Tiled Documents in Desktop Example” on page 2-13
- “Maximized Documents Outside of the Desktop Example” on page 2-15
- “No Empty Tiles in Desktop Example” on page 2-14
- “Floating (Cascaded) Figures in Desktop Example” on page 2-16
- “Undocked Tools and Documents Example” on page 2-17
Tool Outside of Desktop and Other Tools Tabbed Inside Desktop Example

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

The Command Window is outside of the desktop. To achieve this, click the undock arrow in the tool's title bar when the tool is in the desktop.

You can group tools together inside the desktop and access them via labeled tabs. Here the Current Directory browser, Workspace browser, and Command History are tabbed together. To achieve this, drag the title bar of one tool on top of the title bar of the tool(s) you want to group it with.
Tiled Documents in Desktop Example

When you open a document (for example, an M-file), it also opens the tool (for example, the Editor). You can dock the tool in the desktop, as shown here. If tools or documents are outside the desktop, to move them inside, click the dock arrow in the tool and in any separate M-file’s menu bars, or use Desktop -> Dock menu items. Select Left/Right Split from the Window menu or use the toolbar icon to show two M-files side-by-side.

You can hide a toolbar. Here, the shortcuts toolbar is hidden. Select Desktop -> Toolbar name to hide (or show) a toolbar. To see or move the document bar, select Desktop -> Document Bar, and choose its location, for example, Top.

The shortcuts toolbar is hidden. The document bar is at the top of the Editor/Debugger.
No Empty Tiles in Desktop Example

You can hide a document under another—drag a document’s title bar on top of another document. The document on top fully covers the document underneath. This gives more space to the active document. To see hidden documents, use the Window menu or document bar.

To show two documents at once use a split arrangement. To see more than two documents at once, select the tile icon and move the cursor across the grid menu to select the number of tiles you want. The grid in this example has four tiles, but there are only three documents open. (The empty tile will be shown in gray in the menu.) You can move a document to any empty tile by dragging its title bar to the new location. To close empty tile, position the cursor over the handle on the separator bar. It becomes a close box, as shown here, which you click to close the empty tile. After clicking the close box, the empty tile closes and the neighboring document expands as shown here. Similarly, click the close box between two tiles containing documents and one becomes hidden.

![Before and After](image)
Maximized Documents Outside of the Desktop Example

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 document icon, or select **Window -> Maximize**.
- Display the cell toolbar—select **Cell -> Enable Cell Mode**.
- Access any document using the document bar in the Editor. To show the document bar on the left side of the Editor, select **Desktop -> Document Bar -> Left** from the Editor.
Floating (Cascaded) Figures in Desktop Example

You can show multiple figures at once in the desktop. By default, figures open outside the desktop. Click the dock button in each figure’s menu bar to move the figures into the desktop.

You can float (also called cascade) the figures by selecting Window -> Float, or clicking the float icon ⬇️. This arrangement is not available on the Macintosh platform. To get even more screen area for the figures, hide the document bar as shown in this example—select Desktop -> Document Bar -> Hide.
Undocked Tools and Documents Example

You can use tools and documents outside of the desktop. One way to achieve this is to first undock the tool from the desktop by selecting Desktop -> Undock Toolname. Then undock documents from the undocked tool by selecting Desktop -> Undock Documentname from the tool. If you undock all documents from a tool, an “empty” tool window remains.

In this example, one of the Editor documents, povertystats.m, includes the name of the tool with it and the other Editor document, collatz.m, does not. Contrast this with the Array Editor documents, where neither document window includes the name of the tool. This is because when documents are undocked from both the desktop and their tool, you can close the tool but the tool’s undocked documents remain open. If you closed the Editor, the collatz.m document would remain open. To close all undocked documents and their tools at once, select Window -> Close All Documents from an undocked document window.
Saving Desktop Layouts

When you end a session, MATLAB saves the desktop layout. The next time you start MATLAB, the desktop is restored the way you left it.

To use a predefined layout, select Desktop -> Desktop Layout, and choose a configuration. See “Predefined Layouts” in the online documentation for more information.

To save your own layouts for later reuse, select Desktop -> Save Layout, and provide a name. Reuse the layout by selecting the name from Desktop -> Desktop Layout. See “Saving Your Own Desktop Layouts” in the online documentation for more information.
Common Desktop Features

This section presents useful details about common features of desktop tools:

- “Start Button for Accessing Tools” on page 2-19
- “Shortcuts for MATLAB—Easily Run a Group of Statements” on page 2-21
- “Web Browser” on page 2-29
- “Menus and Context Menus” on page 2-31
- “Toolbars” on page 2-32
- “Status Bar” on page 2-34
- “Sizing, Arranging, and Sorting Columns in Tools” on page 2-34
- “Keyboard Shortcuts” on page 2-35
- “Selecting Multiple Items” on page 2-38
- “Cut, Copy, and Paste” on page 2-39
- “Page Setup Options for Printing” on page 2-40
- “Accessing The MathWorks on the Web” on page 2-42

Start Button for Accessing Tools

The MATLAB Start button provides easy access to tools, demos, and documentation for all your MathWorks products. You can also create and run MATLAB shortcuts from it, where a shortcut is a group of MATLAB statements.

Using 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 to view the Start button contents.
From the menu and submenu items, select an item to open it. Use the icons to quickly locate a type of product or tool—see the following description of icons.

For example, select `Start -> MATLAB -> GUIDE (GUI Builder)` to open that tool.
Icons in the Start Button. Icons help you quickly locate a particular type of product or tool. This legend describes the action performed when you select an entry with one of these icons in the Start button.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description of Action When Opened</th>
</tr>
</thead>
<tbody>
<tr>
<td>📚</td>
<td>Documentation for that product opens in the Help browser.</td>
</tr>
<tr>
<td>🔌</td>
<td>Demos for the product are listed in the Help browser Demos tab.</td>
</tr>
<tr>
<td>🛠️</td>
<td>Selected tool opens.</td>
</tr>
<tr>
<td>📗</td>
<td>Block library opens.</td>
</tr>
<tr>
<td>📖</td>
<td>Document opens in your system Web browser.</td>
</tr>
</tbody>
</table>

Customizing the Start Button
You can add your own toolboxes to the Start button. Select Start -> Desktop Tools -> View Source Files to open the Start Button Configuration Files dialog box. For details, click the Help button in the dialog box, which displays “Adding Your Own Toolboxes to the Development Environment”, a topic in the online documentation.

Shortcuts for MATLAB—Easily Run a Group of Statements
A MATLAB shortcut is an easy way to run a group of MATLAB functions that you use regularly. These topics are covered:

- “What Is a Shortcut?” on page 2-22
- “Examples of Useful Shortcuts” on page 2-22
- “Creating Shortcuts” on page 2-23
- “Running Shortcuts” on page 2-25
- “Shortcuts Toolbar” on page 2-25
- “Organizing and Editing Shortcuts” on page 2-28
What Is a Shortcut?
A MATLAB shortcut is an easy way to run a group of MATLAB statements. First you create a shortcut that contains all of 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, but unlike an M-file, a shortcut does not have to be on the MATLAB search path or in the current directory when you run it. In addition, you conveniently run the shortcut by selecting it from the Start button or desktop Shortcuts toolbar, which are readily accessible.

Although shortcuts run MATLAB statements, they are not M-files and are not stored 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. An example of this is setting 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. Some users create a shortcut for even a single function they use frequently, such as clc to clear the screen.
• 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 do not easily remember but need to use.
Creating Shortcuts

This is an example of a shortcut you might create for a project you work on, the Sea Temperature project. When you work on that project you like to set up your environment in a certain way by running a series of 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 of 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, as shown in the following illustration.

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

   b. Put the environment setup 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 key bindings preferences you specified for the Editor and the Desktop code font preferences you specified. Note 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.
MATLAB adds the shortcut to the **Shortcuts** entry on 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 can also 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 can also use these methods to create shortcuts:

- Add shortcuts to and run them from the desktop Shortcuts toolbar. See “Shortcuts Toolbar” on page 2-25.
- From the Command History, create a shortcut by selecting statements, right-clicking, and selecting Create Shortcut from the context menu. By default, shortcuts created from the Command History are assigned to the Toolbar 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 can also access shortcuts that have the Help Browser Favorites category 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 of the statements in the shortcut Callback field execute. It is as if you ran those statements from the Command Window, although they are not reflected in the Command History.

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

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

1 Select statements from the Command History, 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 keep so the shortcut appears 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 can be displayed at once, use the drop-down list to access all of them.
Click the icon on the Shortcuts toolbar to run the shortcut. You can also run the shortcut from the Start button by selecting it in the Toolbar Shortcuts category.

You can also 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 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. 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, move them to a different category. For instructions, see “Organizing and Editing Shortcuts” on page 2-28.

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 select **Show Labels** again, which selects 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. Select **Shortcuts -> Organize Shortcuts** from the **Start** button. Access it via the shortcuts toolbar context menu.

   The **Shortcuts Organizer** dialog box appears.

   Move shortcuts and more. For example, drag a shortcut to another category.
2 Use the buttons in the dialog box to edit and organize shortcuts and categories. You can also right-click an item and select an action from the context menu.

Changes take effect immediately.

3 Click Close.

MATLAB maintains the shortcuts in the shortcuts.xml file. To see the directory where the file is stored, run `prefdir`.

For more information on what you can do in the Shortcuts Organizer dialog box, click the Help button.

**Web Browser**

Some tools in MATLAB and related products display HTML or XML documents in the MATLAB Web browser. For example, after using the Editor's cell features to publish an M-file to HTML, you view the HTML file in the MATLAB Web browser. Because the MATLAB Web browser is a desktop tool, you can dock it and perform other desktop operations on it.
To display documents in the Web browser, use the `web` function. The `web` function supports arguments that allow you to display documents in your system browser, for example, Netscape, or in the Help browser.

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 “View Documentation in the Help Browser” on page 4-19.

One feature of the Web browser not found in the Help browser is the **Location** field. In the Web browser, type a URL in the field to display that Web page.

Like any 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.
Internet Connection and Fonts for Web Browser—Web Preferences

To specify a proxy server to connect from the MATLAB Web browser to the Internet, use Web preferences. You might need to specify this preference if you have a firewall, for example. If you have a firewall and do not specify the proxy settings, links from the Web browser to URLs will not work.

Select File -> Preferences -> Web. By default, the check box Use a proxy server to connect to the Internet is not selected. This means you have a direct connection to the Internet.

To specify a proxy server, select the check box and specify the Proxy host and Proxy port. See your system administrator for the information you need to specify the proxy settings. As an example, 172.16.10.8 illustrates the format for host, and 3128 is the type of value you enter for port.

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

Menus and Context Menus

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

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.
Toolbars

The toolbar in the desktop provides easy access to frequently used operations. Position the cursor over a button for a second or two and a tooltip appears that describes the item.
Some tools also 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.

To hide a toolbar, or to show it again after hiding it, use the appropriate toolbar item in the Desktop menu. For figure windows, use the toolbar item in its View menu.

**Current Directory Field**

The current directory field in the desktop toolbar shows the MATLAB current working directory. You can change the current directory using this field, using any of these methods:

- Type the current directory you want to change to directly in the field.
- Use the drop-down list to select a previously used current directory. To specify the number of entries maintained each session, use the History preference you access via File -> Preferences -> Current Directory.
- Use the browse button ... to select a new current directory.
- Use the up button \( \uparrow \) to move the current directory up one level.

The same current directory field also appears in the Current Directory browser when the Current Directory browser is undocked from the desktop. Use the Current Directory browser to perform many additional file operations. For more information, see “File Management Operations” on page 5-29.
Status Bar
Along the bottom of the desktop is the status bar. It displays messages, such as when MATLAB is busy executing statements or when the Profiler is on. Some tools, such as the Editor/Debugger, display additional status information, such as the current line number. Not all status information appears on the status bar—many MATLAB functions and tools provide status information that is not reported to the status bar.

You can construct your own functions to provide status information. See the timer function, and search for other specific terms describing the status of interest.

Sizing, Arranging, and Sorting Columns in Tools
Some desktop tools present information in columns, such as the Current Directory browser.

To change the column width, drag the separator bar between two column headings in a tool. When a column is too narrow to show all the information in it, position the cursor over an item and the full value for that item displays like a tooltip.

To rearrange the columns in a tool, drag the column header to a different position. To 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. Some columns also allow you to reverse the sort order by clicking the column header again. A small gray arrow in the header indicates the current sort order—for example, an up arrow in the Last Modified Date column header indicates an ascending sort order, meaning the oldest files are at the top of the list.
Keyboard Shortcuts

You can access many of the menu items using shortcut keys (sometimes called accelerators) for your platform. For example, use Ctrl+X to perform a cut on 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.

The keyboard shortcuts for the Command Window and Editor/Debugger also depend on settings for key bindings that you specify using preferences. For details, see “Command Line Key Bindings” on page 3-29, and “Arrow and Control Keys (Shortcuts) in the Editor” on page 6-21. Instructions in the documentation specify shortcuts using the Windows Ctrl convention, but with Macintosh key bindings selected, you can use the Command key instead.
On Windows platforms, you can also use mnemonics, such as Alt+F to open the File menu. Mnemonics are listed with the menu item. For example, on the File menu the F in File is underlined, which indicates that Alt+F opens the menu. Note that some versions of Windows 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 in order to see the shortcuts on the menus. In Windows 2000, go to the Display Control Panel, select Effects, and clear the item Hide keyboard navigation indicators until I use the Alt key. See your Windows documentation for details.

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>The equivalent of double-clicking, it performs the default action for a selection. For example, press Enter while a statement in the Command History is selected to run that statement in the Command Window. For buttons in tools and dialog boxes, 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 2-38 for an illustration.</td>
</tr>
<tr>
<td>Escape</td>
<td>Cancels the current action. For example, if you select the Edit menu, the menu items display. Pressing Escape retracts the menu items. Pressing Escape in a dialog box is the same as selecting the Cancel button.</td>
</tr>
<tr>
<td>Ctrl+Tab</td>
<td>Moves to the next open tool in the desktop, or to the next open group of tools tabbed together.</td>
</tr>
<tr>
<td>Tab</td>
<td>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.</td>
</tr>
<tr>
<td>Key</td>
<td>Result (Continued)</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Alt+S</td>
<td>Displays <strong>Start</strong> button menu (except on Macintosh platforms).</td>
</tr>
<tr>
<td>Alt+Y</td>
<td>Provides access to current directory field in toolbar.</td>
</tr>
<tr>
<td>Space bar</td>
<td>For buttons in tools and dialog boxes, activates the active button. See “Default Button and Active Button (Button with Focus)” on page 2-38 for an illustration of selecting default and active buttons using keys.</td>
</tr>
<tr>
<td>+ or - or *</td>
<td>Use these keys on the numeric keypad to expand and collapse items in tree views. The Help browser <strong>Navigator</strong> pane and the Command History 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.</td>
</tr>
<tr>
<td>Ctrl+Shift+Tab</td>
<td>Moves to the previous open tool or group of tabbed tools in the desktop.</td>
</tr>
<tr>
<td>Ctrl+Page Down</td>
<td>Moves to the next tool within a group of tools. In a group of documents, moves to next document.</td>
</tr>
<tr>
<td>Ctrl+Page Up</td>
<td>Moves to the previous tool within a group of tools tabbed together. In a group of documents, moves to previous document.</td>
</tr>
<tr>
<td>Ctrl+F6</td>
<td>Moves to the next tool or document (only for Windows and Solaris platforms).</td>
</tr>
<tr>
<td>Ctrl+Shift+F6</td>
<td>Moves to the next tool or document (only for Windows and Solaris platforms).</td>
</tr>
<tr>
<td>Alt+F4</td>
<td>Closes the desktop, thereby quitting MATLAB, or outside the desktop, closes the active window.</td>
</tr>
</tbody>
</table>

For additional shortcuts available in the various desktop tools, see the documentation for each tool. For example, in the Command Window, see “Keyboard Shortcuts in the Command Window” on page 3-16.
Go To First Letter Feature in Desktop Tool Lists
In the Current Directory browser and Command History, you can type a letter to move directly to the next item in the list that starts with the letter you typed.

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.

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 your platform is Windows, 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.
Cut, Copy, and Paste

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. Use the Edit menu, toolbar, context menus, or standard keyboard shortcuts. For example, you can copy a selection of statements from the Command History and paste them into the desktop.

Use Paste to move items copied to the clipboard from other applications. The Paste Special 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 Excel, into MATLAB. For details, see the Import Wizard documentation.

When editing text in the Command Window and Editor, select text and drag it to move the text to a new location. Press Ctrl and drag the selected text to copy it to a 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.

Drag and Drop

You can also move or copy a selection from one tool to another by dragging the selection. For example, make a selection in the Command History 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 drag a filename from the Current Directory browser to the Editor to open that file in 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 Microsoft Word document into the Editor cuts and pastes it into the open file. Dragging an M-file from Windows Explorer to the Command Window runs the file.
Page Setup Options for Printing

You can specify page setup options to be used when you print from the Command History, Workspace browser, and Array Editor. Select File -> Page Setup. A standard system page setup dialog box 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

- “Specifying Page Setup Options” on page 2-40
- “Layout Options for Page Setup” on page 2-41
- “Header Options for Page Setup” on page 2-41
- “Fonts Options for Page Setup” on page 2-42

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** tab.
- **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 are highlighted in the Command Window and in print; output is not highlighted.

**Header Options for Page Setup**

If you want to print a header, select the **Layout** tab and on it, select **Print header**. Then 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.

Accessing The MathWorks on the Web
You can access popular MathWorks Web pages from the MATLAB desktop. Select one of the following items from the Help -> Web Resources menu. For most items, the selected Web page then opens in your default system Web browser, for example, Netscape:

- **Products**—MathWorks Products page (http://www.mathworks.com/products/), with information about the full family of products.
- **Membership**—Access Login page (http://www.mathworks.com/accesslogin/) for Access Login members. If you are not a member, you can join online. Membership helps you stay up to date on the latest MATLAB developments.
- **Technical Support Knowledge Base**—MathWorks Support page (http://www.mathworks.com/support), where you can look for solutions to problems you are having, or report new problems.
• **MATLAB Central**—MATLAB Central Web site (http://www.mathworks.com/matlabcentral/) for the MATLAB user community. It includes MATLAB contest entries and results, a MATLAB screen saver, and these technical resources:

  - **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 MATLAB users. 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.

**Check for Updates**

This feature allows you to easily determine if more recent versions of your MathWorks products are available. Select **Help -> Check for Updates**. A dialog box appears, listing the version numbers of all MathWorks products installed on your system. Click **Check for Updates** in the dialog box, which accesses the MathWorks Web site and reports back for each product if a newer version is available or if your version is the latest.
Fonts, Colors, and Other Preferences

Use preferences to change the font characteristics and the text and background colors for tools in the desktop.

- “Fonts Preferences for Desktop Tools” on page 2-44
- “Colors Preferences for Desktop Tools” on page 2-50
- “General Preferences for MATLAB” on page 2-54

Each tool also has its own set of preferences. For details, see “About Preferences” on page 2-56.

Fonts Preferences for Desktop Tools

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

- Type, for example, SansSerif
- Style, for example, bold
- Size in points, for example, 12 point

Select File -> Preferences -> Fonts to set fonts for desktop tools. You can set some font options differently for printing—see “Page Setup Options for Printing” on page 2-40.
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. Many users prefer that code display in a monospace font to provide better alignment, and prefer a more narrow font style for text information. With the desktop code font preference, you set just one preference to apply a monospace style to all tools that display code. Similarly, you can set just one preference to apply a text font to all desktop tools that display text.
The following illustrations show how the Editor would look using a monospace font or a proportional font. Note that a monospaced font is useful when you care about alignment, but a proportional font might use less space.

With a monospace font, all characters are the same width. Here, the font is 10 pt. Monospaced. 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. SansSerif. Each line contains 10 characters but none ends at the same column. The Editor's right-hand text limit is not relevant.

**Default Font Settings.** Default settings are listed in the following table. Note that Lucida Console approximates the `fixedsys` font available in earlier versions of MATLAB.
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 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.

See Also
“About Preferences” on page 2-56.

Custom Fonts Preferences
You can specify that a tool use the code font, the text font, or a different font. Select File -> Preferences -> Fonts. Click + and select Custom. The Fonts Custom Preferences panel appears.
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 it uses, code or text, appears under Font to Use. In the illustration shown, the Command Window uses the Desktop code font, which is defined in the Fonts panel as described in the previous section.

To change the font type the selected tool uses, select a different radio button. For Custom, you then specify the font characteristics for that tool.
Changing the Font—Example

This example changes the default settings (p. 2-20) for the desktop code font, changes the Command History preference so that it uses the desktop text font instead of code, and specifies a custom font for the Current Directory browser:

1. Change the characteristics for the desktop code font. On the Fonts pane, set the Desktop code font to Times New Roman, Plain, 14 point. Use the default for the Desktop text font, SansSerif, Plain, 10 point. Click Apply.

2. Make the Command History use the desktop text font. Select Fonts, click +, select Custom, and then select Command History from Desktop tools. Select the Desktop text radio button.

3. Apply a custom font to the Current Directory browser. Select Current Directory from Desktop tools. Select the Custom radio button. Select Arial Narrow and Plain, and type 11 in the size field. Click OK.

The following table details the results of the changes.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Font Type</th>
<th>Font Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Window</td>
<td>Desktop code</td>
<td>Times New Roman, Plain, 14 point</td>
</tr>
<tr>
<td>Command History</td>
<td>Desktop text</td>
<td>SansSerif, Plain, 10 point</td>
</tr>
<tr>
<td>Editor</td>
<td>Desktop code</td>
<td>Times New Roman, Plain, 14 point</td>
</tr>
<tr>
<td>Help Navigator</td>
<td>Desktop text</td>
<td>SansSerif, Plain, 10 point</td>
</tr>
<tr>
<td>HTML Proportional Text</td>
<td>Desktop text</td>
<td>SansSerif, Plain, 10 point</td>
</tr>
<tr>
<td>Current Directory</td>
<td>Custom</td>
<td>Arial Narrow, Plain, 11 point</td>
</tr>
<tr>
<td>Workspace</td>
<td>Desktop text</td>
<td>SansSerif, Plain, 10 point</td>
</tr>
<tr>
<td>Array Editor</td>
<td>Desktop text</td>
<td>SansSerif, Plain, 10 point</td>
</tr>
</tbody>
</table>

See Also. For help about how MATLAB stores preferences and help for other preferences, see “About Preferences” on page 2-56.
Colors Preferences for 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 “Page Setup Options for Printing” on page 2-40.

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

The Sample area shows how the changes will look.
Desktop Tool Colors

Use desktop tool colors to change the color of the text and background in the desktop tools. 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, 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.
Syntax Highlighting Colors

In the Command Window, Command History, Editor/Debugger, 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.

In the Command Window, only input you type is highlighted; output from running MATLAB functions is not highlighted. In the Editor/Debugger, you can specify syntax highlighting for use with files in C/C++, Java, and HTML. For details, see the topic “Language Preferences” in the online documentation.

Use preferences to specify the syntax highlighting colors. When you choose a color, 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 (\ldots{}), 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 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 the `!` (shell escape) are colored gold.

- **Errors**—Error text is colored red.

Click **Restore Default Colors** to return to the default settings for syntax highlighting colors as well as desktop tool colors.
See Also
For help about how MATLAB stores preferences and help for other preferences, see “About Preferences” on page 2-56.

General Preferences for MATLAB
These preferences apply to all relevant tools in MATLAB.

![Preferences dialog box](image)

Toolbox Path Caching Preference
See “Toolbox Path Caching in MATLAB” on page 1-10.
Figure Window Printing
Select an option regarding color printing for figures. For more information on printing figures, see the topic “Printing Graphics” in the online documentation.

Default Behavior of the Delete Function
Files you delete using the delete function are permanently deleted by default. There is no opportunity to retrieve them.

You can use this preference to instead move deleted files to the Recycle Bin on Windows, to the Trash Can on Macintosh, or to a tmp directory on UNIX 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 Windows. When you select this preference, delete might run slower.

Function alternative. This MATLAB preference 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

```matlab
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 during the current session, when you next 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 or Deleting Files and Directories” on page 5-35.

See Also
For help about how MATLAB stores preferences and help for other preferences, see “About Preferences” on page 2-56.
MAT-Files
The MAT-file save options apply when you use the save function and the Save menu items (for MAT-files) in desktop tools.

By default, 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 encoding for strings when you save a MAT-file, making the data accessible to other MATLAB users, regardless of the default character encoding scheme used by their systems.

Prior releases of MATLAB 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.

The default preference for saving prevents you from using the MAT-files with MATLAB Version 6 or 6.x. To save MAT-files for use with a previous version, select the preference Ensure backward compatibility (-v6). Alternatively, you can override the preference by using the save function with the -v6 option, which, for occasional use, might be more convenient than the changing the preference. For more information, see the save reference page.

Source Control
Specify the source control system you want to interface MATLAB to. For more information, see Chapter 9, “Source Control.”

About Preferences
Use preferences to specify options for each desktop tool. To access preferences:

1 Select File -> Preferences.

2 In the left pane of the Preferences dialog box, preferences appear for MATLAB tools as well as for any other MathWorks products installed on your system.

Choose a tool and click the + to display more preferences for that tool. 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 MATLAB sessions.

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

**Function Alternative**
Open the preferences dialog box using the `preferences` function.

**Preferences File, matlab.prf**
Preferences are stored in a preferences file, `matlab.prf`. Type `prefdir` in the Command Window to see the location of the file.

On Macintosh platforms, the directory might be in a hidden folder, for example, `myname/.matlab/R14`. To access the directory, select **Go -> Go to Folder** in the Mac OS Finder. In the resulting dialog box, type the path returned by `prefdir` and press **Enter**. The `matlab.prf` file is loaded when MATLAB starts and is overwritten when you close MATLAB.

When you install a new version of MATLAB, it tries to use your existing preferences from the previous version, where possible. To return to default preferences, delete `matlab.prf` in the directory for the current version, R14, as well as in the directories for previous versions, for example, R13.
## Summary of Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>What You Can Specify</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Preferences</td>
<td>Toolbox path caching, figure window printing, delete function behavior, MAT-file save formats, and source control.</td>
</tr>
<tr>
<td>Fonts</td>
<td>Font type, style, and size for desktop tools. Customize for any tool.</td>
</tr>
<tr>
<td>Colors</td>
<td>Colors for text, background, and syntax highlighting in desktop tools.</td>
</tr>
<tr>
<td>Command Window</td>
<td>Numeric format and display, key bindings, tab size, and tab completion.</td>
</tr>
<tr>
<td>Command History</td>
<td>Display, filtering, and saving.</td>
</tr>
<tr>
<td>Editor/Debugger</td>
<td>Editor type, startup options, display, keyboard, indenting, language, publishing, and autosave.</td>
</tr>
<tr>
<td>Help</td>
<td>Product filter and synchronization.</td>
</tr>
<tr>
<td>Web</td>
<td>Internet proxy server settings.</td>
</tr>
<tr>
<td>Current Directory</td>
<td>Number of entries in history and display options.</td>
</tr>
<tr>
<td>Workspace</td>
<td>Confirm deletion of variables.</td>
</tr>
<tr>
<td>Array Editor</td>
<td>Numeric format and use of Enter key.</td>
</tr>
<tr>
<td>GUIDE</td>
<td>Display options.</td>
</tr>
<tr>
<td>Figure Copy Template</td>
<td>Application, text, line, uicontrols, axis, format, background color, and size.</td>
</tr>
<tr>
<td>Other Products</td>
<td>Preferences for other installed MathWorks products.</td>
</tr>
</tbody>
</table>
Running Functions—
Command Window and
History

If you are using the Help browser, watch the Desktop and Command Window video demo and the Command History video demo for an overview of the major functionality. The Command Window is where you run MATLAB statements, while the Command History is a log of the statements you have run.

Opening the Command Window (p. 3-2) Access the Command Window.

Running Functions and Programs, and Entering Variables (p. 3-3) Enter statements at the prompt. Run M-files, interrupt programs, run external programs, and examine errors. Evaluate and open selections.

Controlling Input (p. 3-8) Includes case sensitivity, long statements, syntax highlighting, editing, and keyboard shortcuts.

Controlling Output (p. 3-18) Covers suppressing and paging output, printing, and saving a session.

Searching in the Command Window (p. 3-22) Use the Find dialog or incremental search features to find content in the Command Window.

Preferences for the Command Window (p. 3-27) Specify options for text, display, the keyboard, and indenting.

Command History (p. 3-30) View session histories. Run statements, copy entries, search, and print the history. Set preferences.

Preferences for Command History (p. 3-37) Specify how often to automatically save the history file and the types of statements to exclude.
Opening the Command Window

The Command Window is one of the main tools you use to enter data, run MATLAB functions and scripts, and display results. If you are using the Help browser, watch the Desktop and Command Window video demo for an overview of the major functionality.

When the Command Window is not open, select 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, select Desktop -> Desktop Layout -> Command Window Only. For more information about arranging tools, see “Arranging the Desktop—Overview” on page 2-5.

The prompt, >>, is where you enter functions. 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 function. This prompt is also known as the command line.

When MATLAB displays the K>> prompt in the Command Window, it indicates MATLAB is in debug mode. Type dbquit to return to normal mode. For more information, see “Editing and Debugging M-Files” on page 6-1.

MATLAB displays the EDU>> prompt for the MATLAB Student Version.
Running Functions and Programs, and Entering Variables

- “Running Statements at the Command Line Prompt” on page 3-3
- “Running External Programs” on page 3-6
- “Evaluating or Opening a Selection” on page 3-6
- “Hyperlinks for Running Functions” on page 3-7

Running Statements at the Command Line Prompt

- “Entering Variables and Running Functions” on page 3-3
- “Examining Errors” on page 3-4
- “Running M-Files” on page 3-5
- “Processing Order” on page 3-5
- “Interrupting a Running Program” on page 3-5

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, MATLAB responds with

A =

1  2  3
4  5  6
7  8 10
To run a function, type the function including all arguments and press **Enter** or **Return**. All of the information you type before pressing **Enter** or **Return** is known as a statement. MATLAB displays the result. For example, type

```
magic(2)
```

and MATLAB returns

```
an =
  1   3
  4   2
```

When you enter program control statements, such as `if` ... `end`, the prompt does not appear until you complete the set of statements. 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`.

To display each function in a function or M-file as it executes, run **echo on**. For details, see the echo reference page.

**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 filename and press **Ctrl+Enter**. The offending M-file opens in the Editor, scrolled to the line containing the error.
Running M-Files
Run M-files, files that contain code in the MATLAB language, the same way that you would run any other MATLAB function. Type the name of the M-file in the Command Window and press **Enter** or **Return**. The M-file must be in the MATLAB current directory or on the MATLAB search path—for details, see “Search Path” on page 5-18. You can also use the `run` function and specify the full pathname to an M-file script.

To determine the name of the M-file currently running, use `mfilename`.

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 will run when the previous one finishes.

Interrupting a Running Program
You can stop a running program by pressing Ctrl+C or Ctrl+Break at any time. On Macintosh platforms, you can also use Command+. (Command key and the period key) to stop the program. For certain operations, this might generate errors in the Command Window.

For M-files that run for a long time, or that call built-ins or MEX-files that take a long time, Ctrl+C does not always effectively stop execution. In that event, include a `drawnow` command in your M-file, for example, within a large loop.

Note that Ctrl+C might be less responsive if you started MATLAB with the -nodesktop option.
Running External Programs
The exclamation point character, !, 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, 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. On the PC, add & to the end of the line, such as

```
!dir &
```

to display the output in a separate window or to run the application in background mode. For example

```
!excel.exe &
```

opens Excel and returns control to the Command Window so you can continue running MATLAB statements.

See the functions unix, dos, and system to run external programs that return results and status.

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.

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, press Enter or Return. For example, you can scroll up in the Command Window, select a statement you entered previously, and then press Enter or Return to run it. You cannot evaluate a selection while MATLAB is busy, for example, running an M-file.
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 Array Editor.
- Models open in Simulink.

See open 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.

**Hyperlinks for Running Functions**

Use `matlab:` to run a specified statement when you click a hyperlink in the Command Window. For example

```matlab
disp('<a href="matlab:magic(4)">Generate magic square</a>')
```

displays

Wrap the hyperlink text in HTML tags. The example 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. For more information, including examples, see the `matlabcolon` (`matlab:`) reference page.
Controlling Input

- “Case and Space Sensitivity” on page 3-8
- “Syntax Highlighting” on page 3-9
- “Cut, Copy, Paste, and Undo Features” on page 3-10
- “Enter Multiple Lines Without Running Them” on page 3-10
- “Entering Multiple Functions in a Line” on page 3-11
- “Entering Long Statements” on page 3-11
- “Recalling Previous Lines” on page 3-12
- “Tab Completion” on page 3-12
- “Keyboard Shortcuts in the Command Window” on page 3-16
- “Navigating Above the Command Line” on page 3-17

Case and Space Sensitivity

- “Uppercase and Lowercase for Variables” on page 3-8
- “Uppercase and Lowercase for Files and Functions” on page 3-8
- “Spaces in Expressions” on page 3-9

Uppercase and Lowercase for Variables

With respect to case, MATLAB 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, filenames, 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. The warning text includes instructions for how to suppress the warning.

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 run on Windows and UNIX 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 to Java actually used 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`. 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`. On UNIX platforms, MATLAB runs `a.m` but does not display a warning. On Windows platforms, MATLAB runs `a.m` but displays a warning the first time you do this. As noted in the warning message, you can suppress the warning by running

```
warning off ...
MATLAB:dispatcher:CaseInsensitiveMatchPrecedesExactMatch
```

Include the statement in your `startup.m` file if you never want to see this warning message.

**Spaces in Expressions**

Blank spaces around operators such as `-`, `;`, and `( )`, are optional, but they improve readability. For example, MATLAB interprets the following statements the same way.

```
y = sin (3 * pi) / 2
y=sin(3*pi)/2
```

**Syntax Highlighting**

Some entries appear in different colors to help you better find elements, such as matching `if/else` statements. This is known as syntax highlighting. You can change the colors using preferences. Note that output does not appear with syntax highlighting, except for errors. For more information, see “Colors Preferences for Desktop Tools” on page 2-50.
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.

In a multiline set of statements containing keywords, such as if ... end, you cannot edit a line after entering it, even though you have not completed the flow. Use Ctrl+C to end the flow, and then enter the statements again. Use Shift+Enter instead of Enter at the end of each line so that you can edit the lines before completing the flow.

Enter Multiple Lines Without Running Them

To enter multiple lines before running any of them, use Shift+Enter or Shift+Return after typing a line. MATLAB does not run the line, the cursor moves down to the next line, and no prompt displays but you can type the next line. Continue for more lines. Then press Enter or Return to run all of the lines.

This allows you to edit any of the lines you entered before you pressing Enter or Return.
**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

```matlab
format short; x = (1:10)'; logs = [x log10(x)]
```

and then press **Enter** or **Return**. The functions run in left-to-right order.

**Entering Long Statements**

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, typing

```matlab
headers = ['Author Last Name, Author First Name, ' ...
          'Author Middle Initial']
```

results in

```matlab
headers =
Author Last Name, Author First Name, Author Middle Initial
```

MATLAB produces an error when you do not complete the string, so you could not continue the statement as shown here.

```matlab
headers = ['Author Last Name, Author First Name, ...
          Author Last Name']
```

```matlab
??? 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 6-17.
Recalling Previous Lines
Use the arrow, tab, and control keys on your keyboard to recall, edit, and reuse functions you typed earlier. For example, suppose you mistakenly enter
\[
\rho = (1+ \sqrt{5})/2
\]
Because you misspelled \texttt{sqrt}, MATLAB responds with

\texttt{Undefined function or variable 'sqt'.}

Instead of retyping the entire line, press the up arrow \texttt{\uparrow} key. The previously typed line is redisplayed. Use the left to move the cursor, then add the missing \texttt{r}, and press \texttt{Enter} or \texttt{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, 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 up arrow to recall the previous line. For example, type the letters \texttt{plo} and then press the up arrow key. This displays the last line that started with \texttt{plo}, as in the most recent \texttt{plot} function. Press the up arrow key again to display the next most recent line that began with \texttt{plo}, and so on. Then press \texttt{Enter} or \texttt{Return} to run the line. This feature is case sensitive.

Another way to view and access commands from the current and previous MATLAB sessions is with the Command History.

Tab Completion
MATLAB helps you automatically completes the name of a function on the search path, filename, variable, structure, or Handle Graphics property if you type the first few characters and then press the \texttt{Tab} key. To use tab completion, you must have the tab completion preference selected. For details, see “Keyboard and Indenting Preferences for the Command Window” on page 3-29.

The examples explain how to use tab completion:

- “Basic Example—Unique Completion” on page 3-13
- “Multiple Possible Completions” on page 3-13
- “Tab Completion for Directories and Filenames” on page 3-14
- “Tab Completion for Structures” on page 3-14
- “Tab Completion for Properties” on page 3-15
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, displaying

```
costs_march
```

Then complete the statement, adding any arguments and 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.

You can also 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`.

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

```
  com
  comets
  cosh
  costs_march
```
The resulting list of possibilities includes the variable name you created, costs_march, but also includes functions that begin with cos, including cosets from the Communications Toolbox.

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 to select that item to complete 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 select from the list of possible completions using navigation keys such as up and down arrows, and Page Up. You can clear the list without selecting anything by pressing Escape. Note that the list of possible completions might include items that are not valid commands, such as private functions.

**Tab Completion for Directories and Filenames**

Tab completion works for directories and filenames in MATLAB functions. For example, type

```
edit d:/<Tab>
```

MATLAB displays the list of directories and files in d, from which you can choose one. For example, type

```
mym<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 filenames until you finish the edit statement.

**Tab Completion for Structures**

For structures, after the period separator, press Tab. For example, type

```
mystruct.<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<Tab>

which completes the entry mystruct.name, where mystruct contains no other fields that begin with n.

**Tab Completion for Properties**

Complete property names using tab completion, as in this graphics example. Here, f is a figure. Type

    set(f, 'paper<Tab>

MATLAB displays

```
    set(f, 'paper'
    PaperOrientation
    PaperPosition
    PaperPositionNode
    PaperSize
    PaperType
    PaperUnits
```

Select a property from the list. For example, type

    u<Enter>

and MATLAB completes the property, including the closing quote.

    set(f, 'paperunits'

Continue adding to the statement, as in this example

    set(f, 'paperunits', 'c<Tab>

MATLAB automatically completes the property

    set(f, 'paperUnits', 'centimeters'

because centimeters is the only possible completion.
**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, you can use shortcuts for menu items, which you can view on the menus, as well as general desktop shortcuts described in “Keyboard Shortcuts” on page 2-35. If you select the Emacs (MATLAB standard) preference for keybindings (see “Command Line Key Bindings” for an explanation), you can also use the Ctrl+key combinations shown in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Control Key for Emacs (MATLAB standard) Preference</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>Ctrl+P</td>
<td>Recall previous line—for details, see “Recalling Previous Lines” on page 3-12. See also “Command History” on page 3-30, which is a log of previously used functions, and “Keeping a Session Log” on page 3-21. With the Accessibility preference selected, moves the cursor up a line when it is above the command line. In that event, use Ctrl+↑ to recall previous lines for Windows and Macintosh key bindings.</td>
</tr>
<tr>
<td>↓</td>
<td>Ctrl+N</td>
<td>Recall next line—for details, see “Recalling Previous Lines” on page 3-12. 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 command line. In that event, use Ctrl+↓ to recall previous lines for Windows and Macintosh key bindings.</td>
</tr>
<tr>
<td>Ctrl+Home</td>
<td></td>
<td>Move to top of Command Window.</td>
</tr>
<tr>
<td>Ctrl+End</td>
<td></td>
<td>Move to end of Command Window.</td>
</tr>
<tr>
<td>←</td>
<td>Ctrl+B</td>
<td>Move back one character.</td>
</tr>
<tr>
<td>→</td>
<td>Ctrl+F</td>
<td>Move forward one character.</td>
</tr>
<tr>
<td>Ctrl+ →</td>
<td></td>
<td>Move right one word.</td>
</tr>
</tbody>
</table>
### Navigating Above the Command Line

To peruse 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 so you cannot use them 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.

<table>
<thead>
<tr>
<th>Key</th>
<th>Control Key for Emacs (MATLAB standard) Preference</th>
<th>Operation (Continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+ ←</td>
<td>Move left one word.</td>
<td></td>
</tr>
<tr>
<td>Home</td>
<td>Ctrl+A</td>
<td>Move to beginning of line.</td>
</tr>
<tr>
<td>End</td>
<td>Ctrl+E</td>
<td>Move to end of line.</td>
</tr>
<tr>
<td>Esc</td>
<td>Ctrl+U</td>
<td>Clear the command line when cursor is at the command line. Otherwise, move cursor to command line.</td>
</tr>
<tr>
<td>Delete</td>
<td>Ctrl+D</td>
<td>Delete character at cursor in command line.</td>
</tr>
<tr>
<td>Backspace</td>
<td>Ctrl+H</td>
<td>Delete character before cursor in command line.</td>
</tr>
<tr>
<td></td>
<td>Ctrl+K</td>
<td>Cut contents (kill) to end of command line.</td>
</tr>
<tr>
<td>Shift+Home</td>
<td></td>
<td>Select to beginning of line.</td>
</tr>
<tr>
<td>Shift+End</td>
<td></td>
<td>Select to end of last line. Can start at any line in the Command Window.</td>
</tr>
<tr>
<td>Enter</td>
<td>in selection</td>
<td>Append selection to statement at command line and execute it.</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>in hyperlink</td>
<td>Execute hyperlink displayed in Command Window. For example, in the hyperlink of an error message, opens the file in the Editor at that line number.</td>
</tr>
</tbody>
</table>
Controlling Output

- “Suppressing Output” on page 3-18
- “Paging of Output in the Command Window” on page 3-18
- “Formatting and Spacing Numeric Output” on page 3-19
- “Clearing the Command Window” on page 3-20
- “Printing Command Window Contents” on page 3-20
- “Keeping a Session Log” on page 3-21

Suppressing Output
If you end a line with a semicolon (;) and then press Enter or Return, MATLAB runs the statement but does not display any output. This is particularly useful when you generate large matrices. For example, running

\[
A = \text{magic}(100);
\]

creates \( A \) but does not show the resulting matrix in the Command Window.

Paging of Output in the Command Window
If output in the Command Window is lengthy, it might not fit within the screen and might 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

```matlab
--more--
```

indicating there is more output to display. Press one of the following keys.

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter or Return</td>
<td>To advance to the next line</td>
</tr>
<tr>
<td>Space Bar</td>
<td>To advance to the next page</td>
</tr>
<tr>
<td>q</td>
<td>To stop displaying the output</td>
</tr>
</tbody>
</table>
You can scroll up in the Command Window to see input and output that no longer fit in the view. To use the up and down arrow keys to go to the input and output that no longer fit in the view, select the Command Window Accessibility preference.

**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. Use the text display preference to change the numeric format of output for the current and future sessions. 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`, with components of different magnitudes.

```matlab
x = [4/3 1.2345e-6]
```

```matlab
format short
1.3333    0.0000
```

```matlab
format short e
1.3333e+000  1.2345e-006
```

```matlab
format +
++
```

A complete list and description of available formats is at the reference page for `format`. For more control over the output format, use the `sprintf` and `fprintf` functions.
Controlling Spacing
Use the text display preference or format function to control spacing in the output. Use

```matlab
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

```matlab
format loose
```

Clearing the Command Window
Select Clear Command Window from the Edit 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.

**Function Alternative.** Use clc to clear the Command Window. Similar to clc is the home function, which moves the prompt to the top of the Command Window but does not clear the text in the window.

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 “Page Setup Options for Printing” on page 2-40.
Keeping a Session Log

The diary Function
The diary function creates a copy of your MATLAB session in 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. To create a file on your disk called `sept23.out` that contains all the functions you enter, as well as MATLAB output, enter

```matlab
diary('sept23.out')
```

To stop recording the session, use

```matlab
diary('off')
```

Other Session Logs
There are two other means of viewing session information:

- The Command History, which contains a log of all functions executed in the current and previous sessions.
- The `logfile` startup option—see “Startup Options” on page 1-4.
Searching in the Command Window

You can search for specified text that appears in the Command Window, where the text was either part of input you supplied, or output displayed by MATLAB. There are two search features for the Command Window:

- “Find Dialog” on page 3-22
- “Incremental Search” on page 3-23

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.

See also “Recalling Previous Lines” on page 3-12, “Tab Completion” on page 3-12, and “Keyboard Shortcuts in the Command Window” on page 3-16 for techniques to reuse previous statements and navigate in the Command Window.

Find Dialog

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 **Next** or **Previous** to find another occurrence, or press **F3** or **Shift+F3**.

![Find Dialog](image)

MATLAB beeps when a search for **Next** reaches the end of the Command Window, or when a search for **Previous** reaches the top of the Command Window. If you have **Wrap around** selected, it continues searching after beeping.
Note that you can only search for text currently displayed in the Command Window. To increase the amount of information maintained in the Command Window, increase the setting for command session scroll buffer size in Command Window preferences, and do not clear the Command Window.

**Incremental Search**

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. Incremental search is also available in the Editor/Debugger—see “Incremental Search” on page 6-25. To use the incremental search feature in the Command Window:

1. Position the cursor where you want the search to begin.

2. How you begin the incremental search depends on your setting for the command line key bindings preference:
   - Press Ctrl+S for **Emacs**, or
   - Press Ctrl+Shift+S for **Windows**

   To look for the previous occurrence, type 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).
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 3-26.
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.

4 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.

If MATLAB beeps, the display shows Failing F Inc Search, followed by the text you entered. This means either that the text was not found, or the search wrapped past the end or beginning of the Command Window.
To end the incremental search, press Esc 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.

If you end incremental search and then enter Ctrl+S or Ctrl+R, the search term from your previous incremental search appears in the Inc Search field.

Incremental search is also available in the Editor/Debugger.

**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 bO in the Inc Search field and incremental search does not find any matching text because the 0 is uppercase.
Preferences for the Command Window

To set preferences for the Command Window, select File -> Preferences and then select Command Window in the left pane of the Preferences dialog box.

Set these Command Window preferences:

- “Format, Display, and Accessibility Preferences” on page 3-27
- “Keyboard and Indenting Preferences for the Command Window” on page 3-29

Format, Display, and Accessibility Preferences

Text Display
Specify the format, that is, how output appears, in the Command Window.
**Numeric format.** Output format of numeric values displayed in the Command Window. This affects only how numbers are displayed, not how MATLAB computes or saves them. The `format` reference page includes the list of available formats, with examples.

**Numeric display.** 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.

**Limit matrix display width to eighty columns.** When selected, MATLAB displays only 80 columns of matrix output, regardless of the width of the Command Window. Clear the check box if you make the Command Window wider than 80 columns and want matrix output to fill the width of the Command Window. See also the `display` reference page.

To determine the number of columns and rows that will display in the Command Window, given its current size, use

```matlab
get(0,'CommandWindowSize')
```

With the matrix display width preference cleared, the number of columns is based on the width of the Command Window. With the preference set to 80 columns, the number of columns is always 80.

**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 in the Command Window. Using the up arrow, 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 Ctrl+ up arrow or down arrow to recall statements for Windows and Macintosh key bindings, or Ctrl+P and Ctrl+N for Emacs (MATLAB standard) keybindings.

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.

Keyboard and Indentation Preferences for the Command Window

Command Line Key Bindings
Specify the keyboard shortcuts to be used at the command line.

Emacs (MATLAB standard). Use the control keys listed in “Keyboard Shortcuts in the Command Window” on page 3-16, which should be familiar to existing MATLAB users and Emacs users. For example, Ctrl+A moves the cursor to the beginning of the line.

Windows. Use Windows standard control keys. For example, Ctrl+A is the shortcut for Edit -> Select All, which selects the entire contents of the Command Window.

Macintosh. Available only on Macintosh platforms, specify to use Macintosh keys, such as the Command key. When selected, you can use the Macintosh command key instead of the Ctrl key.

Tab Key
Tab size. Number of spaces assigned to a tab stop when displaying output.

Enable tab completions. Select the check box if you want to use tab completion when typing functions in the Command Window. Clear the check box if you do not want to use the tab completion feature. In that event, when you press the Tab key, MATLAB moves the cursor to the next tab stop rather than completing a function.
Command History

The Command History window displays a log of the statements most recently run in the Command Window. If you are using the Help browser, 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 when it is closed, or to select it when it is open. For details, see “Arranging the Desktop—Overview” on page 2-5.

Use the Command History window as described in these sections:

- “Viewing Statements in the Command History Window” on page 3-31
- “Using Statements from the Command History” on page 3-32
- “Searching in the Command History” on page 3-33
- “Searching in the Command History” on page 3-33
- “Printing the Command History” on page 3-35

Timestamp marks the start of each session. Select it to select all statements in the history for that session.

Click - to hide history for that session.
Click + to expand.

Select one or more lines and right-click to copy, evaluate, create a shortcut, or create an M-file from the selection.
MATLAB provides other options for viewing a history of statements. See also

- “Recalling Previous Lines” on page 3-12, which describes using the up arrow in the Command Window
- The diary function reference page
- “Startup Options” on page 1-4, which includes the logfile startup option

**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 - keys to show and hide entries.

The Command History file is history.m. Type prefdir in the Command Window to see the location of the file. The history.m file is loaded when MATLAB starts.

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 with the Settings preference. For details, see “Preferences for Command History” on page 3-37.
Using Statements from the Command History

You can select statements in the Command History and then perform the following actions on the selected statements.

<table>
<thead>
<tr>
<th>Action</th>
<th>How to Perform the Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run the statements in the Command Window</td>
<td>Double-click an entry (entries) in the Command History window to execute the statement(s). For example, double-click edit myfile to open myfile.m in the Editor. You can also run a statement by right-clicking it and selecting <strong>Evaluate Selection</strong> from the context menu, and by selecting a statement and pressing <strong>Enter</strong> or <strong>Return</strong>.</td>
</tr>
<tr>
<td>Edit and run the statements in the Command Window</td>
<td>Select an entry or entries and then select <strong>Copy</strong> 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 <strong>Enter</strong> or <strong>Return</strong> to execute them.</td>
</tr>
<tr>
<td>Copy the statement(s) to another window</td>
<td>Select an entry or entries and then select <strong>Copy</strong> 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 to an open M-file or another application.</td>
</tr>
</tbody>
</table>
## Searching in the Command History

There are two types of search in the Command History:

- “Find Next Entry By Letter” on page 3-33
- “Find Text” on page 3-34

After finding an entry, you can copy and paste it into an M-file or other file, or you can right-click and select **Evaluate Selection** to run the entry.

### Find Next Entry By Letter

Type a letter in the Command History to move to the next entry that begins with that letter, as illustrated in the following example.

1. Position the cursor at which `collatz` in the Command History.

### Action | How to Perform the Action (Continued)
--- | ---
Create an M-file from the 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 the 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 “Shortcuts for MATLAB—Easily Run a Group of Statements” on page 2-21.
2 Type 1. The Command History selects the next entry that begins with 1, in this example, load.

Type a letter, in this example 1, and the Command History selects the next entry that begins with that letter.

Find 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 **Next** or **Previous** to find another occurrence, or press **F3** or **Shift+F3**. Find looks for visible entries only, that is, it does not find entries in collapsed nodes.
MATLAB beeps when a search for **Next** reaches the end of the Command History, or when a search for **Previous** reaches the top of the Command History. If you have **Wrap around** selected, it continues searching after beeping.

**Printing the Command History**

To print the contents of the Command History window, select **File -> Print**. Specify options for printing by selecting **File -> Page Setup**. For example, you can print the history with a header. For more information, see “Page Setup Options for Printing” on page 2-40.

**Deleting Entries in the Command History Window**

Delete entries in the Command History window when you feel there are too many and it becomes inconvenient to find the ones you want. All entries remain until you delete them.

To delete entries in the Command History window, first select the entries to delete:

- 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 of their entries.

Then right-click and select **Delete selection** from the context menu, or press the **Delete** key.
To delete all entries, select **Edit -> Clear Command History**.

After deleting entries from the Command History, you will not be able to recall those statements in the Command Window as described in “Recalling Previous Lines” on page 3-12.
Preferences for Command History

Using Command History preferences, you can choose to exclude statements from the Command History and specify how often to save the Command History.

To set preferences for the Command History, 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. Note that when you exclude statements from the Command History, you cannot recall them in the Command Window as described in “Recalling Previous Lines” on page 3-12.

Save Exit/Quit Commands
Select the check box to save exit and quit commands in the Command History.

Save Consecutive Duplicate Commands
Select the check box if you want consecutive executions of the same statement to be saved to the Command History.

For example, with this option selected, run \texttt{magic(5)}, and then run \texttt{magic(5)} again. The Command History saves two consecutive entries for \texttt{magic(5)}. With this option cleared, for the same example, the Command History saves one entry for \texttt{magic(5)}. If you then run \texttt{magic(10)}, the Command History saves two entries, \texttt{magic(5)} followed by \texttt{magic(10)}.

Saving
Use Saving preferences to specify how often to automatically save the Command History during a MATLAB session.

Save History File On Quit
Select this option to save the Command History when you end the MATLAB session. 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 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 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 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.
Help for Using MATLAB

If you are using the Help browser, watch the Help and Documentation video demo for an overview of the major functionality.

Types of Documentation (p. 4-2) Use the type of documentation that best meets your need.
Help Browser (p. 4-4) Overview to finding information about your MathWorks products using the Help browser.
Find Information with the Help Browser (p. 4-7) Use the contents listing of the online documentation, a global index, and full-text search.
View Documentation in the Help Browser (p. 4-19) After finding documentation, view the documentation and perform other operations using the display pane.
Demos in the Help Browser (p. 4-23) Run demonstration programs, and view and copy the M-file code behind them.
Preferences for the Help Browser (p. 4-27) Specify fonts used in the Help browser and limit the documentation included using the product filter.
Printed Documentation (p. 4-30) Print from the Help browser or from the PDF version of the documentation, or purchase printed documentation.
Help Functions (p. 4-32) Use functions to get information, such as help and doc.
Other Forms of Help (p. 4-37) Use product-specific help features, download M-files, contact Technical Support, see documentation for other MathWorks products, view a list of other books, and participate in a MATLAB newsgroup.
## 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.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Type of Documentation</th>
<th>Description and When to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>🌟</td>
<td>Getting Started</td>
<td>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 or demos and examples.</td>
</tr>
<tr>
<td>📂</td>
<td>Release Notes</td>
<td>An overview of new products and features in a release, it also includes upgrade information and any known problems and limitations. Review the Release Notes for all your products when you first start using a new release.</td>
</tr>
<tr>
<td>📚 or 📚</td>
<td>Product</td>
<td>MATLAB and toolboxes use orange book icons 📚, while Simulink, Blocksets, and related products use blue book icons 📚.</td>
</tr>
<tr>
<td>🕵️</td>
<td>Index of Examples</td>
<td>Accessible via the Help browser Contents listing, this is an index of the major examples included in the Help browser documentation.</td>
</tr>
<tr>
<td>📖</td>
<td>User Guides (blue)</td>
<td>User guide material is comprehensive, containing overviews as well as detailed instructions. Consult it after reviewing Getting Started material.</td>
</tr>
<tr>
<td>🗑️</td>
<td>Reference Pages (orange)</td>
<td>Each function has a reference page that provides the syntax, description, examples, and other information for that function. It includes links to related functions and additional information. Reference pages are also provided for blocks and properties.</td>
</tr>
</tbody>
</table>
Types of Documentation

<table>
<thead>
<tr>
<th>Icon</th>
<th>Type of Documentation</th>
<th>Description and When to Use (Continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>🎨</td>
<td>Printable Documentation</td>
<td>Most products provide the online documentation in a printable format, PDF. Access PDF files via the Help browser and print them from your PDF reader, such as Adobe Acrobat. Most PDF files reside only on The MathWorks Web site, so you need an Internet connection to view them.</td>
</tr>
<tr>
<td>🔍</td>
<td>Support and Web Services</td>
<td>Provides access to the MathWorks Technical Support searchable database, maintained on the MathWorks Web site. Use it to find solutions to questions posed by users.</td>
</tr>
<tr>
<td>🎉</td>
<td>Demos</td>
<td>MathWorks products come with examples that demonstrate features of the product. Many of the demos actually run MATLAB code. Use the Help browser Demos tab to access demos for the products you have installed.</td>
</tr>
<tr>
<td>none</td>
<td>M-File Help</td>
<td>Get M-file help in the Command Window to quickly access basic information for a function. 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.</td>
</tr>
</tbody>
</table>

**Accessing Documentation on the Web**

You can access all product documentation on The MathWorks Web site at http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml. Use the Web site documentation for products you have not installed, or for prior versions of a product. PDF documentation is available only on the Web site.
Help Browser

Use the Help browser to search and view documentation and demonstrations for MATLAB and all other installed MathWorks products. The Help browser is an HTML browser integrated with the MATLAB desktop.

To open the Help browser, click the help button \( \text{\textcopyright} \) 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 Contents, Index, Search, and Demos tabs. For more information, see “Find Information with the Help Browser” on page 4-7.
- The display pane, on the right, for viewing documentation and demos.
**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 “Help Fonts Preferences—Specifying Font Name, Style, and Size” on page 4-29.

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 \[\times\] in the pane’s upper right corner. To open the Help Navigator pane from the display pane, click the Help Navigator icon on the toolbar.[page 4-5] Alternatively, use the View menu.

To show only the display pane, as in this illustration, select View -> Help Navigator.

Then click the Help Navigator icon to display the Navigator again.

---

**Tab Completion**

MATLAB helps you automatically completes the name of a function on the search path, filename, variable, structure, or Handle Graphics property if you type the first few characters and then press the Tab key. To use tab completion, you must have the tab completion preference selected. For details, see Keyboard and Indenting Preferences for the Command Window.

The examples explain how to use tab completion:

- Basic Example—Unique Completion
- Multiple Possible Completions
- Many Completions
- Tab Completion for Directories and Filenames
Adding Your Own Help Files to the Help Browser

You can add your own HTML help files so they appear in the Help browser. For details, see “Adding Your Own Toolboxes to the Development Environment” in the online documentation.
Find Information with the Help Browser

Use the Help Navigator, the left pane in the Help browser, to find information. These sections describe the main features:

- “Contents Listing in the Help Browser” on page 4-7—View an expandable table of contents for documentation.
- “Index for Help Browser” on page 4-10—Use keywords to find information.
- “Search Documentation with the Help Browser” on page 4-12—Find documentation using full-text and other forms of search.
- “Favorites” on page 4-18—Bookmark pages you want to refer to again.

Contents Listing 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.
Product Roadmap
When you select a product in the Contents pane (any entry with a book icon), such as MATLAB or the Communications Toolbox, a roadmap of the documentation for that product appears in the display pane. The roadmap includes links to commonly used 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 MATLAB to the printable documentation)

Navigate 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 Navigator pane. Shortcuts are: 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. Shortcuts are: 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 expand all nodes for the selection.
- Use the down and up arrow keys to move through the list of items.

Icons in the Contents Listing
Icons for entries in the top levels of the Contents listing represent the type of documentation so you can quickly find the kind of information you need for a product. See the legend for icons in “Types of Documentation” on page 4-2.

Product Pages
After expanding the listing for a product in the Contents pane, the last entry is Product Page (Web). This links to the MathWorks Web site for the latest information about that product. Like other links to the Web, the page opens in your system Web browser.
Synchronize Contents Listing with Display Pane

By default, the topic highlighted in the Contents pane matches the title of the page appearing in the display pane. The Contents 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. With synchronization, you know what book and section the displayed page is part of. Note that synchronization only applies to the major headings in a document. For pages that begin with lower level headings, the Contents listing does not synchronize.

You can turn off synchronization. To do so, use preferences. See “General—Keep Contents Synchronized” on page 4-28.

Synchronization only applies to the Contents pane. The page shown in the display pane does not necessarily correspond to the selection in the Index, Search, or Demos tabs. However, if you return to the Contents pane, the Contents pane synchronizes with the displayed page.
Index for Help Browser
To find specific index entries (selected keywords) in the MathWorks documentation for installed products, use the Index in the Help Navigator pane.

1 Click the Index tab.

2 Type a word or words in the Search index for 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 headings and do not link to any page.)

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 for all installed products, select File -> Preferences -> Help, and clear the Enable product filter check box.
- To see entries for selected products only, select File -> Preferences -> Help and set the product filter.
- 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 creating M-files, type M-files and look for the subentry creating.
- After selecting an entry, search for specified text in the displayed page using the Find tool, accessible from the binoculars icon 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 it.
- For different but related results, try the Search tab—for instructions, see “Search Documentation with the Help Browser” on page 4-12.
Search Documentation with the Help Browser

- “Search Pane in Help Browser” on page 4-12
- “Boolean Operators in Search” on page 4-15
- “More About Search” on page 4-15
- “Get Fewer Results” on page 4-16
- “Get More Results” on page 4-17

Search Pane in Help Browser
To look for a specific word or phrase in the documentation, use the Search tab in the Help Navigator.

1 To limit (or extend) the products whose documentation is searched, set the product filter.

2 Click the Search tab.

3 Type the word or words you want to find in the Search for field, and click Go (or press Enter or Return). You can use Boolean operators between the words—see “Boolean Operators in Search” on page 4-15.

The documents containing all of the exact search words are listed, with the documentation Section and the Product name shown in the second third
columns, providing context for the results. You might need to make the Navigator pane wider to see all columns. The total number of results appears at the bottom left side of the pane.

4 Select an entry from the list of results.

The selected page appears in the display pane with all occurrences of the search words highlighted using a different color for each search word. Search words remain highlighted until you view another page or until you click the page refresh button in the toolbar.

In the display pane, use the Find tool, accessible from the binoculars icon on the toolbar, to go directly to the next instance in that page of a word you specify.

5 Search results are ordered by relevance. For example, titles that contain all search words appear first, while pages containing a single instance of each search word appear last.

- Change the order of the results by clicking a column heading. For example, click Product to group results by product. Click Title to sort titles alphabetically. A triangular icon indicates the column on which you most recently sorted. After changing the order of results, you need to rerun the search to see results ordered by relevance.

- Change the order of the columns by dragging a column to a new position. For example, you can drag the Product column so it is second, which makes the Title column third.

- Make columns wider or narrower by dragging the separator bar between the column headings.

6 For more results, you can search for the words in the Technical Support database on The MathWorks Web site by clicking the link at the bottom of the Search results pane.
Results are sorted by title relevance. Change the order of results by clicking a column heading.

View and run previous searches.

Instances of search words are highlighted. Click the refresh button to clear highlights.

Use Find to go directly to specified words.

Summary of search results.

For more results, search Technical Support solutions and notes.
**Function Alternative.** From the Command Window, use `docsearch` to open the Help browser to the **Search** pane and search for the specified term. For example

```
    docsearch('print figure')
```

finds all pages that contain the words `print` and `figure`. For details, see the `docsearch` reference page.

**Boolean Operators in Search**

The search automatically performs a Boolean **AND** for multiple words. In the example `print figure`, it finds all pages that have both the word `print` and the word `figure`, although the page might not necessarily have the exact phrase “`print figure`”.

You can refine the search by including the Boolean operators **AND**, **OR**, and **NOT** between words. The operators must be in all capital letters and there must be a space before and after each operator. The Boolean operators are evaluated in left to right order.

**Example Using Boolean Operators in Search.** Type

```
    print OR printing AND figure NOT exporting
```

to find all pages that contain the words `print` and `figure`, or `printing` and `figure`, but only if the page does not contain the word `exporting`. At the top of the results list are any pages that contain all the **AN**Ded and **OR**ed words in the page title.

**More About Search**

These are the guidelines search uses:

- Insignificant words (a, an, the, of) are ignored.
- Search is not case sensitive.
- You cannot enter quotation marks around words to find exact phrases and you cannot use wildcards.
- Search does not find operators and special characters, such as +, so instead use the **Index**.
- Search does not find numbers, but does find text that contains numbers. For example, search does not find 7, but does find `V7`.
• If you are searching for information about an option, try including the hyphen (-) before the option, for example, save - append.
• Search does not find words in demos.
• If you search for a function that is used in multiple products (called an overloaded function), the reference pages for all those products are listed. Use the Product column to distinguish the reference page you want.

Get Fewer Results
If there are too many results for the search to be useful, try the following.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Try These Suggestions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too many products</td>
<td>Select <strong>File -&gt; Preferences -&gt; Help</strong> and enable the product filter for specified products. For details, see “Product Filter” on page 4-27. Order results by product—click the <strong>Product</strong> (third) column. If you cannot see it, make the pane wider.</td>
</tr>
<tr>
<td>Page is not about search word, but just mentions it</td>
<td>Try the <strong>Index</strong> tab to see more important entries for that search word.</td>
</tr>
<tr>
<td>Too many irrelevant results</td>
<td>Type more than one word in the <strong>Search for</strong> field. Use Boolean operators (in all capitals), for example, printing <strong>AND</strong> figures <strong>NOT</strong> exporting.</td>
</tr>
<tr>
<td>Topic is not relevant</td>
<td>Look at the <strong>Section</strong> (second) column in the search results list, which provides context for the result. If you cannot see the column, make the pane wider.</td>
</tr>
</tbody>
</table>
**Get More Results**
If you want more results, try the following.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Try These Suggestions</th>
</tr>
</thead>
<tbody>
<tr>
<td>No results for the product</td>
<td>Be sure the product filter is set correctly. Select <strong>File -&gt; Preferences -&gt; Help</strong> and disable the product filter. For details, see “Product Filter” on page 4-27.</td>
</tr>
<tr>
<td>No results but you know the word should be there</td>
<td>Try variations of the search words with an OR between the words. For example, search for preference OR preferences to find all pages that contain either the word preference or the word preferences.</td>
</tr>
<tr>
<td>Not enough information</td>
<td>Try searching the Technical Support database by clicking the link at the bottom of the <strong>Search</strong> results pane.</td>
</tr>
</tbody>
</table>
Favorites
Favorites are bookmarks to pages in the Help browser documentation.

Add Favorites
To designate the displayed page as a favorite (that is, to bookmark it):

1 Select **Favorites -> Add to Favorites**.

2 The **Favorites Editor** dialog box opens. You can accept the defaults and click **Save**, or make changes to the entries:
   - Use the **Label** provided, or change it to another term.
   - Do not change the entry for **Callback**.
   - Maintain the **Category** as **Help Browser Favorites** so you can access them from the **Favorites** menu.
   - For **Icon**, keep the default **Help icon**, or choose another.

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.

Go to Favorites
Select the **Favorites** menu to view the list of pages you previously designated as favorites (bookmarks). Select an entry and that page appears in the display pane.

Organize Favorites
You can rename, remove, and reorder the listing of favorites. Select **Favorites -> Organize Favorites**. For more information, click **Help** in the **Organize Favorites** dialog box.
View Documentation in the Help Browser

After finding documentation with the Help Navigator, view the documentation in the display pane. The features available to you while viewing the documentation are

- “Browse to Other Pages” on page 4-20
- “Links” on page 4-21
- “Find Text in Displayed Pages” on page 4-21
- “Copy Information” on page 4-22
- “Evaluate a Selection” on page 4-22
- “View the Page Source (HTML)” on page 4-22
Browse to Other Pages

Use the arrow buttons in the page and in the toolbar to go to other pages.

Back button shows previous page you viewed in the Help browser.

Use left and right arrows in the page to go to the previous and next pages in the document.

View list of and go to previously viewed pages.

Return to pages previously viewed.

Desktop Tools and Development Environment

Tab Completion

MATLAB helps you automatically completes the name of a function on the search path, filename, variable, structure, or Handle Graphics property if you type the first few characters and then press the Tab key. To use tab completion, you must have the tab completion preference selected. For details, see Keyboard and Indenting Preferences for the Command Window.

The examples explain how to use tab completion:

- Basic Example—Unique Completion
- Multiple Possible Completions
- Many Completions
- Tab Completion for Directories and Filenames

View the next page in a document by clicking the right arrow at the top or bottom of the page. View the previous page in a document by clicking the left arrow 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 the page previously shown by clicking the back button in the display pane toolbar. After using the back button, view the next page shown by clicking the forward button in the display pane toolbar. These buttons work like the
forward and back buttons of popular Web browsers. You can also go back or forward by right-clicking a page and selecting **Back** or **Forward** from the context menu.

**Links**
Click links in the displayed page to get more information on the subject. Links appear underlined and in blue. Visited links appear in purple. Links to the Web display the linked page in your system Web browser. Click the middle mouse button to open the linked page in a separate window.

**Find Text in Displayed Pages**
You can find a phrase in the currently displayed page:

1. Click the binocular icon †. In the resulting **Find** dialog box, type the word or phrase you are looking for. You can type a partial word, for example, preference to find all occurrences of preference and preferences. Use the check boxes to specify options. Click **Next**.

   The page scrolls to the first occurrence of the phrase in the page and highlights it.

2. Press **Next** or **Previous** in the **Find** dialog box to find more occurrences in that page.

See “Search Documentation with the Help Browser” on page 4-12 for instructions on looking through all the documentation instead of just one page.
Copy Information
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 can then paste the information into another tool, such as the Command Window or Editor, or into another application, such as a word processing application.

Evaluate a Selection
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.

View the 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.
Demos in the Help Browser

MATLAB and related products include demos that you can access from the Help browser Demos tab. There are three basic kinds of demos:

- Video tutorials play a movie file highlighting features, such as the demos for the MATLAB Desktop Tools and Development Environment.
- Published M-files are HTML files that illustrate an example, step-by-step. An example is the MATLAB Programming Nested Functions demo.
- M-file GUIs are interactive tools for exploring a feature. An example is the MATLAB Graphics Visualizing Sound demo.

The M-file code for the demos (except video tutorials) is 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 the available demos for the products you have installed:

1. Click the Demos tab in the Help Navigator.

   You can also access demos from the Start button, using the demo function, or from the Help menu for some tools.

2. Click the + for a product area to list the products or categories that have demos. Then click + for a product or product category to list its demos.

3. Select a specific demo to use it. Information about the demo, including instructions for running it, appears in the display pane.
Access demos for all installed products using the **Demos** tab.

The code for the demo is in the specified file. Click this link to view the M-file code in the Editor.

Click this link to run the demo.

Expand the listing for a product and category to see its demos.

Select a demo to see details about it.

**Nested Function Examples**

This gives examples of how nested functions can be used for easy data sharing, as well as providing a new way to create customized functions.

**Contents**

- Example 1: Sharing data
- Example 2: Creating customized functions
- Example 3: Creating customized functions with __built__

**Example 1: Sharing data**

Let's first take a look at **taxDemo.m**, which contains a nested function:

```matlab
function taxDemo
    type taxDemo.m
```
4 You can
- View the source code (M-file) for the demo—click the View code link on the top left. For the example shown, it is nesteddemo.m. The M-file opens in the MATLAB Editor.
- Run the demo—click the link at the top right, Run this Demo. You can instead double-click the demo in the Navigator pane to run it.

When you run nesteddemo, the following appears. Follow the instructions shown to continue running the demo. In this example, click Start >>.
Notes About Demos

Some Help browser features do not apply to demos:

- You cannot use the **Search** tab to look for words or code contained in the demos. You can use the Find Files tool to search for code in M-file demo files.
- You cannot use **Add to Favorites** to bookmark some demos.
- The product filter does not apply to demos.

Function Alternative

To view the **Demos** 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.

To run a demo, type the demo name at the command line. For example, type

```
vibes
```

to run a visualization demonstration showing an animated L-shaped membrane.

Running Published M-File Demos. To run published M-file demos from the command line, type `playshow` followed by the demo name. To determine if a demo is a published M-file type, view the H1 line for the demo M-file, that is, the first comment line. If it begins with two comment symbols (%), it is a published M-file demo. For example, the first line in `nesteddemo` begins with two comment symbols:

```
%% Nested Function Examples
```

Therefore, type `playshow nesteddemo` to run the demo.

Adding Your Own Demos

You can add your own demos so they appear under the **Demos** tab. For details, see “Adding Your Own Toolboxes to the Development Environment” in the online documentation.
Preferences for the Help Browser

- “Product Filter” on page 4-27
- “PDF Reader—Specifying Its Location” on page 4-28
- “General—Keep Contents Synchronized” on page 4-28
- “Help Fonts Preferences—Specifying Font Name, Style, and Size” on page 4-29

Product Filter
If you have MathWorks products in addition to MATLAB, such as Simulink, toolboxes, and blocksets, set the product filter to limit the product documentation used, making it easier to narrow searches, for example:

1. Select File -> Preferences -> Help.
2. Under Product filter, select the check box for Enable product filter. Click Select products.

   The Help Product Filter dialog box opens.

3. Select the products whose documentation you want to appear in the Help Navigator. Click OK.

   The Help Navigator updates to include only those products you specified. The product filter settings are remembered for your next MATLAB session.

4. When you want to use documentation for all installed products, in Help preferences, clear the check box for Enable product filter.

With the product filter enabled:
- Contents shows only the subset of products you specify.
- Index shows only index terms for the subset of products you specify.
- Search only looks through the subset of products you specify.
- Demos is not affected; demos for all installed products are always shown.

**Example Using the Product Filter**

If you do a search and know the information you are seeking is in MATLAB or the Communications Toolbox, in the Help Product Filter, click Clear All and then select MATLAB and Communications Toolbox.

The Contents only shows MATLAB and the Communications Toolbox documentation, the Index only shows entries for MATLAB and the Communications Toolbox, and the Search feature only looks in and shows results for MATLAB and the Communications Toolbox.

**PDF Reader—Specifying Its Location**

If you want to view the PDF version of the documentation, use the link on the roadmap page for that product. To open the PDF file, the Help system needs to know the location of your PDF reader (for example, Adobe Acrobat).

For Windows systems, MATLAB reads the location from the registry, so you cannot specify its location using preferences.

For UNIX systems, when you installed MATLAB, it looked for your system's PDF reader. If found, MATLAB automatically supplied the PDF reader location to the PDF reader field in Help preferences. If MATLAB could not locate your PDF reader or if you moved your PDF reader since installation, change the location. Select File -> Preferences -> Help, and enter the location in the PDF reader field. Use the browse (...) button to navigate your file system to select the location.

**General—Keep Contents Synchronized**

By default, the displayed page is synchronized with the Contents listing. For more information about this feature, see “Synchronize Contents Listing with Display Pane” on page 4-9. Use this preference to turn synchronization off.

Select File -> Preferences -> Help. Under General, clear the check box for Keep contents tree synchronized with displayed document. Select the check box to turn synchronization back on.
Help Fonts Preferences—Specifying Font Name, Style, and Size

You can specify the font type, style, and size used in the Help Navigator and the display pane.

Use the same method as you would to specify fonts for any desktop tool—for more information, see “Fonts Preferences for Desktop Tools” on page 2-44. By default, the Help Navigator uses the desktop text font. The display pane is considered to be an HTML Proportional Text tool, and by default, uses the desktop text font.

This example changes the display pane font:

1 Select File -> Preferences -> Fonts -> Custom.

2 From the Desktop tools list, select HTML Proportional Text. The Help browser display pane is considered to be an HTML proportional text tool, as is the MATLAB Web browser. Changing the font preference affects both tools.

3 For Font to use, select Custom, then specify the font characteristics:
   - Type, for example, SansSerif
   - Style, for example, bold
   - Size in points, for example, 12 points

   After you make a selection, the Sample area shows how the font will look.

4 Click OK. The Help display pane fonts use the new settings. The MATLAB Web browser fonts will also use the new settings.
Printed Documentation

Printed manuals are provided for the major releases of some products and tools. The online documentation often has information not included with the printed manuals and is often more current. If you want to purchase printed documentation, see the online store at the MathWorks Web site at http://www.mathworks.com.

You can print the current page displayed in the Help browser, or print a page or an entire book from the PDF version of the documentation.

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.

The Pages field in the Print dialog box shows the total number of pages to be printed and lets you specify the range of pages you want to print. When there is more than one page, it means that multiple physical pages are needed to print the single page displayed in the Help browser.

Complete the dialog box and press OK to print the page.

Printing the PDF Version of Documentation

If you need to print only a few pages and if the quality does not need to be equivalent to pages in a printed book, you can print directly from the MATLAB Help browser—see “Printing a Page from the Help Browser” on page 4-30.

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. PDF documentation is shown and printed using your PDF reader, usually Adobe Acrobat Reader. The PDF documentation reproduces the look and feel of the printed book, complete with fonts, graphics, formatting, and images. In the PDF document, use links from the table of contents, index, or within the document to go directly to the page of interest. Note that some documentation available from the Help browser is not available in PDF format.
Note  The Help browser accesses PDF documentation from The MathWorks Web site. Therefore, you need Internet access to view or print PDF documentation.

1  In the Help browser, click the Contents tab and select a product, for example, MATLAB.

   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.

   MATLAB accesses the selected document from The MathWorks Web site. Your PDF reader opens, displaying the documentation.

   For PDF printing problems on UNIX platforms, check the Help preferences. See “PDF Reader—Specifying Its Location” on page 4-28 for more information.

3  To print the documentation, select Print from the File menu in your PDF reader.
Help Functions

There are several help functions that provide forms of help different from the Help browser documentation, or provide alternative ways to access the Help browser information.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbtype</td>
<td>Displays the M-file with line numbers. If you want to see only the input and output arguments for a function, use <code>dbtype function 1</code>, which displays the first line of the M-file.</td>
</tr>
<tr>
<td>demo</td>
<td>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.</td>
</tr>
<tr>
<td>doc</td>
<td>Displays in the Help browser, the reference page for the specified function, block, or property. More extensive than <code>help</code> results, the reference page provides syntax, a description, examples, and links to related functions.</td>
</tr>
<tr>
<td>docopt</td>
<td>On UNIX systems, specifies Web browser information, used when displaying Internet Web pages.</td>
</tr>
<tr>
<td>docsearch</td>
<td>Run the Help browser search feature for the specified term.</td>
</tr>
<tr>
<td>help</td>
<td>Displays M-file help (a description and syntax) in the Command Window for the specified function or block.</td>
</tr>
<tr>
<td>helpbrowser</td>
<td>Opens the Help browser, the MATLAB interface for accessing documentation.</td>
</tr>
<tr>
<td>helpdesk</td>
<td>Opens the Help browser. In previous releases, <code>helpdesk</code> displayed the Help Desk, which was the precursor to the Help browser.</td>
</tr>
<tr>
<td>helpwin</td>
<td>Displays a list of all functions in the Help browser, and provides access to M-file help for the functions.</td>
</tr>
</tbody>
</table>
### View 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

```matlab
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 MATLAB 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

```matlab
doc set
```

displays the reference page for the MATLAB `set` function in the Help browser, and in the Command Window, displays a hyperlinked list of the `set` functions residing in other directories, such as

```matlab
database/set
```

which is the `set` function for the Database Toolbox. Click a link to go to that `set` reference page.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description (Continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lookfor</code></td>
<td>Displays in the Command Window a list and brief description of all functions whose brief description includes the specified keyword.</td>
</tr>
<tr>
<td><code>web</code></td>
<td>Opens the specified URL in the specified browser. Use <code>web</code> in your own M-files to display HTML documentation you create for your work.</td>
</tr>
<tr>
<td><code>whatsnew</code></td>
<td>Displays the Release Notes in the Help browser.</td>
</tr>
</tbody>
</table>
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 the Database Toolbox, type

`doc database/set`

**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 make them stand out from the rest of the text. When typing function names, however, use lowercase characters. Some functions for interfacing to Java do use mixed case; the M-file help accurately reflects that, and you should use mixed case when typing them.

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 MATLAB search path having that name, and displays a hyperlinked list of the overloaded functions at the end.

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. 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, type

    help database/set
Help for Classes and Methods

To get help for a method, use

```
help classname.methodname
```

where `classname` is the fully qualified class. If you do not know the fully qualified class for the method, use `class(obj)`. Here, `methodname` is of the same class as the object `obj`. This example shows how to get help for the `lsb` method of the `fi` class in the Fixed-Point Toolbox.

```matlab
a = fi(pi);
class(a)
ans =
    embedded.fi
```

The fully qualified class for `lsb` is `embedded.fi`. Therefore, use

```
help embedded.fi.lsb
```

to display help for the `lsb` method.

To get help for a class, specify the fully qualified class. For example, to get help for the `fi` class, use

```
help embedded.fi
```

Help for the class is actually the help for the class’s object constructor, in this case, `fi`.

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. See the `help` reference page for details.

Directory Reports for Help

The Help Report and the Contents Report provide other ways of looking at and managing help for M-files—see “Directory Reports in Current Directory Browser” on page 7-11.
Other Forms of Help

In addition to using the Help browser and help functions, there are the other forms of help for MATLAB and related products:

- “Documentation for Other Products” on page 4-37
- “Product-Specific Help Features” on page 4-37
- “User-Contributed M-Files” on page 4-38
- “Technical Support” on page 4-38
- “Newsgroup for MathWorks Products” on page 4-39
- “Other Resources for MATLAB Information” on page 4-39
- “Version and License Information” on page 4-40
- “Provide Feedback” on page 4-40

Documentation for Other Products

The Help browser provides access to documentation for all products installed on your system. You can view any product’s online documentation at the MathWorks Web site at http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml.

Product-Specific Help Features

In addition to the Help browser and help functions, some products and tools allow other methods for getting help. You will encounter some methods in the course of using a product, such as entries in the Help menu, Help buttons in dialog boxes, and selecting Help from a context menu. These methods all display context-sensitive help. Other methods for getting help, such as pressing the F1 key, are described in the documentation for the product or tool that uses the method.
User-Contributed M-Files
You can download M-files contributed by users and developers of MATLAB, Simulink, and related products from MATLAB Central. Before you write an M-file yourself, especially if it seems to be a more generic utility, check the list of contributed files to see if someone has already written it. These files are freely contributed and can be used without charge by anyone who downloads them. 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/index.jsp. You can access this in any desktop component via Help -> Web Resources.

If you write M-files that you think would be of use to others, consider submitting them to the MATLAB Central File Exchange via the Web page.

Technical Support
MathWorks Technical Support provides help for product problems:

- 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 for questions posed by users.

- Select Technical Support Knowledge Base from the Help -> Web Resources menu to go to the Technical Support Web page (http://www.mathworks.com/support). The page displays in your system’s default Web browser. You can find out about other types of information such as third-party books, ask questions, make suggestions, and report possible bugs.

- If you cannot access the Web site, you can e-mail Technical Support using the address support@mathworks.com. You must provide your license number to obtain support. It is helpful if you also provide your operating system and MATLAB version number. You can obtain all of this information by running the ver function or by selecting Help -> About.
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 from any desktop component.

First-time users to the newsgroup should read the newsgroup FAQ, linked to from that 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.

Other Resources for MATLAB Information

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 MATLAB users. More frequently, the MathWorks issues MATLAB Digest, an electronic bulletin consisting of technical notes, solutions, and timely announcements to the user community. See http://www.mathworks.com/company/newsletters/.
- Books—There are hundreds of MATLAB based books. 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, and the MathWorks offers training classes for MATLAB and other products. For details, see http://www.mathworks.com/company/events/.
- Mathtools.net—A technical computing Web portal with links to many resources for MATLAB users. See http://www.mathtools.net/.
Version and License Information

If you need the product version or license information, select About from the Help menu for that product. The version is displayed in an About dialog box. Click Show License in the dialog box to view license information. Note that the information displayed does not cover your specific license agreement. If the product does not have a Help menu, use the ver function. To see the license number for MATLAB, type license in the Command Window. See also the ver, version, and license functions.

Provide Feedback

To report problems or provide comments or suggestions to The MathWorks about the documentation and help features, use the form on the Web. To access the form, go to the Contents pane in the Help browser. Select Begin Here from the top of the contents listing. Scroll to the bottom of the Begin Here page and click the link Give Us Your Feedback. Or go directly to the form at http://www.mathworks.com/access/helpdesk/feedback/index.shtml. Alternatively, you can send e-mail to doc@mathworks.com.

To suggest enhancements or provide feedback about MathWorks products, send e-mail to suggest@mathworks.com. To report problems, send e-mail to bugs@mathworks.com or contact Technical Support at http://www.mathworks.com/support/.
Workspace, Search Path, and File Operations

If you are using the Help browser, watch the Workspace Browser video demo, the Array Editor video demo, and the Current Directory Browser video demo for an overview of the major functionality.

**MATLAB Workspace (p. 5-2)**

The workspace is the set of variables maintained in memory during a MATLAB session. Use the Workspace browser or equivalent functions to view the workspace.

**Viewing and Editing Workspace Variables with the Array Editor (p. 5-10)**

View and make changes to variables using the Array Editor.

**Search Path (p. 5-18)**

MATLAB uses a search path to find M-files and other MATLAB related files. View and change the path using the **Set Path** dialog box or equivalent functions.

**File Management Operations (p. 5-29)**

Search for, view, open, and make changes to MATLAB related directories and files, using the Current Directory browser or equivalent functions.
MATLAB Workspace

The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory. You add variables to the workspace by using functions, running M-files, and loading saved workspaces. For example, if you type
\[
t = 0:pi/4:2*pi;
y = sin(t);
\]
the workspace includes two variables, \( y \) and \( t \), each having nine values.

You can perform workspace operations and related features using the Workspace browser. Equivalent functions are available and are documented with each feature of the Workspace browser. If you are using the Help browser, watch the Workspace browser video demo for an overview of the major functionality:

- “Opening the Workspace Browser” on page 5-3
- “Viewing and Editing Values in the Current Workspace” on page 5-3
- “Saving the Current Workspace” on page 5-4
- “Loading a Saved Workspace and Importing Data” on page 5-6
- “Changing and Copying Variable Names” on page 5-7
- “Clearing Workspace Variables” on page 5-7
- “Viewing Base and Function Workspaces Using the Stack” on page 5-8
- “Creating Graphics from the Workspace Browser” on page 5-8
- “Opening Variables and Objects for Viewing and Editing” on page 5-8
- “Preferences for the Workspace Browser” on page 5-9.
Opening the Workspace Browser

To open the Workspace browser, select **Workspace** from the **Desktop** menu in the MATLAB desktop, or type `workspace` at the Command Window prompt.

The Workspace browser opens.

![Workspace Browser](image)

Viewing and Editing Values in the Current Workspace

The Workspace browser shows the name of each variable, its value, its array size, its size in bytes, and the class. The icon for each variable denotes its class.

To resize the columns of information, drag the column header borders. To show or hide any of the columns, or to specify the sort order, select **View -> Choose Columns**.

You can select the column on which to sort as well as reverse the sort order of any column. Click a column heading to sort on that column. Click the column heading again to reverse the sort order in that column. For example, to sort on **Name**, click the column heading once. To change from ascending to descending, click the heading again. You cannot sort by the **Value** column in the Workspace browser.

You can edit values directly in the Workspace browser **Value** column. To edit a value, select the row to change in the **Value** column and type in the new value.
Function Alternative

Use \texttt{who} to list the current workspace variables. Use \texttt{whos} to list the variables and information about their size and class. For example:

\begin{verbatim}
who
Your variables are:
 A   M   S   V
\end{verbatim}

\begin{verbatim}
whos
Name      Size  Bytes   Class
 A   1x4   32 double array
 M   3x1   202 cell array
 S   1x2   598 struct array
 V   1x35  70 char array
\end{verbatim}

Grand total is 76 elements using 902 bytes

Use \texttt{exist} to see if the specified variable is in the workspace.

Saving the Current Workspace

The workspace is not maintained across MATLAB sessions. 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 MATLAB specific binary file. 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.

\begin{itemize}
  \item \textbf{Note} The .mat extension is also used by Microsoft Access.
\end{itemize}
Saving All Variables
To save all of the workspace variables using the Workspace browser:

1 From the File menu, select Save Workspace As, or click the save button in the Workspace browser toolbar.
   The Save 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.
You can also save the workspace variables from the desktop by selecting Save Workspace As from the File menu.

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.
To specify preferences for saving MAT-files, see “MAT-Files” on page 2-56.
Function Alternative
To save workspace variables, use the `save` function followed by the filename you want to save to. For example,

```matlab
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 working 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`. For a related function, see `genvarname`. MATLAB provides additional functions for saving information—see the Import Wizard documentation.

Loading a Saved Workspace and Importing Data
To load saved variables into the workspace:

1. Click the load data file button on the toolbar in the Workspace browser. The **Open** 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. If any variables being loaded have the same names as variables in the current workspace, the values from the MAT-file replace the values in the current workspace. Any variables in the MAT-file that are not in the workspace are added to the workspace.

Function Alternative
Use `load` to open a saved workspace. For example,

```matlab
load('june10')
```

loads all workspace variables from the file `june10.mat`.  


Importing Data
MATLAB provides other methods and functions for loading information. You can use one of these methods, the Import Wizard, from the Workspace browser—select Edit -> Paste Special or use Ctrl+V to import data to MATLAB using the Import Wizard. For more information on the Import Wizard and other methods for loading information, see the Import Wizard documentation.

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 or Return.

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.

Clearing Workspace Variables
You can clear a variable, which removes it from the workspace.

To clear a variable using the Workspace browser:

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 may appear. If it does, click Yes to clear the variables.

The confirmation dialog box appears if you specify it as a preference. See “Preferences for the Workspace Browser” on page 5-9 to change the preference.
Function Alternative
Use the clear function to clear variables from the workspace. For example,

```matlab
clear A M
```

clears the variables A and M from the workspace.

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 MATLAB base workspace. To access the base and function workspaces when 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 Array Editor and the Editor/Debugger. See “Debugging M-Files” on page 6-33 for more information.

Creating Graphics from the Workspace Browser
From the Workspace browser, you can generate a graph of a variable. To create a graph, click the graph button on the Workspace browser toolbar and select the graph type. The graph appears in a figure window.

In addition, you can right-click the variable you want to graph. From the context menu, choose the type of graph you want to create. You can also select multiple variables to graph together.

For more information about creating graphs in MATLAB, see the Using MATLAB Graphics documentation.

Opening Variables and Objects for Viewing and Editing
In the Workspace browser, double-click a variable and it opens in the Array Editor, where you can view and edit the contents of the variable. See “Viewing and Editing Workspace Variables with the Array Editor” on page 5-10 for more information about opening arrays.

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
You can specify whether or not you want a confirmation dialog box to appear when you clear variables using the Workspace browser. Select File -> Preferences -> Workspace. If you want a confirmation dialog box to appear when you delete a variable, select the Show confirmation dialog when deleting variables check box.
Viewing and Editing Workspace Variables with the Array Editor

Use the Array Editor to view and edit a visual representation of one or two-dimensional numeric arrays, strings, cell arrays of strings, and structures. If you are using the Help browser, watch the Array Editor video demo for an overview of the major functionality.

The features of the Array Editor are

- “Opening the Array Editor” on page 5-10
- “Navigating in the Array Editor” on page 5-12
- “Changing Array Size, Content, and Format of Elements in the Array Editor” on page 5-12
- “Cut, Copy, Paste, and Delete in the Array Editor” on page 5-13
- “Exchanging Data with the Command Window” on page 5-16
- “Exchanging Data with Excel” on page 5-16
- “Creating Graphs and Variables from the Current Selection” on page 5-16
- “Preferences for the Array Editor” on page 5-16

Opening the Array Editor

You can open the Array Editor from the Workspace browser:

1. In the Workspace browser, select the variable you want to open. 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 on the toolbar. For one variable, you can also open it by double-clicking it.

   The Array Editor opens, displaying the values for the selected variable.
Repeat the steps to open additional variables in the Array Editor. Access each variable via its tab at the bottom of the window, or use the **Window** menu.

**Function Alternatives**

To open a variable in the Array Editor, use `openvar` with the name of the variable you want to open as the argument. For example, type

```matlab
openvar('M')
```

MATLAB opens M in the Array Editor.

To see the contents of a variable in the workspace, just type the variable name at the Command Window prompt. For example, type

```matlab
M
```

and MATLAB returns

```
M =
    'one'
    'two'
    'three'
```
Navigating in the Array Editor

Use the following keys to move among elements in the Array Editor. Navigating in the Array Editor is much like navigating in Microsoft Excel.

<table>
<thead>
<tr>
<th>Key</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>Commit any changes to the element and move to next element, where next element is specified using “Preferences for the Array Editor” on page 5-16 (default is down)</td>
</tr>
<tr>
<td>Tab</td>
<td>Move right</td>
</tr>
<tr>
<td></td>
<td>Within a selection, also moves from the last column to the first column in the next row</td>
</tr>
<tr>
<td>Shift+Enter or Shift+Tab</td>
<td>Move in opposite direction of Enter or Tab</td>
</tr>
<tr>
<td>Page Up</td>
<td>Move up m rows, where m is the number of visible rows</td>
</tr>
<tr>
<td>Page Down</td>
<td>Move down m rows, where m is the number of visible rows</td>
</tr>
<tr>
<td>Home</td>
<td>Move to column 1</td>
</tr>
<tr>
<td>Ctrl+Home</td>
<td>Move to row 1, column 1</td>
</tr>
<tr>
<td>Shift+Home</td>
<td>Select to column 1</td>
</tr>
<tr>
<td>End</td>
<td>Move to last column in current row</td>
</tr>
</tbody>
</table>

Changing Array Size, Content, and Format of Elements in the Array Editor

To increase the size of an array, scroll to the desired location in the array and enter a value. The array will automatically expand to accommodate the new value. Empty cells are filled 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 to select the entire row, right-clicking, and selecting Delete.
To change the value of an element in the Array Editor, click in that element and type a new value. Press **Enter** or **Return**, or click in another element to make the change take effect. You can specify where the cursor moves to after you press **Enter**—see “Preferences for the Array Editor” on page 5-16.

If you want to change the display format for the Array 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 Array Editor” on page 5-16.

If you opened an existing MAT-file and made changes to it using the Array Editor, save that MAT-file if you want the changes to be saved. For instructions, see “Saving the Current Workspace” on page 5-4.

**Cut, Copy, Paste, and Delete in the Array 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 in the row or column heading (the element that shows the row or column number). **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.

When you cut elements, the value of each element you cut becomes **0** if numeric or `[]` if a character array. After cutting, select the elements whose value you want to replace with the cut elements and then choose **Paste**. If the shape of the elements you cut differs from the shape of the elements into which you are pasting, the Array 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**.
Example Copying and Pasting Array Elements
In this example, two elements are copied but the selected area for pasting is only one element, so the Array Editor expands the selected area for pasting.
Example Cutting and Pasting Array Elements

In this example, the area selected for pasting requires the Array Editor to expand the array size in order for all cut elements to be pasted.

Two elements are selected and cut.

The values of the elements are cleared.

One element is selected as the paste area.

The Array Editor adds a row so that both cut elements can be pasted.
Deleting Elements, Rows and Columns

You can clear elements, rows, or columns in the array by selecting them and then selecting **Delete** from the **Edit** menu or context menu. When you delete cells, a dialog box appears asking how you want the remaining cells to shift.

Exchanging Data with the Command Window

You can copy data from the Array 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 Array Editor. Be sure the data types are compatible. For example, you cannot paste text from the Command Window into a numeric array in the Array Editor.

Exchanging Data with Excel

You can cut or copy cells from Microsoft Excel and paste them into the Array Editor. You can also cut or copy elements from the Array Editor and paste them into Excel.

Be sure the data types are compatible. For example, you cannot paste text from Excel into a numeric array in the Array Editor.

Creating Graphs and Variables from the Current Selection

You can create graphs and new variables from the Array Editor. To create a graph, select a cell, row, or column, and in the right-click context menu, choose the graph type. To create a new variable, select a cell, row, or column in the Array Editor, right-click, and from the context menu, select **Create Variable from Selection**.

Preferences for the Array Editor

To set preferences for the Array Editor, select **Preferences** from the **File** menu. The **Preferences** dialog box opens showing **Array Editor Preferences**.

Default Format

Specify the output format of numeric values displayed in the Array 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 in 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**.
Search Path

This section covers the following topics:

- “About the Search Path” on page 5-18
- “How the Search Path Determines Which Function to Use” on page 5-19
- “How MATLAB Finds the Search Path, pathdef.m” on page 5-20
- “Viewing and Setting the Search Path” on page 5-20
- “Using the Path in Future Sessions” on page 5-26
- “Recovering from Problems with the Search Path” on page 5-27

About the Search Path

MATLAB uses a search path to find M-files and other MATLAB related files, which are organized in directories on your file system. By default, the files supplied with MATLAB and MathWorks products are included in the search path. These are all of the directories and files under $matlabroot/toolbox.

Any file you want to run in MATLAB must reside in a directory that is on the search path, or in the current directory. If you create any MATLAB related files, add the directories containing the files to the MATLAB search path. For instructions to view the search path and add directories to it, see “Viewing and Setting the Search Path” on page 5-20, including “Caution Against Saving Files in $matlabroot/toolbox” on page 5-25.

The search path is also referred to as the MATLAB path. Directories included are considered to be on the path. When you include a directory in the search path, you add it to the path. Subdirectories must be explicitly added to the path; they are not on the path just because their parent directories are.
How the Search Path Determines Which Function to Use

The order of directories on the path is relevant. MATLAB looks for a named element, for example, `foo`, as described here. If you enter `foo` at the MATLAB prompt, MATLAB performs the following actions:

1. Looks for `foo` as a variable.
2. Looks in the current directory for a file named `foo.m`.
3. Searches the directories on the MATLAB search path, in order, for `foo.bi` (built-in function) or `foo.m`.

If there is more than one function with the same name, the order of directories on the path determines which of those functions MATLAB uses. When MATLAB looks for that function, it uses the first one found in the search path:

- To use a function with the same name that is located in a directory further down on the search path, called a *shadowed* function, make its location the current directory. 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.
- If you are not sure of the function MATLAB is using, run `which` for a specified function and MATLAB returns the full path to the function.

Although the actual search path rules are more complicated because of the restricted scope of private functions, subfunctions, and object-oriented functions, this simplified perspective is accurate for the ordinary M-files you usually work with. For more information, see “Determining Which Function Is Called” in MATLAB Programming documentation.
How MATLAB Finds the Search Path, pathdef.m
The search path is stored in the file pathdef.m, which by default, is located in $matlabroot/toolbox/local. You can store it in the MATLAB startup directory, and modify it for the current session or for all future sessions.

When MATLAB starts, it looks for a pathdef.m file in its startup directory. If none is found, it uses pathdef.m in $matlabroot/toolbox/local. MATLAB modifies the path based on any path statements in a startup.m file. During a session, you can save changes to the path using the Set Path dialog box or the savepath function, and MATLAB uses the path you saved to for the remainder of the session. If MATLAB finds a pathdef.m in the current directory, it uses that version instead. To avoid problems, do not maintain a pathdef.m file in a directory other than the MATLAB startup directory or $matlabroot/toolbox/local.

Viewing and Setting the Search Path
Use the Set Path dialog box to view and modify the MATLAB search path. Equivalent functions are documented for each feature of the Set Path dialog box. Select Set Path from the File menu, or type pathtool at the Command Window prompt. The Set Path dialog box opens.
Make changes to the search path.

Directories on the current MATLAB search path.

Save changes for use in future MATLAB sessions.

Use the changes for the current session, but do not save the changes for use in future MATLAB sessions.

Replace current path with all directories installed with MATLAB and related products.
Use the **Set Path** dialog box for the following actions. Equivalent functions are listed as well:

- “Viewing the Search Path” on page 5-22
- “Adding Directories to the Search Path” on page 5-22
- “Moving Directories Within the Search Path” on page 5-23
- “Removing Directories from the Search Path” on page 5-24
- “Restoring the Default Search Path” on page 5-24
- “Reverting to the Previous Path” on page 5-25
- “Saving Settings to the Path” on page 5-25

See also

- “About the Search Path” on page 5-18 for background information
- “Using the Path in Future Sessions” on page 5-26 for options
- “Recovering from Problems with the Search Path” on page 5-27

**Viewing the Search Path**

The **MATLAB search path** field in 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.

**Function Alternative.** Use the `path` function to view the search path.

**Adding Directories to the Search Path**

Add directories to the search path when you want to run M-files in those directories.

To add directories to the MATLAB search path using the **Set Path** dialog box:

1. Click the **Add Folder** or the **Add with Subfolders** button.
   - If you want to add only the selected directory but do not want to add all of its subdirectories, click **Add Folder**.
   - If you want to add the selected directory and all of its subdirectories, click **Add with Subfolders**.

The **Browse for Folder** dialog box opens.
2 In the **Browse for Folder** dialog box, use the view of your file system to select the directory to add, and then click **OK**.

The selected directory, and subdirectories if specified in step 1, are added to the top of the search path.

3 To use the newly modified search path in future sessions, click **Save**. For more information about saving the path, see “Saving Settings to the Path” on page 5-25.

4 Click **Close**. If you did not save the changes in the previous step, the directories you added remain on the search path until you end the current MATLAB session.

You cannot add method directories (directories that start with @) and private directories directly to the search path. Instead, add their parent directories using the **Add with Subfolders** button.

**Adding Directories to the Path from the Current Directory Browser.** In the Current Directory browser, select the 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.** To add directories to the top or the end of the search path, use **addpath**. The **addpath** function offers an option to get the path as a string and to concatenate multiple strings to form a new path.

You can include **addpath** statements in your **startup** M-file to automatically modify the path when MATLAB starts. For details, see “Modifying the Path in a startup.m File” on page 5-26.

**Moving Directories Within the Search Path**

The order of files on the search path is relevant—for more information, see “How the Search Path Determines Which Function to Use” on page 5-19.

To modify the order of directories within the search path:

1 Select the directory or directories you want to move.

2 Click one of the **Move** buttons, such as **Move to Top**. The order of the directories changes.
To use the newly modified search path in future sessions, click **Save**. For more information about saving the path, see “Saving Settings to the Path” on page 5-25.

Click **Close**. If you did not save the changes in the previous step, the new order of files on the search path remains in effect until you end the current MATLAB session.

**Function Alternative.** While there is not a specific function to move directories, you can edit the `pathdef.m` file with any text editor to change the order of the directories. Use caution when editing the file so that you do not make MATLAB and toolbox functions unusable.

### Removing Directories from the Search Path

To remove directories from the MATLAB search path using the **Set Path** dialog box:

1. Select the directories to remove.
2. Click **Remove**. The directories are removed from the path.
3. To use the newly modified search path in future sessions, click **Save**. For more information about saving the path, see “Saving Settings to the Path” on page 5-25.
4. Click **Close**. If you did not save the changes in the previous step, the directories are removed from the search path until you end the current MATLAB session.

**Function Alternative.** To remove directories from the search path, use `rmpath`.

You can include `rmpath` statements in your **startup** M-file to automatically modify the path when MATLAB starts. For details see “Modifying the Path in a startup.m File” on page 5-26.

### Restoring the Default Search Path

To restore the default search path, click **Default** in the **Set Path** dialog box. This changes the search path so that it includes only the directories installed with MATLAB and related products.
Reverting to the Previous Path
To restore the previous path, click Revert in the Set Path dialog box. This cancels any unsaved changes you have made in the Set Path dialog box.

Saving Settings to the Path
When you make changes to the search path, they remain in effect during the current MATLAB session. To keep the changes in effect for subsequent sessions, you need to save them. To save changes using the Set Path dialog box, click Save.

If you want to automatically use this search path in future sessions, save the path to your MATLAB startup directory, which saves pathdef.m to that location. You can save the changes to the default pathdef.m file, in $matlabroot/toolbox/local if you have write permission for that directory but see the following caution. Alternatively, you can include addpath and rmpath statements in a startup.m file, which avoids some problems you might have with saving the path, for example, using the same path with both Windows and UNIX platforms. For more information, see “Using the Path in Future Sessions” on page 5-26.

Caution Against Saving Files in $matlabroot/toolbox. Save any M-files you create and any MathWorks-supplied M-files 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 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 MATLAB session 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 MATLAB” on page 1-10.

Function Alternative. Use savepath to save the current path to pathdef.m. Locate pathdef.m in your MATLAB startup directory to automatically use it in future sessions. Consider using savepath in your finish.m file. To modify the default path upon startup, include addpath and rmpath functions in your startup.m file. For more information, see “Modifying the Path in a startup.m File” on page 5-26.
Using the Path in Future Sessions

There are three basic ways for MATLAB to automatically use a search path you specify, each with advantages and disadvantages:

- “Modifying the Path in a startup.m File” on page 5-26
- “Saving the Path in the MATLAB Startup Directory” on page 5-26
- “Saving the Path in $matlabroot/toolbox/local” on page 5-27

For background information, see “How MATLAB Finds the Search Path, pathdef.m” on page 5-20.

Modifying the Path in a startup.m File

Put addpath and rmpath statements in a startup.m file, and include the startup file in MATLAB's startup directory. When MATLAB starts, it uses the search path defined in pathdef.m in $matlabroot/toolbox/local and modifies it based on the commands in the startup.m file.

By maintaining an unaltered pathdef.m in $matlabroot/toolbox/local, you avoid inadvertently removing directories supplied by The MathWorks from the path. This method continues working even when you update to a new version of MATLAB. If you run MATLAB on both Windows and UNIX platforms, this method works well—for example, for each platform, include separate addpath sections in the startup.m file, with each section preceded by an ispc or isunix statement.

One disadvantage of this method is that changes you make to the path using the Set Path dialog box are not incorporated in the startup.m file.

Saving the Path in the MATLAB Startup Directory

Copy pathdef.m from $matlabroot/toolbox/local to the MATLAB startup directory. Make changes to the path using the Set Path dialog box, and with addpath and rmpath functions—choose whichever suits your needs. You can use this method if you do not have write access to $matlabroot/toolbox/local.
There are some disadvantages to this method. You might inadvertently remove
directories supplied by The MathWorks from the path. When you update to a
new version of MATLAB, you cannot use the pathdef.m file in the startup
directory, but must delete it and create a new version. If you run MATLAB on
both Windows and UNIX platforms, you need to maintain a separate
pathdef.m file for each.

**Saving the Path in $matlabroot/toolbox/local**

If you have write access to $matlabroot/toolbox/local, make and save
changes to the path using the Set Path dialog box, and with addpath and
rmpath functions—choose whichever suits your needs.

There are some disadvantages to this method. You cannot maintain this file
when you update to a new version of MATLAB, but will need to use the new
default pathdef.m and make changes to it. If you run MATLAB on both
Windows and UNIX platforms, you need to maintain a separate pathdef.m file
for each.

**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 you edit the pathdef.m file directly and make it invalid, or if the file
becomes corrupt, 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. View the `pathdef.m` and `startup.m` files, 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.

2. Use the default path for MathWorks products. In 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.

3. Run `restoredefaultpath`. This sets the search path to include only installed products from the MathWorks. 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 4.

4. Perform these steps after trying step 3.
   
   a. Run
      
   ```plaintext
   restoredefaultpath; matlabrc
   ```

      This might run for a few minutes. It sets the search path to include only 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 the with a good `pathdef.m` file, for example, one you can generate at this point with

   ```plaintext
   savepath('path_to_your_startup_directory/pathdef.m')
   ```

   c. Start MATLAB again to be sure the problem does not reappear.
File Management Operations

MATLAB file operations use the current directory and the MATLAB search path as reference points. Any file you want to run must either be in the current directory or on the search path. The key tools for performing file operations are

- “Current Directory Field” on page 5-30
- “Current Directory Browser” on page 5-30
- “Viewing and Making Changes to Directories” on page 5-32
- “Creating, Renaming, Copying, and Removing Directories and Files” on page 5-33
- “Opening, Running, and Viewing the Content of Files” on page 5-37
- “Finding Files and Content Within Files” on page 5-40
- “Accessing Source Control Features” on page 5-42
- Setting “Preferences for the Current Directory Browser” on page 5-42

**Note** You generally cannot perform operations on files and directories for which you do not have proper permission. For example, you cannot copy a file to a read-only directory using the Current Directory browser. You can do so using movefile.
**Current Directory Field**

A quick way to view or change the current directory is by using the current directory field in the desktop toolbar.

![Current Directory Field](image)

To change the current directory from this field, do one of the following:

- In the field, type the path for the new current directory.
- Click the down arrow to view a list of previous working directories, and select an item from the list to make that directory become the MATLAB current working directory. The directories are listed in order, with the most recently used at the top of the list. You can clear the list and set the number of directories saved in the list—see “Preferences for the Current Directory Browser” on page 5-42.
- Click the browse button (…) to set a new current directory.
- Use the up button to move the current directory up one level.

The current directory field in the desktop also appears in the Current Directory browser, when the Current Directory browser is undocked. Consider it to be one tool with two different means of accessing it.

**Current Directory Browser**

To search for, view, open, find, and make changes to MATLAB related directories and files, use the MATLAB Current Directory browser. Most features of the Current Directory browser have equivalent functions that perform similar actions. If you are using the Help browser, watch the Current Directory Browser video demo for an overview of the major functionality.

In addition to the features described here, the Current Directory browser includes tools to help you manage your M-files—see the “Visual Directory in Current Directory Browser” on page 7-2 and “Directory Reports in Current Directory Browser” on page 7-11.

To open the Current Directory browser, select Desktop -> Current Directory from the MATLAB desktop, or type `filebrowser` at the Command Window prompt. The Current Directory browser opens.
The main tasks you perform with the Current Directory browser are

- “Viewing and Making Changes to Directories” on page 5-32
- “Creating, Renaming, Copying, and Removing Directories and Files” on page 5-33
- “Opening, Running, and Viewing the Content of Files” on page 5-37
- “Finding Files and Content Within Files” on page 5-40
- “Accessing Source Control Features” on page 5-42
- Setting “Preferences for the Current Directory Browser” on page 5-42
Viewing and Making Changes to Directories
You can change the current directory, view its contents, add directories to the MATLAB search path, and change the way the Current Directory browser presents entries.

Changing the Current Working Directory and Viewing Its Contents
To change the current directory, use the current directory field. The Current Directory browser lists the files and directories in the current directory.

To view the contents of a subdirectory, double-click it, or select the subdirectory and press Enter or Return.

To move up one level in the directory structure, press the backspace key.

Function Alternative. Use dir to view the contents of the current working directory or another specified directory.

Use what to see only the MATLAB related files in a directory. With no arguments, what displays the MATLAB related files in the current working directory. Use which to display the pathname for the specified function. Use exist to see if a directory or file exists. Use fileattrib to see or set file attributes, much like attrib in DOS or chmod in UNIX.

Adding Directories to the MATLAB Search Path
From the Current Directory browser, you can add directories to the MATLAB search path. Right-click and from the context menu, select Add to Path. Then select one of the options:

• **Current Directory**—Adds the current directory to the path.

• **Selected Folders**—Adds the directory selected in the Current Directory browser to the path.

• **Selected Folder and Subfolders**—Adds the directory selected in the Current Directory browser to the path, and adds all of its subdirectories to the path.
Changing the Display
To specify the types of files shown in the Current Directory browser, use the View menu. For example, you can show only M-files. If All Files is selected and you want to see specific file types, first clear the selection for All Files and then select the specific file types.

You can sort the information shown in the Current Directory browser by column. Click the title of column on which you want to sort. The display is sorted, with the information in the that column shown in ascending order, and an up arrow icon indicates the direction. Click a second time on the column title to sort the information in descending order.

Creating, Renaming, Copying, and Removing Directories and Files

General Notes
If you have write permission, you can create, copy, remove, and rename MATLAB related files and directories for the directory shown in the Current Directory browser. If you do not have write permission, you can still copy files and directories to another directory, or you can use equivalent functions, such as movefile.

To run functions whose arguments require the use of a pathname or filename, use the function form rather than the unquoted or command form of the syntax when the pathname or filename includes spaces. For example, the command form

```
delete my file.m
```

generates a warning and does not delete myfile.m. Instead use the function form of the syntax.

```
delete('my file.m')
```
Creating New Files
To create a new file in the current directory:

1. Select New from the context menu or File menu and then select the type of file to create.

   An icon for that file type, for example, an M-file icon, with the default name Untitledn appears at the end of the list of files shown in the Current Directory browser.

2. Type over Untitledn with the name you want to give to the new file.

3. Press Enter or Return.

   The file is added.

4. To enter the contents of the new M-file, open the file—see “Opening, Running, and Viewing the Content of Files” on page 5-37. If you created the file using the context menu, the new file opens in the Editor with a template for writing an M-file function.

   Function Alternative. Use the edit function to create a new M-file or other type of text file in the Editor/Debugger.

Creating New Directories
To create a new directory in the current directory:

1. Click the new folder button in the Current Directory browser toolbar, or select New -> Folder from context menu.

   An icon, with the default name NewFolder appears at the end of the list of files shown in the Current Directory browser.

2. Type over NewFolder with the name you want to give to the new directory.

3. Press the Enter or Return key.

   The directory is added.
Function Alternative. To create a directory, use the `mkdir` function. For example,

```plaintext
mkdir newdir
```

creates the directory `newdir` within the current directory.

Renaming Files and Directories

To rename a file or directory, select the item, right-click, and select Rename from the context menu. 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.

Function Alternative. You can use `movefile` to rename a file or directory. For example,

```plaintext
movefile('myfile.m','projectresults.m')
```

renames `myfile.m` to `projectresults.m`.

Cutting or Deleting Files and Directories

To cut or delete files and directories:

1. Select the files and directories to remove. Use Shift+click or Ctrl+click to select multiple items.

2. Right-click and select Cut or Delete from the context menu.

The files and directories are removed.

Files and directories you delete from the Current Directory browser go to the Recycle Bin on Windows (or the Trash Can on Macintosh platforms). If you do not want the selected items to go to the Recycle Bin, press Shift+Delete. A confirmation dialog box displays before the items are deleted if you have set that option in your operating system. For example, on Windows, 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.

Function Alternative. To delete a file, use the `delete` function. For example,

```plaintext
delete('d:/mymfiles/testfun.m')
```

deletes the file `testfun.m`. You can recover deleted files if you use the preference described in “Default Behavior of the Delete Function” on page 2-55 or the `recycle` function.

5-35
To delete a directory and optionally its contents, use `rmdir`. For example,

```
    rmdir('myfiles')
```

removes the directory `myfiles` from the current directory.

**Copying and Pasting Files and Directories**

Using the Current Directory browser, you can copy (or cut) and paste files and directories:

1. Select the files or directories to copy. Use Shift+click or Ctrl+click to select multiple items. 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.

You can also copy and paste files and directories to and from tools outside of MATLAB, such as Windows Explorer. You can use Current Directory browser menu items and the keyboard shortcuts, or you can drag the items.

**Function Alternative.** Use `movefile` or `copyfile` to cut and paste or to copy and paste files or directories. 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')
```
Opening, Running, and Viewing the Content of Files

Opening Files
You can open a file using the open features of the Current Directory browser. The file opens in the tool associated with that file type.

Select one or more files and perform one of the following actions:

- Press the Enter or Return key.
- Right-click and select Open from the context menu.
- Double-click the file(s).

The file opens in the appropriate tool. For example, the Editor/Debugger opens for M-files, and Simulink opens for model (.mdl) files.

To open a file in the Editor/Debugger, no matter what type it is, select Open as Text from the context menu. One exception is P-files (.p), which you cannot open.

To open a file using an external application, select Open Outside MATLAB from the context menu. For example, if you select myfile.doc, Open Outside MATLAB opens myfile.doc in Microsoft Word, assuming you have the .doc file association configured to start Word.

You can also import data from a file. Select the file, right-click, and select Import Data from the context menu. The Import Wizard opens. See the Import Wizard documentation for instructions to import the data.
Function Alternative. Use the open function to open a file in the tool appropriate for the file, given its file extension. Default behavior is provided for standard MATLAB file types. You can extend the interface to include other file types and to override the default behavior for the standard files. For name.ext, open performs the following actions.

<table>
<thead>
<tr>
<th>File Type</th>
<th>Extension</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure file</td>
<td>fig</td>
<td>Opens figure name.fig in a figure window.</td>
</tr>
<tr>
<td>HTML file</td>
<td>html</td>
<td>Opens HTML file name.html in the MATLAB Web browser.</td>
</tr>
<tr>
<td>M-file</td>
<td>m</td>
<td>Opens M-file name.m in the Editor.</td>
</tr>
<tr>
<td>MAT-file</td>
<td>mat</td>
<td>Opens MAT-file name.mat in the Import Wizard.</td>
</tr>
<tr>
<td>Model</td>
<td>mdl</td>
<td>Opens model name.mdl in Simulink.</td>
</tr>
<tr>
<td>P-file</td>
<td>p</td>
<td>Cannot open P-files.</td>
</tr>
<tr>
<td>PDF file</td>
<td>pdf</td>
<td>Opens the PDF file name.pdf in the installed PDF reader, for example, Adobe Acrobat.</td>
</tr>
<tr>
<td>Variable</td>
<td>none</td>
<td>Opens the numeric or string array name in the Array Editor; open calls openvar.</td>
</tr>
<tr>
<td>Other</td>
<td>custom</td>
<td>Opens name.custom by calling the helper function opencustom, where opencustom is a user-defined function.</td>
</tr>
</tbody>
</table>

Use winopen to open a file using an external application on Windows platforms.

To view the content of an ASCII file, such as an M-file, use the type function. For example

```matlab
type('startup')
```

displays the contents of the file startup.m in the Command Window.
Running M-Files
To run an M-file from the Current Directory browser, select it, right-click, and select **Run** from the context menu. The results appear in the Command Window.

To get the name of the currently running M-file, use `mfilename`.

Viewing Help for an M-File
You can view help for the M-file selected in the Current Directory browser. From the context menu, select **View Help**. The reference page for that function appears in the Help browser, or if a reference page does not exist, the M-file help appears.

You can view the M-file help in the Current Directory browser—for instructions, see “Preferences for the Current Directory Browser” on page 5-42.
Finding Files and Content Within Files

Use the Find Files tool to search for files or for specified text within files.

To search for files in one or more directories, or to search for specified text in files, follow these instructions:

1. Open the Find Files tool by clicking the find files button in the Current Directory browser toolbar, or selecting Edit -> Find Files from the desktop Edit menu.

   The Find Files dialog box opens.
2 Type in the filename or text you are searching for:
   • To search for files, type the filename in the **Find files named** field. You can use the wildcard character (*) in the filename. For example, type `coll*` to search for filenames that start with `coll`.
   • To search for text, type the text in the **Find files containing text** field. For example, search for the `plot`. Do not use a wildcard character, but instead use the **Advanced Options Search type** (see step 4).
   • To search for text in specified filenames only, type entries in both fields.

Use the dropdown list to view and select previous entries from the current MATLAB session.

3 From the **Look in** list box, select the directories to search in. You can instead type the full pathname for one or more directories into this field (with each pathname separated by a semicolon ;), or select the **Browse** option at the end of the list box. To include subdirectories in the search, select the **Include subdirectories** check box.

   For example, search the MATLAB current directory.

4 To specify additional search criteria, click **Advanced Options**.

   For example, select `*.m` in the **In file type** field to limit the search to M-files only. The default, `*`, searches all file types.

   By default, a text search looks for partial matches (the **Contains text** option for **Search type**). You can instead select **Matches whole word**.

5 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 panel on the right side of **Find Files** dialog box, with a summary of the results at the bottom of the panel. For text searches, the line number and line of code are shown. To see the full pathnames for the files, select the **Show full pathnames** check box under **Advanced Options**.
6. Click a column heading to sort the results based on that column. Click the
column heading again to reverse the sort order for that column. For
example, click **Line** to sort results by line number.

You can open any M-files that appear in the results list by doing one of the
following:

- Double-click the file
- Select the files and press **Enter** or **Return**
- Right-click and select **Open** from the context menu

The M-files open in the Editor/Debugger. For text searches, the file opens
scrolled to the line number shown in the results section of the **Find Files** dialog
box. Once in the Editor/Debugger, you can use the **Find & Replace** tool to
replace specified text.

To see the results of a previous search, select its tab at the bottom of the results
panel.

**Function Alternative.** Use **lookfor** to search for the specified text in the first line
of help for all M-files on the search path.

**Accessing Source Control Features**

Select a file or files in the Current Directory browser and right-click to view the
context menu. From there you can access features for Source Control. For
details on these features, see Chapter 9, “Source Control”.

**Preferences for the Current Directory Browser**

Using preferences, you can specify the number of recently used current
directories to maintain in the history list as well as the type of information to
display in the Current Directory browser.

From the Current Directory browser **File** menu, select **Preferences**. The
**Current Directory Preferences** pane appears in the **Preferences** dialog box.
History

The dropdown list in the current directory field shows the history of current directories, that is, the most recently used current directories.

Saving Directories. When the MATLAB session ends, the list of directories will be maintained. Use the Save most recent directories field to specify how many directories will appear on the list at the start of the next MATLAB session.

Removing Directories. To remove all entries in the list, click Clear History. The list is cleared immediately.
Browser Display Options

In the Current Directory browser, you can view the file type, last modified date, M-file descriptions, and M-file comments and MAT-file contents by selecting the appropriate Browser display options.

General

By default, the Auto-refresh directory view check box is selected. Changes you make to file and directory names in the current directory using other applications are immediately reflected in the Current Directory browser.

If you do not want the Current Directory browser to immediately reflect the changes, clear this preference. You can then right-click and select Refresh from the context menu to update the display.
MATLAB provides powerful tools for creating, editing, and debugging files, as detailed here. For information about the MATLAB language and writing M-files, see the MATLAB Programming documentation.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin with Existing Code (p. 6-2)</td>
<td>Before you begin writing MATLAB programs, consider starting with existing code, and then modify that code using the Editor/Debugger. Code resources include your own Command Window and History, and existing M-files, demos, and examples.</td>
</tr>
<tr>
<td>Ways to Edit and Debug Files (p. 6-4)</td>
<td>You can use the MATLAB Editor and Debugger with MATLAB, use the Editor without MATLAB, use another editor you already have, and use debugging functions in the Command Window. In the Editor/Debugger, you can edit M-files as well as other file types.</td>
</tr>
<tr>
<td>Starting, Customizing, and Closing the Editor/Debugger (p. 6-6)</td>
<td>Create new files, open existing files, open files without starting MATLAB, arrange document windows, and set preferences.</td>
</tr>
<tr>
<td>Creating, Editing, and Running Files (p. 6-13)</td>
<td>Control the appearance of files during editing, navigate in files, run M-files, and save, print, and close files.</td>
</tr>
<tr>
<td>Debugging M-Files (p. 6-33)</td>
<td>Find errors and debugging features.</td>
</tr>
<tr>
<td>Rapid Code Iteration Using Cells (p. 6-63)</td>
<td>Readily experiment with changes to your M-files by modifying and running sections of the files, called cells, and by incrementally modifying values.</td>
</tr>
</tbody>
</table>
Begin with Existing Code

Before you begin writing MATLAB code in a blank file, consider starting by using existing resources for the code, and then use the Editor/Debugger to modify the code. This section presents some resources to draw upon.

Create M-Files from Command Window and History
In many cases, 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/Debugger 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 code that accomplishes what you want to do, copy it and use it in your own M-file, assuming you have legal permission to do so. Here are some resources.

MATLAB and Toolbox M-Files
You can access and reuse the code in MATLAB and toolbox functions that have a .m file extension. You cannot use MATLAB and toolbox functions that have a .bi file extension (for built-in), meaning 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/Debugger 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 6-29 for details.

Demos and Examples
MATLAB 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 panel. For more information about demos, see “Demos in the Help Browser” on page 4-23.
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 panel, click + for the 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, Simulink, 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/index.jsp. You can access this via the Help -> Web menu in any desktop component.
## Ways to Edit and Debug Files

There are several methods for creating, editing, and debugging files with MATLAB as described in this table.

<table>
<thead>
<tr>
<th>Task</th>
<th>Option</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating and editing M-files</td>
<td>M-files in MATLAB Editor/Debugger</td>
<td>“Starting, Customizing, and Closing the Editor/Debugger” on page 6-6, and “Creating, Editing, and Running Files” on page 6-13</td>
</tr>
<tr>
<td></td>
<td>M-files in MATLAB Editor in stand-alone mode (without running MATLAB)</td>
<td>“Opening the Editor Without Starting MATLAB” on page 6-10</td>
</tr>
<tr>
<td></td>
<td>M-files in any text editor, such as Emacs or vi</td>
<td>To specify another editor as the default for use with MATLAB, select <strong>File -&gt; Preferences -&gt; Editor/Debugger</strong>, and for <strong>Editor</strong>, specify the <strong>Text editor</strong>. Click <strong>Help</strong> in the <strong>Preference</strong> panel for details. You can still debug the M-files using the MATLAB Editor/Debugger or debugging functions.</td>
</tr>
<tr>
<td>Creating and editing other file types</td>
<td>Other file types in the MATLAB Editor</td>
<td>Create, open, edit, and save other file types in the MATLAB Editor. See “Working with Other Text File Types” on page 6-12.</td>
</tr>
<tr>
<td>Debugging M-files</td>
<td>General debugging tips</td>
<td>“Finding Errors in M-Files” on page 6-33</td>
</tr>
<tr>
<td></td>
<td>MATLAB Debugger</td>
<td>“Debugging Process and Features” on page 6-39</td>
</tr>
<tr>
<td></td>
<td>MATLAB debugging functions (for use in the Command Window)</td>
<td>“Debugging Process and Features” on page 6-39</td>
</tr>
</tbody>
</table>
Use preferences for the Editor/Debugger to set up the editing and debugging environment to best meet your needs. For information about the MATLAB language and writing M-files, see the MATLAB Programming documentation.
Starting, Customizing, and Closing the Editor/Debugger

The MATLAB Editor/Debugger provides a graphical user interface for basic text editing features for any file type, as well as for M-file debugging. The Editor/Debugger is a single tool that you can use for editing, debugging, or both.

There are various ways to start the Editor/Debugger. The Editor/Debugger automatically starts when you open a document or create a new one, as detailed in these sections:

- “Creating a New File in the Editor/Debugger” on page 6-7
- “Opening Existing Files in the Editor/Debugger” on page 6-8
- “Opening the Editor Without Starting MATLAB” on page 6-10 (no Debugger)

After starting the Editor/Debugger, follow the instructions for

- “Arranging Editor/Debugger Documents” on page 6-10
- “Preferences for the Editor/Debugger” on page 6-11
- “Working with Other Text File Types” on page 6-12
- “Closing the Editor/Debugger” on page 6-12

This figure shows an example of the Editor/Debugger outside of the desktop opened to an existing M-file, and calls out some of the tool’s useful features.
Creating a New File in the Editor/Debugger

To create a new text file in the Editor/Debugger, either click the new file button on the MATLAB desktop toolbar, or select File -> New -> M-File from the MATLAB desktop. The Editor/Debugger opens, if it is not already open, with an untitled file in the MATLAB current directory, in which you can create an M-file or another type of text file.

The location of the new file and the Editor/Debugger are determined by document positioning guidelines. You can rearrange the documents to suit your needs. For details, see “Opening and Arranging Documents” on page 2-7.
If the Editor/Debugger is open, create more new files by using the new file button □ on the toolbar, or select **File -> New -> M-File**.

Other tools also provide features for creating new M-files. For example, in the Command History, select statements, right-click, and select **Create M-File** from the context menu. Similarly, create a new file from the context menu in the Current Directory browser—see “Creating New Files” on page 5-34.

**Function Alternative for Creating a New M-File**

Type `edit` in the Command Window to create a new file in the Editor/Debugger.

Type `edit filename.ext` to create the file `filename.ext`. If `filename.ext` already 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 prompt appears asking if you want to create a new file titled `filename.ext`:

- If you click **Yes**, the Editor/Debugger 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/Debugger 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.

If you want the dialog to appear, specify that using the preference Show dialog prompt.... For details, see the online documentation for preferences, “Opening Files in Editor”.

**Opening Existing Files in the Editor/Debugger**

To open an existing file in the Editor/Debugger, click the open button □ on the desktop or Editor/Debugger 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 filename, 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/Debugger, the files in the directory for the current file are shown.
The Editor/Debugger opens, if it is not already open, with the file displayed. The location of the file and the Editor/Debugger are determined by document positioning guidelines. You can rearrange the documents to suit your needs. For details, see “Opening and Arranging Documents” on page 2-7.

To make a document in the Editor/Debugger become the current document, click it, or select it from the Window menu or document bar.

Other Methods for Opening Files in the Editor/Debugger

These are other ways to open files in the Editor/Debugger:

- Drag a file from another MATLAB desktop tool or a Windows tool into the Editor/Debugger. For example, drag files from the Current Directory browser, or from Windows Explorer.
- Open files from the Current Directory browser—see “Opening Files” on page 5-37.
- Select a file to open from the most recently used files, which are listed at the bottom of the File menu in the Editor/Debugger 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/Debugger or another desktop tool such as the Command Window, select a filename, right-click, and select Open Selection from the context menu to open that file. See “Opening a Selection in an M-File” on page 6-28 for details.
- Set a preference that instructs MATLAB, upon startup, 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/Debugger. For example, type

```matlab
edit collatz.m
```

to open the file `collatz.m` in the Editor/Debugger, where `collatz.m` is on the search path or in the current directory. Use the relative or absolute pathname for the file you want to open if it is not on the search path or in the current directory.

Opening the Editor Without Starting MATLAB

On Windows platforms, you can use the MATLAB Editor without starting MATLAB. For example, double-click an M-file in Windows Explorer and the M-file opens in the MATLAB Editor without starting MATLAB. To open the Editor to a new file, run `$matlabroot/bin/win32/meditor.exe`. Regardless of the type of MATLAB license you have, you can open multiple instances of `meditor` because it is not considered an instance of MATLAB.

When you open the MATLAB Editor without starting MATLAB, the Editor is a stand-alone (or standalone) application. You cannot debug M-files from it, evaluate a selection, access source control features, dock the Editor in the MATLAB desktop, nor access help from it. It remains a stand-alone application, even if you subsequently open MATLAB. Other than these limitations, you can use the editing features as described in “Creating, Editing, and Running Files” on page 6-13.

For Windows platforms, when MATLAB is installed, the stand-alone Editor is automatically associated with files having a `.m` extension. If you double-click an M-file, the stand-alone Editor opens. You can change the association using Windows Explorer so that files with a `.m` extension open in the Editor/Debugger in MATLAB.

Arranging Editor/Debugger Documents

You can arrange the size and location of M-files and other text documents you open in the Editor/Debugger. Editor/Debugger documents follow the same arrangement practices as other desktop documents. For details, see “Opening and Arranging Documents” on page 2-7.
Preferences for the Editor/Debugger

Using preferences, you can specify the default behavior for various aspects of the Editor/Debugger.

To set preferences for the Editor/Debugger, select Preferences from the File menu in the 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 panel displays. Make changes and click Apply or OK.
For details about specific Editor/Debugger preferences, see “Preferences for the Editor/Debugger” in the online documentation, or click Help in the Preferences dialog box.

**Working with 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/Debugger.

When working with files created for C/C++, Java, and HTML, you can specify syntax highlighting and indenting preferences appropriate to those languages. Select File -> Preferences -> Editor/Debugger -> Language. See details in the online documentation for language preferences or click the Help button in the dialog box.

**Closing the Editor/Debugger**

To close the Editor/Debugger, click the close box in the title bar of the Editor/Debugger. This is different from the close box in the menu bar of the Editor/Debugger, which closes the current file when multiple files are open in a single window.

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 arrays in the Array Editor, and it will close the tools as well, that is, the Editor/Debugger and Array Editor, for example.

When you close the Editor/Debugger and any of the open files have unsaved changes, you are prompted to save the files.
Creating, Editing, and Running Files

In the Editor/Debugger, use these editing features to create, modify, and run your files:

- “Entering Statements” on page 6-13, including adding comments
- “Appearance of an M-File” on page 6-19, including syntax highlighting
- “Navigating in an M-File” on page 6-21, including go to, and find and replace
- “Opening a Selection in an M-File” on page 6-28
- “Saving M-Files” on page 6-29
- “Running M-Files from the Editor/Debugger” on page 6-31
- “Printing M-Files” on page 6-32
- “Closing M-Files” on page 6-32

Entering Statements

After opening an existing file or creating a new file, enter statements in the Editor/Debugger. Follow the same rules you would use for entering statements in the Command Window as described in these sections:

- “Case and Space Sensitivity” on page 3-8
- “Entering Multiple Functions in a Line” on page 3-11
- “Entering Long Statements” on page 3-11
- “Suppressing Output” on page 3-18
- “Formatting and Spacing Numeric Output” on page 3-19

In addition, utilize these Editor/Debugger features.

- “Adding Comments” on page 6-14
- “Changing the Case of Selected Text” on page 6-18
- “Undo and Redo” on page 6-19
- “Finding and Replacing Text in the Current File” on page 6-24
Adding Comments

Comments are strings or statements in an M-file 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 filename. 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.

- “Commenting Using the MATLAB Editor” on page 6-14
- “Commenting Using Any Text Editor” on page 6-15
- “Commenting Out Part of a Statement” on page 6-17
- “Formatting Comments” on page 6-18

Commenting Using the MATLAB Editor. You can comment the current line or a selection of lines:

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 6-19.

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.
Commenting Using Any Text Editor. You can make any line a comment by typing % at the beginning of the line. To put a comment within a line, type % followed by the comment text; MATLAB treats all the information after the % on that line as a comment.

```
% This is a comment. ——MATLAB ignores this comment line when you run the M-file.
This is not a comment. ——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 %} cannot 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, %}. At that point, only the lines between the block comment symbols...
have the syntax highlighting color for comments. 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.

```matlab
a = magic(3)

%{
    sum(a)
diag(A)
    sum(diag(A))
%}
sum(diag(flipr(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(flipr(a)))
```

```matlab
```
Commenting Out Part of a Statement. To comment out the end of a statement, put the comment character, %, before the comment. When you run the file, MATLAB ignores any text on the line after the %.

Any text following a % within a line is considered to be a comment.

```matlab
a = acos(10) % Initialize matrix
```

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 ....

```matlab
header = [{'Last Name', ' '}, ...
'First Name', ' ' ...
... 'Middle Initial,' ...
'Title']
```

MATLAB ignores the text following the ... on the line

```matlab
... '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

```matlab
'Title'
```

MATLAB effectively runs

```matlab
headers = [{'Last Name', ' '}, ...
'First Name', ' ' ...
'Title']
```
**Formatting Comments.** 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/Debugger. 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/Debugger. When typing a comment line, as you reach the 75th column, the comment automatically continues on the next line.

**Changing the Case of Selected Text**

To change the case of text in the Editor, select the text and then use one of these key sequences:

- Press **Alt+U, U** to change all text to upper case
- Press **Alt+U, L** to change all text to lower case
- Press **Alt+U, R** to change the case of each letter

This is useful, for example, when copying syntax from help in an M-file, where uppercase is used but will not run in MATLAB. In this example, the text was copied and pasted from `help get`.

\[ V = \text{GET}(H, \text{‘Default’}) \]
Select the line of text. Press \texttt{Alt+U} and then press \texttt{L}. The line becomes
\begin{verbatim}
v = get(h, 'default')
\end{verbatim}
Pressing \texttt{Alt+U} and then \texttt{R} changes the original help line to
\begin{verbatim}
v = get(h, 'DEFAULT')
\end{verbatim}

\textbf{Undo and Redo}
You can undo many of the Editor/Debugger actions listed in \textit{Edit} and \textit{Text} menus. Select \textit{Edit} \texttt{-> Undo}. You can undo multiple times in succession until there are no remaining actions to undo. Select \textit{Edit} \texttt{-> Redo} to reverse an undo.

\textbf{Appearance of an M-File}
The following features make M-files more readable:

- “Syntax Highlighting” on page 6-19
- “Indenting” on page 6-19
- “Nested Functions Indenting” on page 6-20
- “Line and Column Numbers” on page 6-20
- “Right-Hand Text Limit” on page 6-20
- “View Function or Subfunction” on page 6-20

You can specify the default behaviors for some of these—see “Fonts, Colors, and Other Preferences” on page 2-44.

\textbf{Syntax Highlighting}
Some entries appear in different colors to help you better find matching elements, such as \textit{if/else} statements. This is called syntax highlighting and is used in the Command Window and History, as well as in the Editor/Debugger. For more information, see the Command Window documentation for “Syntax Highlighting” on page 3-9.

\textbf{Indenting}
Program control entries are automatically indented to aid in reading loops, such as \textit{while/end} statements.

To move the current or selected lines further to the left, select \texttt{Decrease Indent} from the \textit{Text} menu. To move the current or selected lines further to
the right, select Increase Indent from the Text menu. If after using these features you want to apply automatic indenting to selected lines, select Smart Indent from the Text menu, or right-click and select it from the context menu.

Select a line or group or lines and press the Tab key to indent the lines. Press Shift+Tab to decrease the indent for the selected lines. This works differently if you select the 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 smart indenting and indenting preferences, see the Editor Keyboard and Indenting Preferences for the Editor/Debugger and Language Preferences for the Editor/Debugger in the online documentation.

**Nested Functions Indenting**

If you select the language preference for smart indenting, you can select from three indenting options when you enter a nested function (a function within a function) in the Editor. For details, see “Function Indenting Format” in the online documentation for Editor preferences.

**Line and Column Numbers**

Line numbers are displayed along the left side of the Editor/Debugger window. You can elect not to show the line numbers using preferences—see Display Preferences for the Editor/Debugger.

The line and column numbers for the current cursor position are shown in the far right side of the status bar in the Editor/Debugger.

**Right-Hand Text Limit**

By default, a faint vertical line appears at column 75 in the Editor/Debugger, providing a cue as to when a line becomes wider than desired, which is useful to know for printing, for example. You can hide the line or change the column number at which it appears—see Display Preferences for the Editor/Debugger.

**View Function or Subfunction**

View the function or subfunction the cursor is currently at in the right side of the status bar in the Editor/Debugger. See also dbstack.
Navigating in an M-File

There are several options for navigating in M-files:

- “Arrow and Control Keys (Shortcuts) in the Editor” on page 6-21
- “Going to a Line Number” on page 6-22
- “Going to a Bookmark” on page 6-22
- “Going to a Function (Subfunctions and Nested Functions)” on page 6-23

See also “Finding Text in Files” on page 6-23.

Arrow and Control Keys (Shortcuts) in the Editor

Following is the list of arrow and control keys that serve as shortcuts for using the Editor. This list does not include shortcut keys for menu items—you can view those on the menus. If you select the Editor’s Emacs Key Bindings for the Editor/Debugger preference, you can also use the Ctrl+key combinations shown. See also general “Keyboard Shortcuts” on page 2-35.

<table>
<thead>
<tr>
<th>Key</th>
<th>Additional Control Key for Emacs Preference</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>Ctrl+P</td>
<td>Move to previous line.</td>
</tr>
<tr>
<td>↓</td>
<td>Ctrl+N</td>
<td>Move to next line.</td>
</tr>
<tr>
<td>Ctrl+↑</td>
<td>none</td>
<td>Scroll screen down one line</td>
</tr>
<tr>
<td>Ctrl+↓</td>
<td>none</td>
<td>Scroll screen up one line</td>
</tr>
<tr>
<td>Ctrl+Home</td>
<td>none</td>
<td>Move to top of file.</td>
</tr>
<tr>
<td>Ctrl+End</td>
<td>none</td>
<td>Move to end of file.</td>
</tr>
<tr>
<td>←</td>
<td>Ctrl+B</td>
<td>Move back one character.</td>
</tr>
<tr>
<td>→</td>
<td>Ctrl+F</td>
<td>Move forward one character.</td>
</tr>
<tr>
<td>Ctrl+→</td>
<td>none</td>
<td>Move right one word.</td>
</tr>
<tr>
<td>Ctrl+←</td>
<td>none</td>
<td>Move left one word.</td>
</tr>
<tr>
<td>Home</td>
<td>Ctrl+A</td>
<td>Move to beginning of line.</td>
</tr>
</tbody>
</table>
Going to a Line Number

Select Edit -> Go to Line. In the resulting dialog box, enter the Line number and click OK. The cursor moves to that line number in the current M-file.

Going to a Bookmark

You can set a bookmark at a line in a file in the Editor/Debugger 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 Edit -> Set Bookmark. A bookmark icon appears to the left of the line.

To go to a bookmark, select Next Bookmark or Previous Bookmark from the Edit menu.
To clear a bookmark, position the cursor anywhere in the line and select Edit -> Clear Bookmark.

Bookmarks are not maintained after you close a 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), click the function button on the toolbar. Select the subfunction or nested function you want to go to from the alphabetical listing of all subfunctions and nested functions in that M-file. The list does not include functions that are called from the M-file, but only lists lines in the current M-file that begin with a function statement.

The subfunction or nested function that the current line is part of is shown at the right side of the status bar.

**Finding Text in Files**

There are different ways to find text in files:

- “Finding Text in the Current File” on page 6-23
- “Finding and Replacing Text in the Current File” on page 6-24
- “Finding Files or Text in Multiple Files” on page 6-25
- “Incremental Search” on page 6-25

**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 the next occurrences of the text.

To find the previous occurrence of selected text (find backwards) in the current file, press Ctrl+Shift+F3, or 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 replace the text within a file.

Finding Text. To search for text in files:

1 Click the find button in the Editor/Debugger toolbar, or select Edit -> Find and Replace.

The Find & Replace dialog box appears.

2 Complete the Find & Replace dialog box to find all occurrences of the text you specify.
   a Type the text in the Find what field.
   b In the Look in list box, select Editor - Current File.
   c Limit the search using Match case, Whole word, or Wrap around.
      These settings are remembered for your next MATLAB session.

3 Click Find. The next occurrence of the text is selected in the file.

You can find the next or previous occurrence of the text in the same file, replace the text in the same file, or find and replace the text in another file.

Finding the Next or Previous Occurrence of the Text. To find the next occurrence of the text you entered in the Find & Replace dialog box, click the Find button (or press the Enter key) if the dialog box is open (and has focus). If the Find & Replace dialog box is closed or does not have focus, select Edit -> Find Next.

To find the previous occurrence of that text (find backwards), select Edit -> Find Previous.
If there are no more occurrences in the file, MATLAB beeps. With Wrap around selected, press **Find Next** or **Find Previous** to continue at the beginning or end of the file.

You can go to a different file and use the same **Find & Replace** dialog box to find the text in the other file. You can also replace the text.

**Function Alternative for Finding Text.** Use `lookfor` to search for the specified text in the help in all M-files on the search path.

**Replacing Text.** After finding text using the Find and 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 currently open 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.

**Finding Files or Text in Multiple Files**

To find directories and filenames that include specified text, or whose contents contain specified text, use **Edit -> Find Files**. For details, see “Finding Files and Content Within Files” on page 5-40.

**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 “Incremental Search” on page 3-23. To use the incremental search feature in the Editor/Debugger.

1. Position the cursor where you want the search to begin.
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 binding (for Macintosh key bindings, use the Command key instead of Ctrl).

- Press Ctrl+Shift+S to search forward or Ctrl+Shift+R to search backward for Windows key bindings (for Macintosh key bindings, use the Command key instead of Ctrl).

An incremental search field appears in the left side of the status bar of the current file window. **F Inc Search** means search Forward from the cursor.
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.

```
function collatzplot(m)
  % Plot length of sequence for Collatz problem
  % 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 `Back Space` 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.
- 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.
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 Back Space key deletes the entire previous search term, rather than just the last letter.

**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/Debugger. Select the name and then right-click and select Open Selection from the context menu. Based on what the selection is, the Editor/Debugger performs a different action.

<table>
<thead>
<tr>
<th>Selection</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subfunction</td>
<td>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/Debugger 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.</td>
</tr>
<tr>
<td>M-file or other text file</td>
<td>Opens in the Editor.</td>
</tr>
<tr>
<td>Figure file (.fig)</td>
<td>Opens in a figure window.</td>
</tr>
<tr>
<td>Variable</td>
<td>Opens in the Array Editor.</td>
</tr>
<tr>
<td>Model</td>
<td>Opens in Simulink.</td>
</tr>
<tr>
<td>Other</td>
<td>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.</td>
</tr>
</tbody>
</table>
After selecting a name, you can also choose **Help on Selection** from the context menu to see documentation for the item. For example, select a function, right-click and select **Help on Selection**. The reference page for that function opens in the Help browser, or if the reference page does not exist, the M-file help appears.

### Saving M-Files

After making changes to an M-file, you see an asterisk (*) next to the filename in the title bar of the Editor/Debugger. This 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 using the save button on the toolbar. If the file has not been changed, **Save** is grayed out, but you can instead use **Save As** from the **File** menu to save to a different filename.
- **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 automatically assigns the `.m` extension to the filename. If you do not want an extension, type a . (period) after the filename.
- **Save All**—Saves all named files to their existing filenames. For all newly created files, the **Save file as** dialog box opens, where you assign a name to each file and save it.

You cannot save a file while in debug mode. If you try to, MATLAB displays a dialog box asking if you want to exit debug mode and then save the file.
Note  Save any M-files you create and any M-files from 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 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 MATLAB session to improve performance. 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 the files in the current session. For more information, see `rehash` or “Toolbox Path Caching in MATLAB” on page 1-10.

Autosave
As you make changes to a file in the Editor/Debugger, every five minutes the Editor/Debugger automatically saves a copy of the file to a file of the same name but with an `.asv` extension. The autosaved copy is useful if you have system problems and lose changes made to your file. In that event, you can open the autosaved 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 autosaved files. For details, see “Autosave Preferences for the Editor/Debugger” in the online documentation.”

By default, autosaved files are not automatically deleted when you delete the source file. So if, for example, you rename or remove a file, delete the corresponding `.asv` file. You can select the Close options Editor/Debugger preference to **Automatically delete autosave files**; then when you close the source file in the Editor, it deletes the corresponding autosave file.

If the M-file you are editing is in a read-only directory, an autosave copy of the file is *not* made.
Accessing Your Source Control System

If you use a source control system for M-files, you can access it from within the Editor/Debugger using File -> Source Control. For more information, see Chapter 9, “Source Control”.

Running M-Files from the Editor/Debugger

You can run a script or a function that does not require an input argument directly from the Editor/Debugger. Click the run button on the toolbar, or select Run from the Debug menu.

If the file 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 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.

Note that if the file has unsaved changes, running it from the Editor/Debugger automatically saves the changes before running. In that event, the menu item is Save and Run.

See “Running an M-File with Breakpoints” on page 6-43 for additional information about running M-files while debugging.

Viewing Datatips

For a script M-file, position the cursor to the left of a variable on that line. Its current value appears—this is called a datatip, which is like a tooltip for data. The datatip stays in view until you move the cursor. If you have trouble getting the datatip to appear, click in the line and then move the cursor next to the variable.

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 a function M-file, if you hover over a variable that variable that 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. To avoid confusion, you might want to turn datatips off while editing. Select File -> Preferences -> Display, and for General Display Options, clear the check mark for Enable datatips in edit mode.

While you are debugging, datatips always display and show the value of the variable in the function or base workspace.
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/Debugger by selecting File -> Page Setup. For example, you can specify printing with a header. For more information, see “Page Setup Options for Printing” on page 2-40.

Closing M-Files
To close the current M-file, select Close filename from the File menu, or click the close box in the Editor’s menu bar. This is different from the close box in the titlebar of the Editor/Debugger, which closes all open files in that Editor/Debugger window.

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 Array 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.
Debugging M-Files

This section introduces general techniques for finding errors in M-files. It then illustrates MATLAB debugger features found in the Editor/Debugger as well equivalent Command Window debugging functions, using a simple example. It includes these topics:

- “Finding Errors in M-Files” on page 6-33
- “Debugging Example—The Collatz Problem” on page 6-36
- “Debugging Process and Features” on page 6-39

Finding Errors in M-Files

Debugging is the process by which you isolate and fix problems with your code. Debugging helps to correct 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.

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:

<table>
<thead>
<tr>
<th>Technique or Tool</th>
<th>Description</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax highlighting</td>
<td>Syntax highlighting helps you identify syntax errors in an M-file before you run the file.</td>
<td>“Syntax Highlighting” on page 6-19</td>
</tr>
<tr>
<td>Error messages</td>
<td>When you run an M-file with a syntax error, MATLAB 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/Debugger, 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.</td>
<td>none</td>
</tr>
<tr>
<td>Editor/Debugger and Debugging Functions</td>
<td>The MATLAB Editor/Debugger and debugging functions are useful for correcting run-time errors 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 halt 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.</td>
<td>“Debugging Example—The Collatz Problem” on page 6-36 and “Debugging Process and Features” on page 6-39</td>
</tr>
<tr>
<td>Cells</td>
<td>In the Editor/Debugger, isolate sections of an M-file, called cells, so you can easily make changes to and run a single section.</td>
<td>“Rapid Code Iteration Using Cells” on page 6-63</td>
</tr>
<tr>
<td>M-Lint</td>
<td>Use M-Lint to help you verify the integrity of your code and learn about potential improvements. Access M-Lint from the Editor/Debugger by selecting <strong>Tools -&gt; Check Code with M-Lint</strong>.</td>
<td>“M-Lint Code Check Report” on page 7-25</td>
</tr>
</tbody>
</table>
Other Useful Techniques for Finding and Correcting Errors

- 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 prompt:

  \[K>>\]

  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 suppress the display of intermediate calculations 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. Similarly, use the dependency report in the Visual Directory tools.
Debugging Example—The Collatz Problem

The example debugging session requires you to 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 \( n \), and plots the results. The plot shows patterns that can be further studied.

Following are the results when \( n \) is 1, 2, or 3.

<table>
<thead>
<tr>
<th>( n )</th>
<th>Sequence</th>
<th>Number of Integers in the Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2 1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3 10 5 16 8 4 2 1</td>
<td>8</td>
</tr>
</tbody>
</table>

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.
Function for collatz.m:

```matlab
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
```

Function for collatzplot.m:

```matlab
function collatzplot(m)
% Plot length of sequence for Collatz problem
% Prepare figure
clf
set(gcf,'DoubleBuffer','on')
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
```
**Trial Run for Example**

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.

![Figure 1](image)

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.

**Debugging Process and Features**

You can debug the M-files using the Editor/Debugger, which is a graphical user interface, as well as using debugging functions from the Command Window. You can use both methods interchangeably. The example describes both methods. Debugging process consists of

- “Preparing for Debugging” on page 6-39
- “Setting Breakpoints” on page 6-40
- “Running an M-File with Breakpoints” on page 6-43
- “Stepping Through an M-File” on page 6-45
- “Examining Values” on page 6-46
- “Correcting Problems and Ending Debugging” on page 6-51

Some additional debugging options include

- “Conditional Breakpoints” on page 6-58
- “Error Breakpoints” on page 6-60

**Preparing for Debugging**

Do the following to prepare for debugging:

- Open the file—To use the Editor/Debugger for debugging, open it with the file you will 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 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.
Example—Preparing for Debugging

Open the file `collatzplot.m`. Make sure the current directory is the directory in which you saved `collatzplot`.

Setting Breakpoints

Set breakpoints to pause execution of the function so you can examine values where you think the problem might be. You can set breakpoints in the Editor/Debugger, 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 6-41.
- A conditional breakpoint, which stops at a specified line in an M-file only under specified conditions. For details, see “Conditional Breakpoints” on page 6-58.
- 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 6-60.

You can disable standard and conditional breakpoints so that MATLAB temporarily ignores them, or you can remove them. For details, see “Disabling and Enabling Breakpoints” on page 6-52. 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.

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/Debugger, click in the breakpoint alley at the line where you want to set the breakpoint. The breakpoint alley is the column on the left side of the Editor/Debugger, 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 Breakpoints 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 10, 11, and 12. The breakpoint at line 10 allows you to step into collatz to see if the problem might be there. The breakpoints at lines 11 and 12 stop the program where you can examine the interim results.
Valid (Red) and Invalid (Gray) Breakpoints. Red breakpoints are valid standard breakpoints. If breakpoints are instead 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 10
```
```
dbstop in collatzplot at 11
```
```
dbstop in collatzplot at 12
```

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/Debugger.

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 and not from the Editor/Debugger.
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/Debugger
  by the green arrow just to the right of the breakpoint, which in the example,
  is line 10 of collatzplot as shown here.

  ![Image of line 10 with breakpoint]

  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

  10  plot_seq = collatz(N);

- The function displayed in the Stack field on the toolbar changes to reflect the
  current function. 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.
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/Debugger or desktop, or use the equivalent functions.

<table>
<thead>
<tr>
<th>Toolbar Button</th>
<th>Debug Menu Item</th>
<th>Description</th>
<th>Function Alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continue or Run or Save and Run</td>
<td>Continue execution of M-file until completion or until another breakpoint is encountered. The menu item says Run or Save and Run if a file is not already running.</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Go Until Cursor</td>
<td>Continue execution of M-file until the line where the cursor is positioned. Also available on the context menu.</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Step</td>
<td>Execute the current line of the M-file.</td>
<td>dbstep</td>
<td></td>
</tr>
<tr>
<td>Step In</td>
<td>Execute the current line of the M-file and, if the line is a call to another function, step into that function.</td>
<td>dbstep in</td>
<td></td>
</tr>
<tr>
<td>Step Out</td>
<td>After stepping in, run the rest of the called function or subfunction, leave the called function, and pause.</td>
<td>dbstep out</td>
<td></td>
</tr>
</tbody>
</table>

Continue Running in the Example

In the example, collatzplot is paused at line 10. Because the problem results are correct for \( N/n = 1 \), we want to continue running until \( N/n = 2 \). Press the continue button three times to move through the breakpoints at lines 10, 11, and 12. Now the program is again paused at the breakpoint at line 10.
Stepping In in the Example

Now that `collatzplot` is paused at line 10 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 10 of `collatzplot` goes to line 9 of `collatz`. If `collatz` is not open in the Editor/Debugger, it automatically opens if you have selected **Debug -> Open M-Files When Debugging**.

The pause indicator at line 10 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 6-47
- “Viewing Values as Datatips in the Editor/Debugger” on page 6-47
- “Viewing Values in the Command Window” on page 6-48
- “Viewing Values in the Workspace Browser and Array Editor” on page 6-49
- “Evaluating a Selection” on page 6-50

Many of these methods are used in “Examining Values in the Example” on page 6-50.
Selecting the Workspace

Variables assigned through the Command Window and created using scripts are considered to be in the base workspace. Variables created in each function have their own 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 function workspace currently running or the base workspace, first select that workspace from the list in the Stack field.

If you use debugging functions from the Command Window, use dbstack to display the call stack. Use dbup and dbdown to change to a different workspace. Use who or whos to list the variables in the current workspace.

Workspace in the Example. At line 10 of collatzplot, we stepped in, putting us at line 9 of collatz. The Stack shows that collatz is the current workspace.

Viewing Values as Datatips in the Editor/Debugger

In the Editor/Debugger, position the cursor to the left of a variable on that line. Its current value appears—this is called a datatip, which is like a tooltip for data. The datatip stays in view until you move the cursor. If you have trouble getting the datatip to appear, click in the line and then move the cursor next to the variable.

Datatips in the Example. Position the cursor over \( n \) in line 9 of collatz. The datatip shows that \( n = 2 \), as expected.
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 MATLAB displays its current value. For the example, to see the value of n, type

\[ n \]

MATLAB returns the expected result

\[ n = 2 \]

and displays the debug prompt, K>>.
Viewing Values in the Workspace Browser and Array 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 Array Editor opens, displaying the content for that variable. You can open the Array Editor directly for a variable using `openvar`.

To see the value of n in the Array Editor for the example, type

```
openvar n
```

and the Array Editor opens, showing that n = 2 as expected.
Evaluating a Selection
Select a variable or equation in an M-file in the Editor/Debugger. Right-click and select Evaluate Selection from the context menu. MATLAB displays the value of the variable or equation in the Command Window. 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

\[
\begin{bmatrix}
2 \\
1
\end{bmatrix}
\]

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 10 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 \texttt{dbstep out} in the Command Window.

In collatzplot, step again to advance to line 11, then line 12. The variable seq_length in line 11 is a vector with the elements

\[
\begin{bmatrix}
1 \\
2
\end{bmatrix}
\]

which is correct.

Finally, step again to advance to line 13. Examining the values in line 12, \( N = 2 \) 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

\[
\begin{bmatrix}
2 \\
1
\end{bmatrix}
\]

it is the incorrect variable for plotting. Instead, seq_length(N) should be plotted.
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 6-51
- “Ending Debugging” on page 6-51
- “Disabling and Clearing Breakpoints” on page 6-52
- “Correcting an M-File” on page 6-53
- “Completing the Example” on page 6-54

Many of these features are used in “Completing the Example” on page 6-54.

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 Array Editor. Then continue running or stepping through the program. If the new value does not produce the expected results, the program has a different or another 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 best to 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 6-42 for details.

To end debugging, click the exit debug mode icon, or select Exit Debug Mode from the Debug menu.

You can instead use the function dbquit to end debugging.
After quitting debugging, the pause indicators in the Editor/Debugger 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 temporarily disable selected breakpoints so the program 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 6-58.

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 Breakpoints or context menu. You can also disable a conditional breakpoint by clicking the breakpoint icon. The breakpoint icon has an X through it as shown here.

```
 Disabled breakpoint  plot_seq = colletz(N):
```

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.

When you run dbstatus, the resulting message for a disabled breakpoint is

```
 Breakpoint on line 10 has conditional expression 'false'.
```

**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/Debugger, click anywhere in a line and select **Set/Clear Breakpoint** from the Breakpoints or context menu. The breakpoint for that line is cleared. Another way to clear a breakpoint is to click a standard breakpoint icon, or a disabled conditional breakpoint icon.
To clear all breakpoints in all files, select Breakpoints -> Clear All Breakpoints, or click the equivalent button on the toolbar.

The function that clears breakpoints is `dbclear`. To clear all breakpoints, use `dbclear all`. 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`

---

**Note**  When `clear name` or `clear all` is in a statement in an M-file that you are debugging, it clears the breakpoints.

---

**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 6-42 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 12, 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 \texttt{collatzplot} for \( m = 6 \) as well as the results for \texttt{collatz}, add this line at the end of \texttt{collatz.m}:

\begin{verbatim}
    sequence
\end{verbatim}

which displays the series in the Command Window. The results for when \( n = 6 \) are:

\begin{verbatim}
    sequence =
    6   3   10   5   16   8   4   2   1
\end{verbatim}

Then run \texttt{collatzplot} for \( m = 6 \) by typing:

\begin{verbatim}
    collatzplot(6)
\end{verbatim}

7 To make debugging easier, you ran \texttt{collatzplot} for a small value of \( m \). Now that you know it works correctly, run \texttt{collatzplot} for a larger value to produce more interesting results. Before doing so, you might want to suppress output for the line you just added in step 6, line 19 of \texttt{collatz.m}, by adding a semicolon to the end of the line so it appears as

\begin{verbatim}
    sequence;
\end{verbatim}

Then run:

\begin{verbatim}
    collatzplot(500)
\end{verbatim}
The following figure shows the lengths of the Collatz series for \( n = 1 \) through \( n = 500 \).
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 6-58
- “Copying, Modifying, Disabling, and Clearing Conditional Breakpoints” on page 6-59
- “Function Alternative for Conditional Breakpoints” on page 6-59

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 Breakpoints or context menu. If an standard breakpoint already exists at that line, use this same method to make it conditional.

The MATLAB Editor conditional breakpoint dialog box opens.
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 example, at line 10 in collatzplot, enter

```matlab
N >= 2
```

A yellow breakpoint icon appears in the breakpoint alley at that line.

When you run the file, MATLAB 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 10 when \( N \) is 2. If you continue executing, MATLAB pauses at line 10 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 **Breakpoints** 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).
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 via 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 When Debugging**. 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 6-60
- “Error Breakpoint Types and Options” on page 6-61 (Errors, Try/Catch Errors, Warnings, NaN or Inf, Use Message Identifiers)
- “Function Alternative for Error Breakpoints” on page 6-62

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/Debugger 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 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 MATLAB encounters a `NaN` (not-a-number) or `Inf` (infinite) value, it pauses, enters debug mode, and opens the M-file, paused at 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, select the Errors, Try/Catch Errors, or Warnings tab, select the Use Message Identifiers option, and click the Add button.

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. 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.
Rapid Code Iteration Using Cells

When working with MATLAB, you often experiment with your code—modifying it, testing it, and updating it—until you have an M-file that does what you want. Use the cell features in the MATLAB Editor to make the experimental phase of your work with M-file scripts easier. You can also use cell features with function M-files, but there are some restrictions—see “Using Cells in Function M-Files” on page 6-73.

If you are using the Help browser, watch the Rapid Code Iteration Using Cells video demo for an overview of the major functionality.

The overall structure of many M-file scripts seems to naturally consist of multiple sections. Especially for larger files, you often focus efforts on a single section at a time, refining the code in just that section. To facilitate this process, use M-file cells, where a cell is a defined section of code.

This is the overall process of using cells for rapid code iteration:

1. In the MATLAB Editor, enable cell mode. Select Cell -> Enable Cell Mode. Items in the Cell menu become selectable and the cell toolbar appears.

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 6-64.

3. Once 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 “Navigating and Evaluating with Cells” on page 6-69.

4. Cells are also useful if you want to share your results by publishing your work in a presentation format, such as an HTML document. See Chapter 8, “Publishing to HTML, XML, LaTeX, Word, and PowerPoint Using Cells”, for details.
Defining Cells

Cell features operate on contiguous lines of code you want to evaluate as a whole in an M-file script, called cells. To define a cell, first be sure that cell mode is enabled (see step 1 on page 6-63). Position the cursor just before the line you want to start the cell and then select Cell -> Insert Cell Divider or click the cell divider button . MATLAB inserts a line after the cursor containing two percent signs (%), which is the “start new cell” indicator to MATLAB. A cell consists of the line starting with %% and the lines that follow, up to the start of the next cell, which is identified by % at the start of a line.

You can also define a cell by entering two percent signs (%) at the start of the line where you want to begin the new cell. Alternatively, select the lines of code to be in the cell and then select Cell -> Insert Cell Dividers Around Selection.
You can, of course, define a cell at the start of a new 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 cells, and moving lines of code.

MATLAB will not execute the code in lines beginning with %, so be sure to put any executable code for the cell on the following line. 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.

Note that the first cell in a file does not have to begin with %% MATLAB automatically understands any lines above the first % line to be a cell. If there are no cell dividers in an M-file, MATLAB understands the entire file to be a single cell.

**Cell Titles and Highlighting**

After the %, type a space followed by a description of the cell. The Editor emphasizes the special meaning of the start of a cell by making any text following the percent signs and space bold. The text on the % line is called the cell title (like a section title). Including cell titles is optional, however, they improve readability of the file and are used for cell publishing features.

When the cursor is positioned in any line within a cell, the Editor highlights the entire cell with a yellow background. This identifies it as the current cell. For example, the highlighted cell is the cell evaluated by the Evaluate Current Cell option on the Cell menu.

If you want cell titles to appear in plain rather than bold text, or if you do not want yellow highlighting for the current cell, change these preferences. Select File -> Preferences -> Editor/Debugger -> Display and change the appropriate Cell display options.
Example—Define Cells

This examples defines two cells for a simple M-file called *sine_wave*, shown in the following code and figure. The first cell creates the basic results, while the second decorates the plot. The two cells in this example 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.

```matlab
% 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')
```
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 Divider**.

   The Editor inserts %% as the first line and moves the rest of the file down one line. All lines are highlighted in yellow, indicating that the entire file is a single cell.

3 Enter a cell title following the %%.

   Type a space first, followed by the description.

   **Calculate and Plot Sine Wave**

4 Position the cursor at the start of line 7, title... Select **Cell -> Insert Cell Divider**.

   The Editor inserts a line containing only %% at line 7 and moves the remaining lines down by one line. Lines 7 through 12 are highlighted in yellow, indicating they comprise the current cell.

5 Enter a cell title for the new cell. On line 7, type a space after the %%.

   Followed by the description

   **Modify Plot Properties**

Save the file. The file appears as shown in this figure.
Removing Cells

To remove a cell, delete one of the percent signs (%) from the line that starts the cell. This changes the line from a cell to a standard comment and merges the cell with the preceding cell. You can also just delete the entire line that contains the %.

```matlab
1  % Calculate and Plot Sine Wave
2  % Define the range for x.
3  % Calculate and plot y = sin(x).
4  x = 0:1:6*pi;
5  y = sin(x);
6  plot(x,y)
7  % Modify Plot Properties
8  title('Sine Wave', 'FontWeight', 'bold')
9  xlabel('x')
10  ylabel('sin(x)')
11  set(gca, 'Color', 'w')
12  set(gca, 'NameBar', 'none')
```
Navigating and Evaluating with Cells

While you develop an M-file, you can use these Editor cell features:

- “Navigating Among Cells in an M-file” on page 6-69
- “Evaluating Cells in an M-file” on page 6-69
- “Modifying Values in a Cell” on page 6-70
- “Example—Evaluate Cells” on page 6-71

Navigating Among Cells in an M-file

To move to the next cell, select Cell -> Next Cell. To move to the previous cell, select Cell -> Previous Cell. To move to a specific cell, click the cell titles button and from it, select the cell title you want to move to. Cells without titles are not listed.

Evaluating Cells in an M-file

To evaluate the code in a cell, use the Cell menu evaluation items or buttons in the cell toolbar. When you evaluate a cell, the results display in the Command Window, Figure window, or otherwise, depending on the code evaluated.

The cell evaluation features run the 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 already exist in the MATLAB workspace.

**Note** While you can set breakpoints and debug a file containing cells, when you evaluate a file from the Cell menu, breakpoints are ignored. To run the file and stop at breakpoints, use Run/Continue in the Debug menu.

Evaluate Current Cell. Select Cell -> Evaluate Current Cell or click the evaluate cell button to run the code in the current cell.

Evaluate and Advance. Select Cell -> Evaluate Current Cell and Advance or click the evaluate and advance button to run the code in the current cell and move to the next cell.
Evaluate File. Select Cell -> Evaluate Entire File or click the evaluate file button to run all of the code in the file.

Note A beep means there is an error. See the Command Window for the error message.

Modifying Values in a Cell
You can use cell features to modify numbers in a cell, which also automatically reevaluates the cell.

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.

You can use the numeric keypad operators instead of the operator buttons on the toolbar.

Note MATLAB does not automatically save changes you make to values using the cell toolbar. To save changes, select File -> Save.
Example—Evaluate Cells
In this example, modify the values for x in sine_wave.m.

1. Run the first cell in sine_wave.m. Click somewhere in the first cell, that is, between lines 1 and 6. Select Cell -> Evaluate Current Cell. The following figure appears.
2 Assume you want more values for \( x \), to produce a smoother curve. Position the cursor in line 4, next to the 1. In the cell toolbar, change the 1.1 default multiplier/divisor value to 2. Click the divisor button •.

Line 4 becomes

\[ x = 0:0.0625:6\pi; \]

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 divisor button three more times. Line 4 becomes

\[ x = 0:0.0625:6\pi; \]

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 multiplier 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 You do not need to evaluate the entire file to modify the plot properties. Instead, evaluate the second cell, that is, lines 7 through 12. You can 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.
Using Cells in Function M-Files

You can define and evaluate cells in function M-files as long as the variables referred to in the cell are in your workspace. For example, 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. Cell publishing is not supported for function M-files.
Tuning and Managing M-Files

This set of tools 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. If you are using the Help browser, watch the Directory Reports video demo for an overview of the major functionality.

Visual Directory in Current Directory Browser (p. 7-2)  
An alternate view of the current directory, it is useful for managing groups of M-files.

Directory Reports in Current Directory Browser (p. 7-11)  
HTML reports about files in the current directory: TODO/FIXME, Help, Contents, Dependency, File Comparison, Coverage (for Profiling), and M-Lint Code Check.

M-Lint Code Check Report (p. 7-25)  
Report that identifies potential errors, problems, and opportunities for improvement in your code.

Profiling for Improving Performance (p. 7-35)  
Report that identifies where an M-file spends the most time, indicating where to focus when looking for performance improvements.
Visual Directory in Current Directory Browser

The MATLAB Current Directory browser displays directories and their files, and allows you to make changes to them. To display the Current Directory browser, select Desktop -> Current Directory. For more information, see “Current Directory Browser” on page 5-30.

The Visual Directory is an alternate view in the Current Directory browser. To access the Visual Directory, click the show visual directory button on the Current Directory browser toolbar. Click the button again to return to the classic view of the Current Directory browser.

![Visual Directory in Current Directory Browser](image)
These are the actions you can take using the Visual Directory tool:

- “Navigate Directory Hierarchy” on page 7-4
- “View and Edit Files” on page 7-5
- “Sort by Contents.m” on page 7-6
- “Run, Make Thumbnail, Delete File (Show Actions)” on page 7-7
- “Show File Sizes” on page 7-9
- “Show Function/Script” on page 7-9
Navigate Directory Hierarchy

The subdirectories of the current directory are listed under **Subfolders**, with the number of M-files they contain listed in parentheses. Click a subdirectory to make it the current directory and display its contents.

The `mydemos` subdirectory contains 11 M-files. Click `mydemos` to make it the current directory and display its contents.

Click the `<UP>` link or the up icon in the toolbar to make the parent directory become the current directory and to display its contents.

When you navigate up and down through directories, the Visual Directory display automatically updates to reflect the currently selected check boxes.
View and Edit Files

Click a filename to open that file in the Editor where you can view and edit it.

M-files you published to HTML using cells also show an html link. Click html to view the HTML output. For more information about publishing M-files to HTML, see “Publishing to HTML, XML, LaTeX, Word, and PowerPoint Using Cells” on page 8-2.
Sort by Contents.m

By default, M-files in the current directory are listed in alphabetical order. To instead list them as they are ordered in the Contents.m file for the directory, select the Sort by Contents.m check box and click Refresh. A Contents.m file is a file you create that contains the filename and a brief description for M-files in the directory. When a user types help followed by the directory name, such as help mydemos, MATLAB displays the Contents.m file, effectively providing an overview of the files in the directory. The Contents report is a tool to help you create and maintain Contents.m files. You can create and edit the via the links in the Visual Directory, edit Contents.m, and run contentsrpt, or via the Directory Reports. For more information about the Contents report and Contents.m file, see the Directory Reports documentation for “Contents Report” on page 7-17.
Run, Make Thumbnail, Delete File (Show Actions)
Select the **Show actions** check box and click **Refresh**. Below the description for each M-file, the Visual Directory lists the actions you can perform on that file.

Click **Show actions**, followed by **Refresh** to view actions you can perform on each M-file. The **make thumbnail** and **delete thumbnail** entries appear only when the **Show thumbnails** check box is also selected.

**Run M-File**
Click **run** to execute an M-file. Script M-files run, but M-file functions that require an input argument do not run.

**Delete M-File**
Click **delete** to remove the file. The file is permanently removed from the directory.
Make and Delete Thumbnail

When Show thumbnails is also selected, the make thumbnail and delete thumbnail actions appear. A thumbnail is a snapshot of the graphical output the M-file produces. Specifically, the snapshot is the figure returned by gcf after the M-file runs. In the example, there is a snapshot shown for logo5.m. Thumbnails help you to

• Distinguish between similar M-files, such as different versions of a file you experimented with.
• Find a file when you do not remember the exact name.
• Look at files created by others and quickly identify the ones in which you might be interested, based on the snapshots.

To create a thumbnail for a file, click make thumbnail. You cannot make a thumbnail for M-file functions that require an input argument. To delete a thumbnail, click delete thumbnail. The thumbnail does not automatically update when you make changes to the file. To update a thumbnail to reflect the latest version of the M-file, click make thumbnail again.
Show File Sizes
Select the **Show file sizes** check box and click **Refresh**. The actual file size appears, as well as a blue horizontal bar representing the file size that allows you to quickly identify the relative file sizes.

Show Function/Script
Select the **Show fcn/script** check box and click **Refresh**. Each M-file is labeled and color coded as a script or function, with scripts highlighted in yellow and functions highlighted in blue. For descriptions of these types of M-files, see the reference pages for script and function.
Color coded labeling lets you quickly identify M-files as scripts or functions.
Directory Reports in Current Directory Browser

- “Accessing and Using Directory Reports” on page 7-11
- “M-Lint Code Check Report” on page 7-25
- “TODO/FIXME Report” on page 7-12
- “Help Report” on page 7-13
- “Contents Report” on page 7-17
- “Dependency Report” on page 7-20
- “File Comparison Report” on page 7-22
- “Coverage Report” on page 7-24

Accessing and Using Directory Reports

Directory reports help you refine the M-files in a directory and improve their performance. They are also useful for when you prepare files for use by others, such as for a finished project, to share on MATLAB Central, or for a toolbox to be distributed.

Access directory reports from the MATLAB Current Directory browser. To display the Current Directory browser, select Desktop -> Current Directory. For more information, see “Current Directory Browser” on page 5-30.

Navigate to the directory whose M-files you want to produce reports about. Then, in the Current Directory browser toolbar, click the down arrow button and select the type of report you want to run for all the M-files in the current directory.
The report you selected appears as an HTML document in the MATLAB Web browser.

- In a report, click a filename to open that file in the Editor, where you can view it or make changes to it. Click a line number to open the file at that line.
- To update a report after making changes to the report options or files, or after changing the current directory, click Refresh at the top of the report.
- When you run a report, it replaces the report currently displayed. Use the back and forward buttons in the toolbar to see a previously run report and then return to the most recent.

You cannot run directory reports when the path is a UNC (Universal Naming Convention) pathname, that is, starts with \\.

You cannot run directory reports when the path is a UNC (Universal Naming Convention) pathname, that is, starts with \\.

**TODO/FIXME Report**

The TODO/FIXME report shows M-files that contain text strings you included as notes to yourself, such as TODO. Use this report to easily identify M-files that still require work or some other actions.

In the report, select one or more check boxes to display lines containing the specified strings (TODO and FIXME), and click Refresh. You can also select the check box for the text field and enter any text string in the field, such as NOTE or TBD to identify lines containing that string.
The TODO/FIXME report identifies lines containing specified text strings, such as TODO, and NOTE shown here, so you can easily see reminders you included.

Enter any text string in this field.

Click the line number to open the file at that line in the Editor, where you can make changes.

**Help Report**

The Help report presents a summary view of the help component of your M-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.

Select one or more check boxes to display the specified help information and click Refresh.
Use this information to help you 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.
Show Subfunctions
With Show subfunctions selected, the Help report displays help information for all subfunctions called by each function. Help information for subfunctions is highlighted in gray.

Description
With Description selected, the Help report displays 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, No description line, highlighted in pink, appears instead.

Examples
With Examples selected, the Help report displays 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 non-blank comment lines. Check 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
With Show all help selected, the Help report displays all the 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
With See Also selected, the Help report displays 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 MATLAB 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 a See-also M-file 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
With Copyright selected, the Help report displays 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

```matlab
% Copyright 2001-2004
```

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.
Contents Report

The Contents report displays information about the integrity of the Contents.m file for the directory. A Contents.m file includes the filename and a brief description of each M-file in the directory. When you type help followed by the directory name, such as help mydemos, MATLAB displays the information in the mydemos/Contents.m file. For more information, see “Providing Help for Your Program” in MATLAB Programming documentation.

If there is no Contents.m file for the directory and you run the Contents report, 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 “internal” or “helper” files that you do not want to expose when displaying help for the directory.

You need to 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 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.
Use the links displayed for each line, or edit the `Contents.m` file directly, or edit the M-files to make the changes. To make all of the suggested changes at once, click **fix all**. To automatically align the filenames and descriptions in the `Contents.m` file, click **fix spacing**.
If you always want the Contents.m file to reflect all files in the directory, it you can automatically generate a new Contents.m file rather than changing the file based on the Contents report. To do this, first delete the existing Contents.m file, run the Contents report, and click yes when prompted for MATLAB to automatically create one.

**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 filenames 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 (default) 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 V5

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 filename 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 ]
```

Dependency Report

The Dependency report shows dependencies among M-files in a directory. This helps you determine all the M-files you need to provide when you tell someone to run a particular M-file. If you do not provide all the dependent M-files along with the M-file you want them to run, they will not be able run the file. The report does not list as dependencies the M-files in the toolbox/matlab directory because every MATLAB user already has those files. Select Show child functions to see a list of all M-files (children) called by each M-file in the directory (parent). The report also indicates where each child function resides, for example, in a specified toolbox. If a child function’s location is listed as unknown, it could be because the child function is not on the search path or in the current directory.
The Dependency report is similar to running the `depfun` function, although the two do not provide the exact same results. For performance purposes, the Dependency report limits the functions considered.

Select **Show parent functions** to list the M-files that call each M-file. The report limits the parent (calling) functions to those in the current directory. Select **Show subfunctions** to include subfunctions in the report. Subfunctions are listed directly after the main function and are highlighted in gray.
**File Comparison Report**

The File Comparison report identifies the differences between two files in the current directory. Some other tools refer to this as a diff report. As an example, you can use this to easily compare an autosaved version of a file to the latest version.

In the File Comparison report listing for a directory, click *file 1* for the first file. Then click *file 2* for the file you want to compare the first file to.

The File Comparison report then displays the files next to each other and highlights lines that do not match. Pink highlighting and an *x* at the start of the line indicates that the content of the lines differs. Green highlighting and a > at the start of the line indicates a line that exists in one file but not the other.
Excerpt of File Comparison report.

Pink highlighting or an x at the start of a line denotes the lines differ.

Green highlighting or a > at the start of a line denotes the line exists in one file but not the other.
Coverage Report

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 7-35.


3. Click the Coverage link to see the profile detail report for the file.
**M-Lint Code Check 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. In MATLAB, the M-Lint tool displays a message for each line of an M-file it determines might be improved. For example, a common M-Lint message is that a variable is defined but never used in the M-file.

- “Accessing M-Lint” on page 7-25
- “M-Lint Graphical User Interface (GUI)” on page 7-25
- “Making Changes Based on M-Lint Messages” on page 7-28

**Accessing M-Lint**

You can get M-Lint messages using

- The `mlint` function. See the `mlint` reference page for instructions.
- A graphical user interface (GUI) for M-Lint in the Current Directory browser Directory Reports, as described in the remainder of this section. You can also access this GUI from the Editor or Profiler.

**M-Lint Graphical User Interface (GUI)**

1. Create, run, and debug an M-file, or multiple M-files in a directory.

2. In the Current Directory browser, navigate to the directory that contains the M-files you want to check with M-Lint. In the toolbar, select the M-Lint Code Check Report from the Directory Reports listing. For the example shown, go to `$matlabroot/matlab/help/techdoc/matlab_env/examples` to use M-Lint to check `lengthoffline.m`.

The M-Lint 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 potential problem or improvement opportunity.

Click line number to open the M-file in the Editor at that line.
3 For each message, review the message and code and make changes to the code based on the message as described here:

- Click the line number to open the M-file in the Editor at that line.
- Review the M-Lint message in the report and change the code in the M-file, based on the message.
- Note that in some cases, you should not make any changes based on the M-Lint messages because the M-Lint messages do not apply to that specific situation. M-Lint does not provide perfect information about every situation. 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, suppress the message by adding %#ok to the end of the statement in the M-file.
- 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 as you resolve problems with the updated file. Use the File Comparison Report, a tool that can help you identify the changes you made in the updated file. For more information, see “File Comparison Report” on page 7-22.
- If 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, see “Making Changes Based on M-Lint Messages” on page 7-28.

4 Run and debug the file(s) again to be sure you have not introduced any inadvertent errors.

5 If the M-Lint report is already displayed, click the Refresh button to update it, or run the M-Lint report again. 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:

- Review the MATLAB Programming and Programming Tips documentation.
- Use the Help browser Index and Search features to look in the documentation for terms presented in the M-Lint messages.

Other techniques to help you identify problems in and improve your M-files include:

- Syntax highlighting features in the Command Window and Editor
- Error messages generated when you run the M-file
- Debugging tools, namely the Editor/Debugger and debugging functions
- Profiler for improving performance

Example Using M-Lint Messages to Improve Code

An example file, lengthofline.m, is included with MATLAB in $matlabroot/matlab/help/techdoc/matlab_env/examples.

To run the M-Lint Code Check report for lengthofline.m, use the Current Directory browser to navigate to the $matlabroot/matlab/help/techdoc/matlab_env/examples directory. Select the M-Lint Code Check report from the list of reports on the toolbar. Note that lengthofline.m is not on the MATLAB path by default. You can run the file or open it in the Editor when $matlabroot/matlab/help/techdoc/matlab_env/examples is the current directory.

The M-Lint Code Check report appears, with its list of messages suggesting improvements you can make to lengthofline.m.
## M-Lint Code Check Report

### Contents
- No messages

### Syntax
- No messages

### 

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22:</td>
<td>The value assigned here to variable 'nohandle' is never used</td>
</tr>
<tr>
<td>23:</td>
<td>NUMEL(x) is usually faster than PROD(SIZE(x))</td>
</tr>
<tr>
<td>24:</td>
<td>Array 'nofunc' is constructed using subscripting. Consider preallocating for speed</td>
</tr>
<tr>
<td>25:</td>
<td>Use STRCMPI(str1,str2) instead of using LOWER in a call to STRCMPI</td>
</tr>
<tr>
<td>26:</td>
<td>NUMEL(x) is usually faster than PROD(SIZE(x))</td>
</tr>
<tr>
<td>31:</td>
<td>Array 'data' is constructed using subscripting. Consider preallocating for speed</td>
</tr>
<tr>
<td>32:</td>
<td>Use</td>
</tr>
<tr>
<td>33:</td>
<td>Use</td>
</tr>
<tr>
<td>34:</td>
<td>Use</td>
</tr>
<tr>
<td>35:</td>
<td>Array 'dim' is constructed using subscripting. Consider preallocating for speed</td>
</tr>
<tr>
<td>36:</td>
<td>Use of brackets [] is unnecessary. Use parentheses to group, if needed</td>
</tr>
<tr>
<td>37:</td>
<td>Format string does not agree with argument count</td>
</tr>
</tbody>
</table>
The following table describes each message and demonstrates a way to change the file, based on the message.

<table>
<thead>
<tr>
<th>Message</th>
<th>Code</th>
<th>Explanation and Updated Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>22: The value assigned here to variable 'nothandle' is never used</td>
<td>nothandle = ~ishandle(hline); for nh = 1:prod(size(hline)) notline(nh) = ~ishandle(hline(nh)) ...</td>
<td>In line 22, nothandle is assigned a value, but nothandle is 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. nothandle = ~ishandle(hline); for nh = 1:numel(hline) notline(nh) = nothandle(nh) ...</td>
</tr>
<tr>
<td>23: NUMEL(x) is usually faster than PROD(SIZE(x))</td>
<td>for nh = 1:prod(size(hline))</td>
<td>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 for nh = 1:numel(hline)</td>
</tr>
<tr>
<td>24: Array 'notline' is constructed using subscripting. Consider preallocating for speed</td>
<td>nothandle = ~ishandle(hline); for nh = 1:numel(hline) notline(nh) = ~ishandle(hline(nh)) ...</td>
<td>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 Arrays in the MATLAB Programming documentation. In the example, add a new line to preallocate notline before the loop. notline = false(size(hline)); for nh = 1:numel(hline) notline(nh) = nothandle(nh) ...</td>
</tr>
<tr>
<td>Message — — Code</td>
<td>Explanation and Updated Code (Continued)</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>24: Use STRCMPI(str1,str2) instead of using LOWER in a call to STRCMP</td>
<td>While <code>strcmpi('line',get(hline(nh),'type'))</code> converts the result of the get function to a lowercase string before doing the comparison, the <code>strcmpi</code> function ignores the case while performing the comparison, with advantages that include more efficiency. Change the line to <code>notline(nh) = ~strcmpi('line',get(hline(nh),'type'))</code>;</td>
<td></td>
</tr>
<tr>
<td>24 notline(nh)=~ishandle(hline(nh))</td>
<td></td>
<td>~strcmp('line',lower(get(hline(nh),'type')));</td>
</tr>
<tr>
<td>28: NUMEL(x) is usually faster than PROD(SIZE(x))</td>
<td>See the same message and explanation reported for line 23. Change the line to <code>for nl = 1:numel(hline)</code></td>
<td></td>
</tr>
<tr>
<td>28 for nl = 1:prod(size(hline))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34: Array 'data' is constructed using subscripting. Consider preallocating for speed</td>
<td>See the same message and explanation reported for line 24. Add this line before the loop <code>data = cell(size(fdata));</code></td>
<td></td>
</tr>
<tr>
<td>33 for nd = 1:length(fdata) 34 data{nd} = getfield(flds,fdata{nd});</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Message — — Code

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 34: Use dynamic fieldnames with structures instead of GETFIELD. Type 'doc struct' for more information | 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 Using Dynamic Field Names in the MATLAB Programming documentation. Change the line to  
|   |   |
| 34 data{nd} = getfield(flds,fdata{nd}); | data{nd} = flds.fdata{nd}); |

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 38: Use || instead of | as the OR operator in conditional statements | While | (the elementwise 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 the lines to  
|   |   |
| 38 if isempty(data{3}) | ...  
39 (length(unique(data{1}(:)))==1 | ...  
40 length(unique(data{2}(:)))==1 | ...  
41 length(unique(data{3}(:)))==1 | ...  
| if isempty(data{3}) || ...  
(length(unique(data{1}(:)))==1 || ...  
length(unique(data{2}(:)))==1 || ...  
length(unique(data{3}(:)))==1) |

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 43: Array 'dim' is constructed using subscripting. Consider preallocating for speed | See the same message and explanation reported for line 24. Before the first line of the loop  
| 43 dim(nl) = 2; | 29 for nl = 1: numel(hline)  
add the line  
dim = len; |
<table>
<thead>
<tr>
<th>Message – Code</th>
<th>Explanation and Updated Code (Continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>49: Use of brackets [ ] is unnecessary. Use parentheses to group, if needed</td>
<td>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 and add unnecessary processing. Change the line to</td>
</tr>
<tr>
<td>49 len(nl) = sum([sqrt(dot(temp',temp'))]);</td>
<td>len(nl) = sum(sqrt(dot(temp',temp')));</td>
</tr>
<tr>
<td>56: Format string does not agree with argument count</td>
<td>The number of %s in the warning statement is 2, but the number of trailing arguments is 4. Because this format will be used twice, the code is correct as written. M-Lint was not able to recognize that and so issued the message. In this case, you do not want to change the code, but you also do not want to see the M-Lint message. To suppress the M-Lint message, add %#ok to the end of the statement. Change line 57 to</td>
</tr>
</tbody>
</table>
| 55 warning ('lengthofline:FillWithNaNs',... | 'Lengths','NaNs','Dimensions','NaNs')
56 '
%\n%\n%\n%) |
57 'Lengths','NaNs','Dimensions','NaNs') | 'Lengths','NaNs','Dimensions','NaNs')
%#ok |
You can view the M-file with all of these changes. Navigate to the $matlabroot/matlab/help/techdoc/matlab_env/examples directory and open lengthofline2.m.

Updated M-Lint report after changing the file lengthofline.m based on M-Lint messages. No messages are reported.
Profiling for Improving Performance

One way to improve the performance of your M-files is using profiling tools. MATLAB provides the M-file Profiler, a graphical user interface that is based on the results returned by the `profile` function. Use the Profiler to help you determine where you can modify your code to make performance improvements. This section covers the following topics:

- “What Is Profiling?” on page 7-35—Profiling assesses where time is being spent in your M-code.
- “The Profiler” on page 7-38—A graphical user interface for viewing where the time is being spent in your M-code.
- “Profile Summary Report” on page 7-42—Describes the summary report produced by the Profiler.
- “Profile Detail Report” on page 7-45—Describes the detail reports produced by the Profiler.
- “The profile Function” on page 7-51—The function on which the Profiler is based, `profile`.

What Is Profiling?

Profiling is a way to measure where a program spends its time. Using the MATLAB Profiler, you can identify which functions in your code consume the most time. You can then determine why you are calling them and look for ways to minimize their use. 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.

**The Profiling Process: 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 7-42 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 7-45 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 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 part 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.
The Profiler

The Profiler is a tool that shows you where an M-file is spending its time. This section covers

- “Opening the Profiler” on page 7-38
- “Running the Profiler” on page 7-39
- “Profiling a Graphical User Interface” on page 7-41
- “Profiling Statements from the Command Window” on page 7-41
- “Changing Fonts for the Profiler” on page 7-42

For information about the reports generated by the Profiler, see “Profile Summary Report” on page 7-42 and “Profile Detail Report” on page 7-45.

Opening the Profiler

You can use any of the following methods to open the Profiler.

- Select Desktop-> Profiler from the MATLAB desktop.
- Select Tools->Open Profiler from the menu in the MATLAB Editor.
- Select one or more commands in the Command History window, right-click to view the context menu, and choose the Profile Code option.
- Enter the following command in the Command Window:

  ```matlab
  profile viewer
  ```
Running the Profiler
This is a quick summary. Details follow.

1 Type profile viewer to open the Profiler.
2 Type statement to run.
3 Click Start Profiling.

To profile an M-file or a line of code, follow these steps:

1 In the Run this code field in the Profiler, enter the statement you want to run.

You can run this example

\[ [t,y] = \text{ode23}('\text{lotka}',[0 2],[20;20]) \]

as the code is provided with MATLAB demos. It runs the Lotka-Volterra predator-prey population model. For more information about this model, type lotkadem0, which runs the demonstration.
If you are running a statement you previously profiled in the current MATLAB session, select the statement from the list box and skip to step 3.

2 Click **Start Profiling** (or press **Enter** after entering 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.

When the profiling is finished, the **Profile time** indicator becomes black and shows the length of time the Profiler ran.

This is not the actual time that your statements took to run, but is the time elapsed from when you clicked **Start Profiling** until you clicked **Stop Profiling**. 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.

3 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 7-42.
Profiling for Improving Performance

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 the 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, do the following.

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, do the following:

1 In the Profiler, clear the **Run this code** field and click the **Start Profiling** button.

2 In the Command Window, enter and run the statements you want to profile.

The status bar in the desktop reports **Profiler on** when MATLAB is not busy and the Profiler is running.

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. For more information, click the Help button in the dialog box.
3. Click Apply or OK.
4. In the Profiler, click the refresh button to update the display.

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. To sort the functions by this time value, click the Self Time link at the top of the column.
Following is the summary report for the Lotka-Volterra model described in “Example: Using the profile Command” on page 7-53.

To print a summary report, clicking 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 7-45 for more information.
## Profile Summary
Generated 08-Apr-2004 14:59:27

<table>
<thead>
<tr>
<th>Function name</th>
<th>Calls</th>
<th>Total Time</th>
<th>Self Time</th>
<th>Total Time Plot (dark band = self time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ode23</td>
<td>1</td>
<td>0.921 s</td>
<td>0.491 s</td>
<td></td>
</tr>
<tr>
<td>surfun/private/customArguments</td>
<td>1</td>
<td>0.170 s</td>
<td>0.100 s</td>
<td></td>
</tr>
<tr>
<td>cellmember</td>
<td>1</td>
<td>0.180 s</td>
<td>0.080 s</td>
<td></td>
</tr>
<tr>
<td>ndget</td>
<td>11</td>
<td>0.050 s</td>
<td>0.040 s</td>
<td></td>
</tr>
<tr>
<td>ismember</td>
<td>1</td>
<td>0.070 s</td>
<td>0.040 s</td>
<td></td>
</tr>
<tr>
<td>remat</td>
<td>2</td>
<td>0.030 s</td>
<td>0.030 s</td>
<td></td>
</tr>
<tr>
<td>surfun/private/callfinalize</td>
<td>1</td>
<td>0.190 s</td>
<td>0.030 s</td>
<td></td>
</tr>
<tr>
<td>matlab/lode (Built-in function)</td>
<td>41</td>
<td>0.030 s</td>
<td>0.030 s</td>
<td></td>
</tr>
<tr>
<td>surfun/private/codyrmissions</td>
<td>1</td>
<td>0.030 s</td>
<td>0.020 s</td>
<td></td>
</tr>
<tr>
<td>matlab/newfunc (Built-in function)</td>
<td>1</td>
<td>0.010 s</td>
<td>0.010 s</td>
<td></td>
</tr>
<tr>
<td>double superimpose</td>
<td>1</td>
<td>0.010 s</td>
<td>0.010 s</td>
<td></td>
</tr>
<tr>
<td>matlab/maps (Built-in function)</td>
<td>166</td>
<td>0.10 s</td>
<td>0.10 s</td>
<td></td>
</tr>
<tr>
<td>cellstr</td>
<td>1</td>
<td>0.010 s</td>
<td>0.010 s</td>
<td></td>
</tr>
<tr>
<td>surfun/private/dataevents</td>
<td>1</td>
<td>0.10 s</td>
<td>0.10 s</td>
<td></td>
</tr>
</tbody>
</table>

Click any column label to sort by column.
Click any function name to display 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 parts, summarized below. By default, the Profile Detail report includes all seven parts, although, depending on the function, not every part contains data. You can customize the display to include only parts you are interested in—see “Controlling the Contents of the Detail Report Display” on page 7-45. The following sections provide more detail about each part.

- “Profile Detail Report Header” on page 7-47 — Provides general information about the function.
- “Parent Files” on page 7-47 — Provides information about the parent function.
- “Busy Lines” on page 7-48 — Lists the lines in the function that used the greatest amount of processing time.
- “Child Files” on page 7-48 — Lists the functions called by this function, with links to Profile Detail reports for these functions.
- “M-Lint Results” on page 7-49 — Lists the lines in the functions that M-lint highlighted.
- “File Coverage” on page 7-50 — Provides statistics about the lines of code in the function that executed while profiling was on.
- “File Listing” on page 7-50 — Includes the source code for the function, if it is an M-file.

To return to the Profile Summary report from the Profile Detail report, click the home button in the toolbar.

Controlling the Contents of the Detail Report Display

You can determine which parts are included in the display by selecting them and then clicking the Refresh button. The following sections provide more detail about each part of this report.
Select parts of report to display and click Refresh.

funfun/private/odearguments (1 call, 0.170 sec)
Generated 06-Apr-2004 16:32:17
M-function in file C:\Program Files\MathWorks\matlabR14\toolbox\funfun/private/odearguments.m
[Copy to new window for comparing multiple runs]

Parents (calling functions)

<table>
<thead>
<tr>
<th>Filename</th>
<th>File Type</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>odes23</td>
<td>M-function</td>
<td>1</td>
</tr>
</tbody>
</table>

Lines where the most time was spent

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Code</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>146</td>
<td>rtol = odeget(options,'RelTol')...</td>
<td>1</td>
<td>0.030 s</td>
<td>17.6%</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>fo = feval(ode,t0,y0,sys();...</td>
<td>1</td>
<td>0.020 s</td>
<td>11.8%</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>end</td>
<td>1</td>
<td>0.020 s</td>
<td>11.8%</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>if (margin(ode) == 2) ...</td>
<td>1</td>
<td>0.020 s</td>
<td>11.8%</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>if strcmp(solver,'ode23s')</td>
<td>2</td>
<td>0.020 s</td>
<td>11.8%</td>
<td></td>
</tr>
<tr>
<td>Other lines &amp; overhead</td>
<td>0.060 s</td>
<td>36.3%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td>0.170 s</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>
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 Profiler detail reports for the two runs. Do not make changes to M-files provided with MathWorks products, that is, files in $matlabroot/toolbox directories.

Parent Files

To include the Parents part in the detail report, click the Show parent files check box. This part of the report provides information about the parent function, with a link to its detail report.
Busy Lines
To include information about the lines of code that used the most amount of processing time in the detail report, click the **Show busy lines** check box. Click a line number to view that line of code in the source listing.

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Code</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>146</td>
<td>rtol = odeget(options,'RelTol')...</td>
<td>1</td>
<td>0.040 s</td>
<td>20.6%</td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>dstate = superriorfloat(t0,y0...</td>
<td>1</td>
<td>0.020 s</td>
<td>13.0%</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>end</td>
<td>1</td>
<td>0.020 s</td>
<td>14.3%</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>if strcmp(solver,'ode15i')</td>
<td>2</td>
<td>0.020 s</td>
<td>14.3%</td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>htry = abs(odeget(options,'Init...</td>
<td>1</td>
<td>0.010 s</td>
<td>7.1%</td>
<td></td>
</tr>
</tbody>
</table>

Other lines & overhead

| Totals       |                                           |       | 0.140 s    | 100%   |           |

Child Files
To include the **Children** part of the detail report, click the **Show child files** check box. This part 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 on its name.
M-Lint Results
To include the M-Lint results part in the detail report display, click the Show M-Lint results check box. This part of the report provides information generated by M-Lint about the function.

<table>
<thead>
<tr>
<th>M-Lint results</th>
<th>Line number</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>61</td>
<td>EXIST with two input arguments is faster than with</td>
</tr>
<tr>
<td></td>
<td>79</td>
<td>EXIST with two input arguments is faster than with</td>
</tr>
</tbody>
</table>
**File Coverage**

To include the **Coverage results** part in the detail report display, click the **Show file coverage** check box. This part of the report provides statistical information about the number of lines in the code that executed during the profile run.

<table>
<thead>
<tr>
<th>Coverage results</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[Show coverage for parent directory]</td>
<td></td>
</tr>
<tr>
<td>Total lines in file</td>
<td>185</td>
</tr>
<tr>
<td>Non-code lines (comments, blank lines)</td>
<td>26</td>
</tr>
<tr>
<td>Code lines (lines that can run)</td>
<td>137</td>
</tr>
<tr>
<td>Code lines that did run</td>
<td>52</td>
</tr>
<tr>
<td>Code lines that did not run</td>
<td>85</td>
</tr>
<tr>
<td>Coverage (did run/can run)</td>
<td>37.96 %</td>
</tr>
</tbody>
</table>

**File Listing**

To include the **File listing** part in the detail report display, click the **Show file listing** check box. If the file is an M-file, the **Profile Detail** report includes a column listing the execution time for a particular line, a column listing the number of times the line was called, and the source code for the function.

In the file 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 color to highlight the lines of code with the longest execution time. The darker the shade of color, 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 other lines of code in the listing, 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. This section describes

- “profile Function Syntax Summary” on page 7-52
- “Example: Using the profile Command” on page 7-53
- “Accessing Profiler Results” on page 7-54
- “Saving Profile Reports” on page 7-56
profile Function Syntax Summary

Here is a summary of the main forms of profile. For details about these and other options, type doc profile. Some people use profile simply to see the child functions; see also depfun for that purpose.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile on</td>
<td></td>
<td>Starts profile, clearing previously recorded statistics.</td>
</tr>
<tr>
<td>-detail level</td>
<td></td>
<td>Specifies the level of function to be profiled. Level can be either:</td>
</tr>
<tr>
<td>-history</td>
<td></td>
<td>Specifies that the exact sequence of function calls is to be recorded.</td>
</tr>
<tr>
<td>profile off</td>
<td></td>
<td>Suspends profile.</td>
</tr>
<tr>
<td>profile resume</td>
<td></td>
<td>Restarts profile without clearing previously recorded statistics.</td>
</tr>
<tr>
<td>profile clear</td>
<td></td>
<td>Clears the statistics recorded by profile.</td>
</tr>
<tr>
<td>profile viewer</td>
<td></td>
<td>Opens the Profiler, a graphical user interface and displays the information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gathered as an HTML-formatted report.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: If you run the obsoleted syntax profile report, the profile function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls this syntax.</td>
</tr>
<tr>
<td>s = profile('status')</td>
<td></td>
<td>Displays a structure containing the current profile status.</td>
</tr>
<tr>
<td>stats = profile('info')</td>
<td></td>
<td>Suspends profile and displays a structure containing profile results.</td>
</tr>
</tbody>
</table>
Example: Using the profile Command
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: [27x1 struct]
   FunctionHistory: [2x766 double]
   ClockPrecision: 1.0000e-007
   Name: 'MATLAB'
   ClockSpeed: 1000
   ```
To save the profile report, use the `profsave` command. This command stores the profile information in separate HTML files, for each function listed in the `FunctionTable` field of the structure, `p`.

```
profsave(p)
```

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`.

### Accessing Profiler Results

The `profile` command returns results in a structure. This example illustrates how you can access these results:

1. To start `profile`, specifying the `detail` and `history` options, type in the Command Window.

```
profile on -detail builtin -history
```

The detail option specifies that built-ins should be included in the profile data. 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. Get the structure containing profile results.

```
stats = profile('info')
```

```
stats =

    FunctionTable: [34x1 struct]
    FunctionHistory: [2x968 double]
    ClockPrecision: 1.1111e-007
    Name: 'MATLAB'
    ClockSpeed: 1000
```
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, MATLAB builtin function.

```matlab
stats.FunctionTable
```

41x1 struct array with fields:
- `FunctionName`
- `FileName`
- `Type`
- `NumCalls`
- `TotalTime`
- `TotalRecursiveTime`
- `Children`
- `Parents`
- `ExecutedLines`

5 View one of the structures in the `FunctionTable` field.

```matlab
stats.FunctionTable(3)
```

```
ans =

FunctionName: 'ode23'
FileName: [1x55 char]
Type: 'M-function'
NumCalls: 1
TotalTime: 0.6810
TotalRecursiveTime: 0.6810
Children: [11x1 struct]
Parents: [0x1 struct]
ExecutedLines: [132x4 double]
IsRecursive: 0
AcceleratorMessages: {1x0 cell}
```
To view the history data generated by profile, view the `FunctionHistory` field. The history data is a 2-by-n array. The first row contains Boolean values where 1 means entrance into a function and 0 (zero) 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 Reports**
To save the profile report, use the `profsave` command.

This command stores the profile information in separate HTML files, for each function listed in the `FunctionTable` field of the structure, `p`.

```
profsave(p)
```

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(p,'mydir')
```
Publishing Results

Publishing to HTML, XML, LaTeX, Word, and PowerPoint Using Cells (p. 8-2)

Use cells to publish an M-file scripts to a popular output format, including the code, comments, and results.

Marking Up Text in Cells for Publishing (p. 8-9)

Prepare an M-file for publishing.

Publishing M-Files Using Cells (p. 8-15)

Publish the file and set preferences for publishing.

Notebook for Publishing to Word (p. 8-18)

Create an M-book in Microsoft Word, enter commands, and perform other basic tasks.

Defining MATLAB Commands as Input Cells for Notebook (p. 8-23)

Make text in the M-book become a MATLAB command.

Evaluating MATLAB Commands with Notebook (p. 8-27)

Run the MATLAB commands in the M-book.

Printing and Formatting an M-Book (p. 8-33)

Control styles and print M-books.

Configuring Notebook (p. 8-39)

Set up Notebook for use with your version of Word.

Notebook Feature Reference (p. 8-41)

Alphabetical listing of features.
Publishing to HTML, XML, LaTeX, Word, and PowerPoint Using Cells

- “Overview of Publishing” on page 8-2
- “Example of Publishing Without Text Markup” on page 8-3
- “Example of Publishing with Text Markup” on page 8-4

Overview of Publishing
When you have completed writing and debugging an M-file script, use the Editor cell features to quickly publish the M-file and its results in any of several presentation formats: HTML, XML, LaTeX, or, when the applications are installed, Microsoft Word or PowerPoint. This allows you to share your work with others, presenting not only the code, but also commentary on the code and results from running the file.

Publishing features evaluate each cell in the M-file script and display the contents of the cell along with the results in a nicely formatted document. For example, published documents include output to the Command Window and figures, and bold headings for each section of the file. The cells in the Editor you use for publishing are the same ones you might already have used for improving your code as described in “Rapid Code Iteration Using Cells” on page 6-63.

If you are using the Help browser, watch the Publishing M Code from the Editor video demo for an overview of the major functionality.

This is the overall process to publish an M-file using cell features in the Editor:

1 Enable cell mode and define cells as described in steps 1 through 3 in “Rapid Code Iteration Using Cells” on page 6-63. For formatted publishing, all comments must appear at the start of a cell, before the code. Comments appearing after code in a cell appear as unformatted comments in code in the output.

2 Use Cell -> Insert Text Markup to insert markup symbols in the M-file comments to stylize the text for the presentation format, for example, displaying selected text as bold or monospace. For details, see “Marking Up Text in Cells for Publishing” on page 8-9.
3 Select **File -> Publish To**, and select the format in which you want to publish the M-file: HTML, XML, LaTeX, Word, or PowerPoint. For details, see “Publishing M-Files Using Cells” on page 8-15.

4 Change Editor/Debugger Publishing and Publishing Images preferences to adjust the output. For example, you can choose to include or exclude the code from the output. For details, see “Modifying Published Output Via Preferences” on page 8-17.

MATLAB publishes the M-file by writing the cell titles, comment text, and code to a file using the specified format. MATLAB also evaluates the cells and writes the results of the evaluation to the output file as well. Any figures created during the evaluation are saved as graphics files, and are shown with the results.

**Example of Publishing Without Text Markup**

This is based on the M-file used in “Example—Evaluate Cells” on page 6-71, as shown here.
Select File -> Publish to HTML to produce the following result.

Example of Publishing with Text Markup
This simple example adds text markup to the sine_wave.m file used in “Example of Publishing Without Text Markup” on page 8-3 to produce the following published HTML document. General instructions for marking up M-files for publishing follow this example.
Add a title for the document.

Display comment lines using TeX format.

Make selected comment text appear in monospace.

Reduce the size of the figure output in the document using Preferences for Publishing Images.

Plot Sine Wave

Calculate and plot a sine wave.

Contents
- Calculate and Plot Sine Wave
- Modify Plot Properties

Calculate and Plot Sine Wave

Define the range for \(x\).

\[ 0 \leq x \leq 6\pi \]

Calculate and plot \(y = \sin(x)\).

```matlab
x = 0:0.125:6*pi;
y = sin(x);
plot(x,y)
```

Modify Plot Properties

```matlab
title('Sine Wave', 'FontWeight', 'bold')
xlabel('x')
```
1 Add an overall title for the published document
   a Add a blank line at the top of the file.
   b Select Cell -> Insert Text Markup -> Cell Title. MATLAB adds the following in the new blank line and adds a blank line beneath it.
      % TITLE
      The %% indicates the start of a new cell, where a cell is a section of an M-file.
   c Type over the text TITLE, replacing it with Plot Sine Wave. Add a comment about the overall file in line 2. Type
      % Calculate and plot a sine wave.
      and add a blank line beneath it for better readability.
      You can add any overall comments about the file in the lines following the this title. You cannot add code after the first title and before the next cell (line starting with %%) if you want the first title to appear as the overall document title.

2 Display equations in comments with symbols and Greek characters using the TeX format. For a list of symbols you can display and the character sequence to create them, see the String property on the MATLAB reference page for graphics text properties.
   a Position the cursor in the line 5, Define the range for x.
   b Use text markup to insert a comment containing the equation
      \[ 0 \leq x \leq 6\pi \]
      Select Cell -> Insert Text Markup -> Insert TeX Equation.
      MATLAB inserts the following lines that contain a sample equation you will replace.
      %
      % $$e^{i\pi} + 1 = 0$$
      %
      The sample equation is the text between the set of $$, and is highlighted.
c Type the following TeX equation to replace the sample equation.
\[ 0 \leq x \leq 6\pi \]
The three new lines that will display the TeX equation in the published document appear as follows in the M-file.

```latex
\%
% $0 \leq x \leq 6\pi$
%```

3 Display selected comment text in a monospaced font.

a Position the cursor in the comment in line 9

```latex
% Calculate and plot \( y = \sin(x) \).
```

b To make the equation \( y = \sin(x) \) appear in monospace in the published document, add the \( | \) symbol before and after the equation so it appears as follows.

```latex
% Calculate and plot \( |y = \sin(x)| \).
```

4 To reduce the size of the resulting figure, select File -> Preferences -> Editor/Debugger -> Publishing Images. In the Preferences dialog box, for Resize image, select the check box Restrict height to and enter 200 for the number of pixels. Click OK to close the dialog box.

5 Select File -> Save and Publish to HTML.

The HTML file displays in your system's default browser, as shown at the start of this example, “Example of Publishing with Text Markup” on page 8-4.

6 By default, MATLAB stores the HTML document, `sine_wave.html`, and the associated image files in `d:/mymfiles/html` for this example.
The file `sine_wave.m` now appears as shown in the following illustration.
Marking Up Text in Cells for Publishing

To publish an M-file and results, mark up the file using cell features. This adds and formats comments for the published results. You can include the markup as you write the basic code, mark up the file after you've written the code, or do both. The markup applies to any of the available publishing options: HTML, XML, LaTeX, Word, and PowerPoint.

Any cell features you use for evaluating and improving your code will be used for publishing purposes as well. The “Example of Publishing Without Text Markup” on page 8-3 shows how the cells used for improving an M-file appear when the M-file is published. You might want to change the existing cells for publishing purposes, but note that of course changes the cells for evaluation purposes as well. For example, to use text markup and format comments in the output document, the comments must appear at the start of a cell, before any code.

Mark up comment text in one of two ways:

- Use Cell -> Insert Text Markup menu items to format the code, which automatically inserts the markup symbols for you, or
- Type the markup symbols directly in the code. Note that what you type is the same as the code that results if you instead use the equivalent menu item.

The following table describes each markup option and how to use it, and refers to “Example of Publishing with Text Markup” on page 8-4.
<table>
<thead>
<tr>
<th>Format</th>
<th>Menu Item to Produce Format (Cell -&gt; Insert Text Markup -&gt; Item)</th>
<th>Resulting Code</th>
<th>Published Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall document heading</td>
<td><strong>Cell Title</strong>&lt;br&gt;Add a blank line at the top of the M-file, select this menu item, and replace TITLE in the resulting text with the document heading you want.&lt;br&gt;See step 1 in the example.</td>
<td><strong>%% TITLE</strong>&lt;br&gt;Add any overall comments about the file in the lines following this title. But do not add code after the first title and before the next cell (line starting with <code>%%</code>) if you want the first title to appear as the overall document title.&lt;br&gt;In the example, it is <strong>Plot Sine Wave</strong></td>
<td>Formatted as a top level heading (h1 in HTML), using a large bold font.&lt;br&gt;Note that the title text automatically appears in bold in the M-file and in the resulting published document.</td>
</tr>
<tr>
<td>Section title (also known as a cell title)</td>
<td><strong>Cell Title</strong>&lt;br&gt;Position the cursor at the start of a cell, select this menu item, and replace TITLE in the resulting text with the cell title you want.</td>
<td><strong>%% TITLE</strong>&lt;br&gt;In the example, these are <strong>Calculate and Plot Sine Wave</strong> <strong>Modify Plot Properties</strong></td>
<td>Formatted as a heading (h2 in HTML), using a medium size, bold font.&lt;br&gt;Note that the title text automatically appears in bold in the M-file and in the resulting published document.</td>
</tr>
</tbody>
</table>
### Marking Up Text in Cells for Publishing

<table>
<thead>
<tr>
<th>Format</th>
<th>Menu Item to Produce Format (Cell -&gt; Insert Text Markup -&gt; Item)</th>
<th>Resulting Code</th>
<th>Published Results (Continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Descriptive text</td>
<td><strong>Descriptive Text</strong>&lt;br&gt;Position the cursor where you want to add a formatted comment, select this menu item, and replace the resulting DESCRIPTIVE TEXT with your comment.</td>
<td>% DESCRIPTIVE TEXT</td>
<td>Text appears as a formatted comment in the output.&lt;br&gt;Note that descriptive text must appear before the first line of code in a cell.</td>
</tr>
<tr>
<td>Bold text</td>
<td><strong>Bold Text</strong>&lt;br&gt;Select the text within a comment that you want to appear in bold and then select this menu item.&lt;br&gt;If no text is selected, inserts a new bold comment with sample text, and you replace the sample text.</td>
<td>% <em>Bold text</em></td>
<td>Text appears in bold.&lt;br&gt;&lt;strong&gt;Bold text&lt;/strong&gt;</td>
</tr>
<tr>
<td>Format</td>
<td>Menu Item to Produce Format (Cell -&gt; Insert Text Markup -&gt; Item)</td>
<td>Resulting Code</td>
<td>Published Results (Continued)</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------</td>
<td>----------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Monospaced text</td>
<td><strong>Monospaced text</strong>&lt;br&gt;Select the text within a comment that you want to appear in a monospaced font and then select this menu item. See step 3 in the example.&lt;br&gt;If no text is selected, inserts a new monospaced comment with sample text, and you replace the sample text.</td>
<td>%</td>
<td>Monospaced text</td>
</tr>
<tr>
<td>Format</td>
<td>Menu Item to Produce Format (Cell -&gt; Insert Text Markup -&gt; Item)</td>
<td>Resulting Code</td>
<td>Published Results (Continued)</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Indented text</td>
<td><strong>Preformatted Text</strong>&lt;br&gt;Position the cursor before the line where you want to add indented text and select this menu item. Replace the sample text inserted with the text you want, including tabs and spaces.&lt;br&gt;A line of preformatted text must begin with a <code>%</code> symbol, followed by two or more spaces.</td>
<td>%&lt;br&gt;% Preformatted&lt;br&gt;% Text&lt;br&gt;% The blank comment lines above and below the preformatted lines distinguish the lines in between as preformatted.</td>
<td>The indents, spacing, and line lengths in the M-file are preserved in the output.&lt;br&gt;Preformatted&lt;br&gt;Text</td>
</tr>
<tr>
<td>Bullets</td>
<td><strong>Bulleted List</strong>&lt;br&gt;Position the cursor before the line where you want to add a bulleted list and select this menu item. Replace the sample text inserted, ITEM 1 and ITEM 2, with the text you want.</td>
<td>%&lt;br&gt;% * Item1&lt;br&gt;% * Item2&lt;br&gt;% Items appear in a bulleted list.&lt;br&gt;• Item1&lt;br&gt;• Item2</td>
<td></td>
</tr>
<tr>
<td>Format</td>
<td>Menu Item to Produce Format (Cell -&gt; Insert Text Markup -&gt; Item)</td>
<td>Resulting Code</td>
<td>Published Results (Continued)</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------</td>
<td>----------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Equations and symbols</td>
<td><strong>TeX Equation</strong> Position the cursor before the line where you want to add an equation or symbols and select this menu item. Replace the sample text inserted, $e^{\pi i} + 1 = 0$, with the TeX equation you want. See step 2 in the example. For a list of symbols you can display and the character sequence to create them, see the String property on MATLAB graphics reference page for text properties.</td>
<td>% $e^{\pi i} + 1 = 0$ % The blank comment lines above and below the equation line, as well as the $$ before and after the equation text, distinguish the TeX equation.</td>
<td>Equation and symbols appear in TeX output format. $e^{\pi i} + 1 = 0$</td>
</tr>
<tr>
<td>Links (for HTML output)</td>
<td>Enter comment text that includes a valid Internet address.</td>
<td>% <a href="http://www.mathworks.com">http://www.mathworks.com</a></td>
<td><a href="http://www.mathworks.com">http://www.mathworks.com</a></td>
</tr>
</tbody>
</table>
Publishing M-Files Using Cells

When you publish an M-file that contains cells and text markup, MATLAB produces an output document consisting of the M-file code, comments, and results.

How to Publish an M-File

After adding cells and text markup to an M-file, select File -> Publish To and select an output format from those listed in the menu: HTML, XML, LaTeX, Word, or PowerPoint. If the M-file contains unsaved changes, the menu item becomes Save and Publish To.

You can also publish to the default output format using the publish button in the Editor toolbar.

MATLAB displays the published document in the appropriate tool for the selected output format:

- HTML displays in your system’s default Web browser.
- XML displays in the MATLAB Editor.
- LaTeX displays in the MATLAB Editor.
- Word displays in Microsoft Word.
- PowerPoint displays in Microsoft PowerPoint.

Note: Publishing to Microsoft Word and to PowerPoint features are available only on PC and Macintosh systems that have the applications installed.

The published file contains the formatted comments, code including syntax highlighting, and results (in gray to distinguish it from code) for each cell, and for all cells in the M-file. When code produces a figure, the last figure generated in a cell appears in the published file. It also contains a Contents heading at the top of the file with a bulleted list of links to the named cells in the rest of the document.
Function Alternative
From the Command Window, run the publish function to run the M-file and publish the results. See the reference page for options you can set.

About Published M-Files
Published Filenames and Locations
MATLAB names the published file the same as the M-file that produced it, adding the relevant extension for the selected output format: .html, .xml, .tex, .doc, or .ppt. MATLAB stores this output file, along with supporting files such as images of figure windows, in the html subdirectory under the directory containing the M-file you published.

For example, when you publish d:/mymfiles/sine_wave.m to HTML, MATLAB creates a directory d:/mymfiles/html that includes the published document sine_wave.html. Any figure windows produced by running the M-file appears as an image file in the directory, for example, sine_wave_img.png. TeX equations are image files as well: in the example, the equation file is sine_wave_eq_eq####.png. In addition, MATLAB creates a thumbnail file for the document, sine_wave_img_thumbnail.png in the example, used in the Visual Directory in the Current Directory browser.
Publishing Code that Displays Hyperlinks in Command Window

If the M-file you publish contains statements that display hyperlinks in the MATLAB Command Window, the published document shows the code rather than the hyperlinks.

For example

```matlab
disp('<a href="http://www.mathworks.com">Link to MathWorks</a>')
```

displays

```
Link to MathWorks
```

in the Command Window. You can click the link to go to the MathWorks Web site. When that `disp` statement is in an M-file you publish, the hyperlink tag and the text between it, that is,

```matlab
<a href="http://www.mathworks.com">Link to MathWorks</a>
```

rather than the link, appears in the published document.

Similarly results occur if you include

```matlab
help matlab_functioname
```

in an M-file.

Modifying Published Output Via Preferences

Use preferences to control execution, output, and options related to images created during publishing. For details about these preferences, click the Help button in the Preferences dialog box for those panels to view the following sections in the online documentation:

- “Publishing Preferences for the Editor/Debugger”
- “Publishing Images Preferences for the Editor/Debugger”
Notebook for Publishing to Word

Notebook allows you to access the numeric computation and visualization software of MATLAB from within the word processing environment, Microsoft Word. 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. Notebook is useful for creating electronic or printed records of MATLAB sessions, class notes, textbooks or technical reports. This section introduces basic Notebook capabilities:

- “Creating an M-Book” on page 8-18
- “Entering MATLAB Commands in an M-Book” on page 8-21
- “Protecting the Integrity of Your Workspace in M-Books” on page 8-22
- “Ensuring Data Consistency in M-Books” on page 8-22

**Note**  Notebook is available only on Windows and Macintosh systems that have Microsoft Word installed.

Creating an M-Book

This section includes

- “Creating an M-Book from MATLAB” on page 8-19
- “Creating an M-Book While Running Notebook” on page 8-20
- “Opening an Existing M-Book” on page 8-20
- “Converting a Word Document to an M-Book” on page 8-21
Creating an M-Book from MATLAB

To create a new M-book from within MATLAB, type

    notebook

at the prompt. If you are running Notebook for the first time, you may need to configure it. See “Configuring Notebook” on page 8-39 for more information.


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 8-39.

Notebook adds the Notebook menu to the Word menu bar. Use this menu, illustrated below, to access Notebook features.
Creating an M-Book While Running Notebook

With Notebook running, you can create a new M-book by selecting New M-book from the Word File menu.

Opening an Existing M-Book

You can use the notebook command to open an existing M-book

notebook filename

where filename is the M-book you want to open, or you can simply double-click on an M-book file in a Windows file management tool, such as Explorer.

When you double-click on 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.
Converting a Word Document to an M-Book

To convert a Word document to an M-book, follow these steps:

2. From the Insert menu, select the File.
3. Select the file you want to convert.
4. Click OK.

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

• Define the command as an input cell
• Evaluate the input cell

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 could also use calc zones to isolate related commands in a section of the M-book. You can 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 use the Editor/Debugger 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 via Notebook. If you debug while evaluating from Notebook, you might experience problems with MATLAB.
Defining MATLAB Commands as Input Cells for Notebook

To define a MATLAB command in a Word document as an input cell:

1. Type the command into the M-book as text. For example,

   This is a sample M-book.

   \[ a = \text{magic}(3) \]

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.

   \[ [a = \text{magic}(3)] \]

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 8-33.

For information about defining other types of input cells, see

- “Defining Cell Groups for Notebook” on page 8-23
- “Defining Autoinit Input Cells for Notebook” on page 8-25
- “Defining Calc Zones for Notebook” on page 8-25
- “Converting an Input Cell to Text with Notebook” on page 8-26

For information about evaluating the input cells you define, see “Evaluating MATLAB Commands with Notebook” on page 8-27.

**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 the 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 8-28 for information about evaluating a cell group. For information about undefining a cell group, see “Ungroup Cells” on page 8-47.

Creating a Cell Group for Notebook
To create a cell group:

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.

```
[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 simply 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 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 8-27 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 you want to include in the calc zone, you 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 8-30 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:

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 8-33. When you convert an input cell to text, Notebook also converts the corresponding output cell to text.
Evaluating MATLAB 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

   ```matlab
   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

   ```matlab
   [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.

   ```matlab
   [a = magic(3) ]
   ```

   ```matlab
   [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 8-33.

For more information about evaluating MATLAB commands in an M-book, see

- “Evaluating Cell Groups with Notebook” on page 8-28
- “Evaluating a Range of Input Cells with Notebook” on page 8-29
- “Evaluating a Calc Zone with Notebook” on page 8-30
- “Evaluating an Entire M-Book” on page 8-30
- “Using a Loop to Evaluate Input Cells Repeatedly with Notebook” on page 8-31.
- “Converting Output Cells to Text with Notebook” on page 8-32
- “Deleting Output Cells with Notebook” on page 8-32

**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):

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 8-23.

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 it 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 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.

\[
\begin{align*}
    t &= 0:pi/100:2*pi; \\
    y &= \sin(t); \\
    \text{plot}(t,y)
\end{align*}
\]

Evaluating a Range of Input Cells with Notebook

To evaluate more than one MATLAB command contained in different but contiguous input cells:

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:

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 8-25.

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:

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:

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:

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

This section describes

- “Printing an M-Book” on page 8-33
- “Modifying Styles in the M-Book Template” on page 8-33
- “Choosing Loose or Compact Format for Notebook” on page 8-34
- “Controlling Numeric Output Format for Notebook” on page 8-35
- “Controlling Graphic Output for Notebook” on page 8-35

Printing an M-Book

You can print all or part of an M-book by selecting **File -> 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. 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.
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 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:

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:

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 8-36
- “Suppressing Graphic Output for Individual Input Cells in Notebook” on page 8-37
- “Sizing Graphic Output in Notebook” on page 8-37
- “Cropping Graphic Output in Notebook” on page 8-38
- “Adding White Space Around Graphic Output in Notebook” on page 8-38
- “Specifying Color Mode in Notebook” on page 8-38
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:

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, all graphic output is directed 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:

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.

Sizing Graphic Output in Notebook

To set the default size of embedded graphics in an M-book:

1. Select Notebook -> Notebook Options.
2. In the Notebook Options dialog box, use the Units, Height, and Width 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.
Cropping Graphic Output in Notebook
To crop an embedded figure to cut off areas you do not want to show:

1  Select the graphic by clicking the left mouse button anywhere in the figure.
2  Hold down the Shift key.
3  Drag a sizing handle toward the center of the graphic.

Adding White Space Around Graphic Output in Notebook
You can add white space around an embedded figure by moving the boundaries of a graphic outward. Select the graphic, then hold down the Shift key and drag a sizing handle away from the graphic.

Specifying Color Mode in Notebook
If you print graphic output that includes surfaces or patches, the output uses 16-color mode by default. To use 256-color mode:

1  Select Notebook -> Notebook Options.
2  Clear the Use 16-Color Figures check box in the Notebook Options dialog box.
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.
Configuring Notebook

After you install Notebook but before you begin using it, you must configure it. (Notebook is installed as part of the MATLAB installation process. For more information, see the MATLAB installation documentation for your platform.)

In Word versions for Office 2000 or higher, before configuring Notebook, you must specify that Word can use the Notebook macros. 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 launching 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.

To configure Notebook:

1. In the MATLAB command window, type
   ```
   notebook -setup
   ```
   Notebook prompts you to specify which version of Microsoft Word you are using.

   Welcome to the utility for setting up the MATLAB Notebook for interfacing MATLAB to Microsoft Word

   Choose your version of Microsoft Word:
   [1] Microsoft Word 97
   [4] Exit, making no changes

   Microsoft Word Version:
2 Type the number that corresponds to your version. For example, type 3 if you have Microsoft Word 2000 XP.

Notebook performs the setup. If Notebook cannot find all of the necessary files, it will prompt you to specify the locations of the files, including the Microsoft Word executable (winword.exe) and the template file (normal.dot).

When setup is complete, the following message appears:

Notebook setup is complete.
Notebook Feature Reference

This section provides reference information about each of the Notebook features, listed alphabetically. To use these features, select them from the Notebook menu:

- “Bring MATLAB to Front” on page 8-41
- “Define Autoinit Cell” on page 8-41
- “Define Calc Zone” on page 8-42
- “Define Input Cell” on page 8-42
- “Evaluate Calc Zone” on page 8-43
- “Evaluate Cell” on page 8-43
- “Evaluate Loop” on page 8-44
- “Evaluate M-Book” on page 8-44
- “Group Cells” on page 8-44
- “Hide Cell Markers” on page 8-45
- “Notebook Options” on page 8-45
- “Purge Selected Output Cells” on page 8-45
- “Toggle Graph Output for Cell” on page 8-46
- “Undefine Cells” on page 8-46
- “Ungroup Cells” on page 8-47

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 8-25.

---

**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 8-30. 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 8-23. For information about evaluating input cells, see “Evaluating MATLAB Commands with Notebook” on page 8-27.

Evaluate Calc Zone

Evaluate Calc Zone sends the input cells in the current calc zone to MATLAB to be evaluated. A calc zone is a contiguous block of related text, input cells, and output cells that describes a specific operation or problem.

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 8-30.

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 8-27. For information about evaluating the entire M-book, see “Evaluating an Entire M-Book” on page 8-30.

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 8-31.

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 8-30.

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 8-23. For information about converting a cell group to individual input cells, see the description of the “Ungroup Cells” on page 8-47.

**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 8-33 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 8-32.
**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 Graphic Output* in the 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 8-36 and “Suppressing Graphic Output for Individual Input Cells in Notebook” on page 8-37 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 8-33. For information about deleting output cells, see the description of the “Purge Selected Output Cells” on page 8-45.
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 the “Defining Cell Groups for Notebook” on page 8-23.
Source Control

| Source Control Interface on PC Platforms (p. 9-2) | Select and view the source control system, add files, check files into and out of source control, undo a check-out, remove files, view file history, compare file versions, and more. |
| Source Control Interface on UNIX Platforms (p. 9-27) | Select and view the source control system, check files into and out of source control, and undo a check-out. |
Source Control Interface on PC Platforms

If you use a source control system (SCS) to manage your files, you can perform source control interface actions on M-files and Simulink and Stateflow® files within MATLAB, Simulink, and Stateflow. You can interface to your source control system by using menus from a graphical user interface (GUI), or by using functions from the MATLAB Command Window.

MATLAB, Simulink, and Stateflow do not perform source control functions, but only provide an interface to your own source control system. This means, for example, that you can open a file in the MATLAB Editor and modify it without checking it out. However, the file will remain read-only so that you cannot accidentally overwrite the source control version of the file.

The Source Control Interface works with any source control system that conforms to the Microsoft Common Source Control standard. Several vendors provide a Microsoft Source Code Control API wrapper for other source control systems.

Note For additional help on Command Window access to source control operations, enter help verctrl in the MATLAB Command Window.

• “Selecting and Viewing the Source Control System” on page 9-3
• “Adding Files to the Source Control System” on page 9-4
• “Checking Files Out of the Source Control System” on page 9-8
• “Checking Files Into the Source Control System” on page 9-11
• “Getting the Latest Version of Files from the Source Control System” on page 9-15
• “Undoing the Check-Out” on page 9-18
• “Removing Files from the Source Control System” on page 9-19
• “Showing File History” on page 9-20
• “Comparing the Working Copy of a File to the Latest Version in Source Control” on page 9-22
• “Displaying Source Control Properties of a File” on page 9-24
• “Starting the Source Control System Client” on page 9-26
Selecting and Viewing the Source Control System

To select the source control system to interface, follow these steps:

1. From the MATLAB desktop, select Preferences from the File menu. You can also select this from Simulink and Stateflow model and library windows.

   The Preferences dialog box opens.

2. Click the + for General and then select Source Control.

   The currently selected system is shown. The list box will be populated by the systems installed in the machine that support the Microsoft Common Source Control standard. The default selection is None.
3 Select the system you want to use from the Source control system list.

4 Click OK.

Function Alternative for Viewing the Source Control System

1 To view the currently selected system, type

```matlab
cmopts
```

MATLAB displays the current source control system. For example:

```matlab
ans =
Microsoft Visual SourceSafe
```

2 To view all of the source control systems installed on your computer, type

```matlab
list = verctrl ('all_systems')
```

MATLAB displays all the source control systems currently installed in your computer. For example:

```matlab
list =
'Microsoft Visual SourceSafe'
'Jalindi Igloo'
```

Adding Files to the Source Control System

You can add a single file or multiple files to the source control system. Note that the file is first added to the source control system using the Add command, not the Check In command.

Adding a Single File

To add a file to the source control system:

1 Select Source Control -> Add to Source Control from the File menu in the MATLAB Editor, Simulink model, or Stateflow model.

The MATLAB Add to source control dialog box opens.
If you want to add the file to the source control system and keep it checked out so you can continue making changes, select **Keep checked out**. This is selected by default. If you have comments, type them in the **Comments** area. Your comments will be submitted whether or not you select **Keep checked out**.

3 Click **OK**.

The file is added to the source control system. If you did not save the file before adding it to the source control system, you are prompted to save the files before adding them to your source control system.

If you did not keep the file checked out and you keep the file open, note that it is a read-only version.
Adding Multiple Files

To add multiple files to the source control system:

1. Select the files in the MATLAB Current Directory window.

![Current Directory Window](image)

2. Right-click the selected files.

3. Select Source Control -> Add to Source Control from the pop-up menu.

The MATLAB Add to source control dialog box opens.
If you want to add the files to the source control system and keep them checked out so you can continue making changes, select **Keep checked out**. This is selected by default. If you have comments, type them in the **Comments** area.

Your comments will be submitted whether or not you select **Keep checked out**.

**5** Click **OK**.

The files are added to the source control system. If you did not save the files before adding them to the source control system, they are automatically saved when they are added.

If you did not keep the files checked out and you keep the files open, note that they are read-only versions.
Function Alternative for Adding File to Source Control

Use `add` as the first argument in the `verctrl` function to add a file to the source control system. Keep the following points in mind:

- Use the `comment` argument to store comments version of the file(s).
- The `verctrl` function with the `add` argument returns a logical 1 to the workspace if the file has changed on disk or a logical 0 to the workspace if the file has not changed on disk.
- You can add a single file or multiple files.

The `verctrl` function with the `add` argument takes this form.

```matlab
fileChange=verctrl('add',{'H:\myFile.m', 'H:\rsimtfdemo.exe',
'H:\mfile-report.html', 'H:\simulink-summary.html'},
'comment', 0);
```

**Note** To use the `verctrl` command, you can create a Windows handle or use 0. For more information about creating handles, enter `doc verctrl` in the MATLAB Command Window.

Checking Files Out of the Source Control System

You can check out a single file or multiple files.

Checking Out a Single File

To check out a single file from the source control system,

1 Select **Source Control -> Check Out** from the **File** menu in the MATLAB Editor, Simulink model, or Stateflow model.

The MATLAB **Check out file(s)** dialog box opens.
2 Click OK.

The file is checked out from the source control system and is available to you for editing.

**Note** The Comments text area will not be included in the Check out file(s) dialog box if the source control system does not support “comments” on file check-out.

**Checking Out Multiple Files**

To check out multiple files from the source control system,

1 Select the files in the MATLAB Current Directory window.

2 Right-click the selected files.

3 Select Source Control -> Check Out from the pop-up menu.

The MATLAB Check out file(s) dialog box opens.
Source Control

**Current Directory - H:**

<table>
<thead>
<tr>
<th>All Files</th>
<th>File Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB_SAMPLE...</td>
<td>MAT-file</td>
</tr>
<tr>
<td>MATLAB_SAMPLE...</td>
<td>MAT-file</td>
</tr>
<tr>
<td>MATLAB_SAMPLE...</td>
<td>MAT-file</td>
</tr>
<tr>
<td>MATLABSAMPLE...</td>
<td>XML File</td>
</tr>
<tr>
<td>mfile-report.html</td>
<td>HTML File</td>
</tr>
<tr>
<td>myFile.m</td>
<td>M-file</td>
</tr>
<tr>
<td>rsimfxfdemo.exe</td>
<td>EXE File</td>
</tr>
<tr>
<td>simulink-summary....</td>
<td>HTML File</td>
</tr>
<tr>
<td>tempreport.rpt</td>
<td>RPT File</td>
</tr>
</tbody>
</table>

**Check out file(s)**

Check out files:

H:\myFile.m  
H:\rsimfxfdemo.exe  
H:\mfile-report.html  
H:\simulink-summary.html

Comments:
4 Click OK.

The files are checked out from the source control system and are available to you for editing.

**Function Alternative for Checking Out Files**

Use `checkout` as the first argument in the `verctrl` function to check a file out of the source control system. Keep the following points in mind:

- You can use the `comment` argument to store comments with this version of the file(s).
- The `verctrl` function with the `checkout` argument returns a logical 1 to the workspace if the file has changed on disk or a logical 0 to the workspace if the file has not changed on disk.
- You can check out a single file or multiple files.

The `verctrl` function with the `checkout` argument takes this form.

```matlab
fileChange=verctrl('checkout',{'H:\myFile.m','H:\rsimtfdemo.exe','H:\mfile-report.html','H:\simulink-summary.html'},'comment',0);
```

**Note**  To use the `verctrl` command, you can create a Windows handle or use 0. For more information about creating handles, enter `doc verctrl` in the MATLAB Command Window.

**Checking Files Into the Source Control System**

You can check in one or more MATLAB M-files, Simulink models, or Stateflow models.

**Checking In a Single File**

To check in a single file into the source control system:

1 Select **Source Control -> Check In** from the **File** menu in the MATLAB Editor, Simulink model, or Stateflow model:

   The MATLAB Check in file(s) dialog box opens.
If you want to check in the file to the source control system and keep it checked out so you can continue making changes, select Keep checked out. If you have comments, type them in the Comments area.

Your comments will be submitted whether or not you select Keep checked out.

3 Click OK.

The file is checked into the source control system. If you did not save the file before checking it in, it is automatically saved when it is checked in.

If you did not keep the file checked out and you keep the file open, note that it is a read-only version.

**Checking In Multiple Files**

To check in multiple files to the source control system,
1 Select the files in the MATLAB **Current Directory** window.

![Current Directory window](image)

2 Right-click the selected files.

3 Select **Source Control -> Check In** from the pop-up menu.

The MATLAB **Check in file(s)** dialog box opens.
4 If you want to check in the files to the source control system and keep them checked out so you can continue making changes, select **Keep checked out**. If you have comments, type them in the **Comments** area.

Your comments will be submitted whether or not you select **Keep checked out**.

5 Click **OK**.

The files are checked into the source control system. If you did not save the files before checking them in, they are automatically saved when they are checked in.

If you did not keep the files checked out and you keep the files open, note that they are read-only versions.
Function Alternative for Checking In Files

Use `checkin` as the first argument in the `verctrl` function to check files into the source control system. Keep the following points in mind:

- Use the `comment` argument to store comments with this version of the file(s).
- The `verctrl` function returns a logical 1 to the workspace if the file has changed on disk or a logical 0 to the workspace if the file has not changed on disk.
- You can check in a single file or multiple files.
- The files can be open or closed when you use `checkin`.

The `verctrl` function with the `checkin` argument takes this form.

```matlab
fileChange = verctrl('checkin', {'H:\myFile.m', 'H:\rsimtfdemo.exe', 'H:\mfile-report.html', 'H:\simulink-summary.html'}, 'comment', 0);
```

**Note** To use the `verctrl` command, you can create a Windows handle or use 0. For more information about creating handles, enter `doc verctrl` in the MATLAB Command Window.

Getting the Latest Version of Files from the Source Control System

You can get the latest version of a file from the source control system for viewing and compiling, but not editing. You can get a single file, a single directory, multiple files, or multiple directories. The file or files will be tagged read-only. The list of files should contain either files or directories but not both.

Getting the Latest Version of a Single File

To get the latest version of a single file:

1 Select **Source Control -> Get Latest Version** from the **File** menu in the MATLAB editor, Simulink model, or Stateflow model.

   The MATLAB **Get latest version** dialog box opens.
2 Click OK.

**Getting the Latest Versions of Multiple Files**

To get the latest versions of multiple files:

1 Select the files in the MATLAB **Current Directory** window.
2 Right-click the selected files.

3 Select **Source Control -> Get Latest Version** from the pop-up menu.

The MATLAB **Get latest version** dialog box opens.

![Get latest version dialog box](image)

4 Click **OK**.

**Function Alternative for Getting Latest Version**

Use `get` as the first argument in the `verctrl` function to get a file from the source control system. Keep the following points in mind:

- Note that the `verctrl` function with the `get` argument returns a logical 1 to the workspace if the file has changed on disk or a logical 0 to the workspace if the file has not changed on disk.
- You can get a single file or multiple files.

The `verctrl` function with the `get` argument takes this form.

```
fileChange=verctrl('get',{'H:\myFile.m', 'H:\rsimtfdemo.exe', 'H:\mfile-report.html', 'H:\simulink-summary.html'}, 0);
```
To use the `verctrl` command, you can create a Windows handle or use 0. For more information about creating handles, enter `doc verctrl` in the MATLAB Command Window.

**Undoing the Check-Out**

You can undo the check-out for a file. The file remains checked in, without any of the changes you made since you checked it out. You can undo the check-out of a single file or multiple files.

**Note** You will lose the changes you have made since you checked out the file. To save these changes when undoing the check-out, use the **Save As** item from the **File** menu.

1. Select **Source Control -> Undo Check-Out** from the **File** menu in the MATLAB Editor, Simulink model, or Stateflow model.

   The MATLAB **Undo check out** dialog box opens.

2. Click **OK**.
**Function Alternative for Undoing a Check-Out**

Use the uncheckout as the first argument in the verctrl function to undo a check-out. Note that the verctrl function with the uncheckout argument returns a logical 1 to the workspace if the file has changed on disk or a logical 0 to the workspace if the file has not changed on disk.

The verctrl function with the uncheckout argument takes this form.

```matlab
fileChange=verctrl('uncheckout',{'H:\myFile.m',
'H:\rsimtdemo.exe', 'H:\mfile-report.html',
'H:\simulink-summary.html'}, 0);
```

**Note**  To use the verctrl command, you can create a Windows handle or use 0. For more information about creating handles, enter doc verctrl in the MATLAB Command Window.

---

**Removing Files from the Source Control System**

You can remove a single file or multiple files from the source control system. To remove a file from the source control system:

1. Select **Source Control -> Remove from Source Control** from the **File** menu in the MATLAB Editor, Simulink model, or Stateflow model.

   The MATLAB **Remove from source control** dialog box opens.

   ![Remove from source control dialog box](image)

   1. Select `H:\myFile.m`
   2. Click **OK**.
**Function Alternative for Removing File from Source Control**

Use `remove` as the first argument in the `verctrl` function to remove a file or group of files from source control system. Note that the `verctrl` function with the `remove` argument does not return anything.

The `verctrl` function with the `remove` argument takes this form.

```matlab
verctrl('remove',{'H:\myFile.m', 'H:\rsimtfdemo.exe', 'H:\mfile-report.html', 'H:\simulink-summary.html'}, 0);
```

**Note** To use the `verctrl` command, you can create a Windows handle or use 0. For more information about creating handles, enter `doc verctrl` in the MATLAB Command Window.

---

**Showing File History**

You can show the history of a single file or multiple files in the source control system. To show the history of a file,

1. Select **Source Control -> Show History** from the **File** menu in the MATLAB Editor, Simulink model, or Stateflow model.

The dialog boxes returned are specific to the source control system being used. For example, if Microsoft Visual SourceSafe is the currently selected source control system, then the **History Options** dialog box is returned.
2 Enter the appropriate label, date, and user information and click **OK**.

The Microsoft Visual SourceSafe **History** dialog box opens.

![History dialog box](image)

**Function Alternative for Showing File History**

Use `history` as the first argument in the `verctrl` function to show the history of the file in the source control system. Note that the `verctrl` function with the `history` argument returns a logical 1 to the workspace if the file has changed on disk or a logical 0 to the workspace if the file has not changed on disk.

The `verctrl` function with the `history` argument takes this form.

```matlab
fileChange=verctrl('history',{'D:\file1.ext','D:\file2.ext'},0);
```

**Note** To use the `verctrl` command, you can create a Windows handle or use 0. For more information about creating handles, enter `doc verctrl` in the MATLAB Command Window.
Comparing the Working Copy of a File to the Latest Version in Source Control

You can use the **Differences** option to compare the current working copy of a file on disk with the latest checked-in version of the file in the source control system. Note that you can only show differences on one file at a time, not multiple files.

**Comparing M-Files**

To show differences of an M-file:

1. Select **Source Control -> Differences** from the **File** menu in the MATLAB Editor.

   A dialog box from the currently selected source control system opens. For example, if Microsoft Visual SourceSafe is the currently selected source control system, then the **Difference Options** dialog box opens.

   ![Difference Options Dialog Box]

2. Click **OK**.

   The Microsoft Visual SourceSafe **Differences** dialog box opens. This compares the working copy of the file to the latest checked-in version of the file.
Function Alternative for Showing File Differences

Use `isdiff` as the first argument in the `verctrl` function to return a Boolean value to the window, which indicates whether or not there are any differences between the current file on disk and the latest checked-in version of the file.

The `verctrl` function with the `isdiff` argument takes this form.

```matlab
fileChange = verctrl('isdiff', 'H:\myFile.m', 0)
```

This will return the following in the Command Window if the two copies of the file are different:

```matlab
1
```

Use `showdiff` as the first argument in the `verctrl` function to show the differences between the disk copy of a file and the latest checked-in version in the source control system. Note that the `verctrl` function with the `showdiff` argument does not return anything.

The `verctrl` function with the `showdiff` argument takes this form.

```matlab
verctrl('showdiff','D:\file.ext', handle);
```
Note  To use the `verctrl` command, you can create a Windows handle or use 0. For more information about creating handles, enter `doc verctrl` in the MATLAB Command Window.

Displaying Source Control Properties of a File
You can display the properties of a single file from the source control system. Note that you cannot display the properties of multiple files. To display the properties of a file,

1. Select **Source Control -> Show Properties** from the **File** menu in the MATLAB Editor, Simulink model, or Stateflow model.

A dialog box from the source control system being used opens. Below is a Microsoft Visual SourceSafe properties dialog box.
Function Alternative for Displaying File Properties

Use properties as the first argument in the verctrl function to display the properties of a file. Note that the verctrl function with the properties argument returns a logical 1 to the workspace if the file has changed on disk or a logical 0 to the workspace if the file has not changed on disk.

The verctrl function with the properties argument takes this form.

```matlab
fileChange=verctrl('properties','D:\file.ext');
```
Starting the Source Control System Client

To start the source control system:

1. Select **Source Control -> Start Source Control** from the **File** menu in the MATLAB Editor, Simulink model, or Stateflow model.

The dialog box from the currently selected source control system opens. Below is the Microsoft **Visual SourceSafe Explorer** dialog box.

![Visual SourceSafe Explorer](image)

Function Alternative for Starting the Source Control Client

Use `runscc` as the first argument in the `verctrl` function to start the currently selected source control system. The `verctrl` function with the `runscc` argument takes this form.

```matlab
verctrl('runscc', 0);
```

**Note** To use the `verctrl` command, you can create a Windows handle or use 0. For more information about creating handles, enter `doc verctrl` in the MATLAB Command Window.
Source Control Interface on UNIX Platforms

If you use a source control system (SCS) to manage your files, you can check M-files and Simulink and Stateflow files into and out of the source control system from within MATLAB, Simulink, and Stateflow.

MATLAB, Simulink, and Stateflow do not perform source control functions, but only provide an interface to your own source control system. This means, for example, that you can open a file in the MATLAB Editor and modify it without checking it out. However, the file will remain read-only so that you cannot accidentally overwrite the source control version of the file.

The Source Control Interface supports four popular source control systems, as well as a custom option:

- ClearCase from Rational Software
- CM Synergy from Telelogic
- PVCS Version Manager from Merant
- Revision Control System (RCS)
- Custom option — Allows you to build your own interface if you use a different source control system

You can interface to your source control system by using menus from a graphical user interface (GUI), or by using functions from the Command Window. There are some options you can perform using the MATLAB functions that are not available with the GUIs — these are noted in the instructions.

- “Selecting and Viewing the Source Control System” on page 9-27
- “Checking Files Into the Source Control System” on page 9-29
- “Checking Files Out of the Source Control System” on page 9-31
- “Undoing the Check-Out” on page 9-32

Selecting and Viewing the Source Control System

Specify the source control system using these steps:

1. Select Preferences from the File menu in the MATLAB Editor, Simulink model, or Stateflow model.

   The Preferences dialog box opens.
2 Click the + for General and then select Source Control.

The currently selected system is shown. The default selection is None.

3 Select the system you want to use from the Source control system list.

**Function Alternative for Viewing the Source Control System**

To view the currently selected system, type `cmopts` in the Command Window. MATLAB displays the current source control system. For example:

```matlab
ans =

   PVCS Source Control```
Setting a View and Checking Out a Directory — For ClearCase on UNIX Only
If you use ClearCase on a UNIX platform, do the following using ClearCase:

1 Set a view.

2 Check out the directory that you want to save files in, check files into, or check files out of.

You can now use the MATLAB, Simulink, or Stateflow interfaces to ClearCase to check files into and out of the directory that you checked out in step 2.

Checking Files Into the Source Control System
After creating or editing a file in the MATLAB Editor, Simulink, or Stateflow, save it, and then check in the file by following these steps:

1 Select Source Control -> Check In from the File menu in the MATLAB Editor, Simulink model, or Stateflow model. The Check In dialog box opens.
If you want to check in the file but keep it checked out so you can continue making changes, select **Keep checked out**. If you have comments, type them in the **Comments** area.

Your comments will be submitted whether or not you select **Keep checked out**.

3 Click **OK**.

The file is checked into the source control system. If you did not save the file before checking it in, it is automatically saved when it is checked in.

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 for Checking In Files**

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({'D:\file1.ext','D:\file2.ext'},'comments','string',
        'option','value')
```

For file, use the complete path. 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 to `on`.
- Check in a file even though it has not changed since the previous check in — set the `force` option to `on`.

The `comments` argument and the `lock` and `force` options apply to all files checked in.

After checking in the file, if you did not keep it checked out and have it open, note that it is a read-only version.
Example—Check In a File with Comments
To check in the file clock.m with a comment Adjustment for Y2K, type

    checkin('\matlabr12\mymfiles\clock.m','comments','Adjustment ...
    for Y2K')

For other examples, see the reference page for checkin.

Checking Files Out of the Source Control System
To check files out of the source control system using MATLAB, follow these steps:

1 Open the M-file, Simulink file, or Stateflow file you want to check out.
   The file opens and the title bar indicates it is read-only.

2 Select Source Control -> Check Out from the File menu in the MATLAB
   Editor, Simulink model, or Stateflow model. The Check Out dialog box
   opens

   ![Check Out Dialog Box]

3 To check out the version that was most recently checked in, select the Latest
   version option. To check out a specific version of the file, select the Version
   number option and type the version number in the field.

   To prevent others from checking out the file while you have it checked out,
   select Lock latest version. To check out a read-only version of the file, clear
   Lock latest version.
4 Click **OK**.

The file is checked out from the source control system and is available to you for editing.

**Function Alternative for Checking Out Files**

Use `checkout` to check a file out of the source control system. You can check out multiple files at once and specify check-out options. The `checkout` function takes this form.

```matlab
checkout({'D:\file1.ext','D:\file2.ext'},'option','value')
```

For `file`, use the complete path.

Use the `option` argument to

- Check out a read-only version of the file — set the `lock` option to `off`.
- Check out the file even if you already have it checked out — set the `force` option 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 checked out. The file can be open or closed when you use `checkout`.

**Example — Check Out a Specific Version of a File**

To check out the 1.1 version of the file `clock.m`, type

```matlab
checkout('\matlab\myfiles\clock.m','revision','1.1')
```

For other examples, see the reference page for `checkout`.

**Undoing the Check-Out**

You can undo the check-out for a file. The files remain checked in, without any of the changes you made since you checked them out. Select **Source Control -> Undo Check-Out**. from the **File** menu in the MATLAB Editor, Simulink model, or Stateflow model. There is no return dialog.

If you want to keep a local copy of your changes, use the **Save As** item from the **File** menu.
**Function Alternative for Undoing a Check-Out**

The undocheckout function takes this form.

```matlab
undocheckout({'D:\file1.ext','D:\file2.ext'})
```

Use the complete path for file. For example, to undo the check-out for the files `clock.m` and `calendar.m`, type

```matlab
undocheckout({'\matlab\mymfiles\clock.m','\matlab\mymfiles\calendar.m'})
```
Symbols

! function 3-6
% comment symbol 6-14
create comment 6-15
%% 6-64
, after functions 3-18
... in statements 3-11
; after functions 3-18
>> prompt in Command Window 3-2
{ block comment symbol 6-15

A
accelerators, keyboard 2-35
add 9-8
adding file to source control system on PC platforms 9-4
addpath 5-23
Array Editor 5-10
 preferences 5-16
arrays
editing 5-10
workspace 5-2
arrow keys
 Command Window usage 3-16
 Editor 6-21
ASCII files
 viewing contents of 5-38
asv 6-30
autocode 6-2
autoinit cells
 converting input cells to 8-41
 converting to input cells 8-42
 defining 8-25
AutoInit style
 definition of 8-34
automatic code generation 6-2
automatic completion of statement 3-12
automation startup option (automation server) 1-6
autosave 6-30

B
backup
 Editor autosave 6-30
bang (!) function 3-6
base workspace 5-8
batch mode for starting MATLAB 1-6
batch mode for starting MATLAB (UNIX) 1-10
blank spaces in MATLAB commands 3-8
block comments 6-15
 extending 6-16
bookmarks
 in files in Editor 6-22
 in Help browser 4-18
Boolean searching in Help browser 4-15
breaking long lines 3-11
breakpoints
 clearing (removing) 6-52
 clearing, automatically 6-53
 conditional 6-58
 disabling and enabling 6-52
 running file 6-43
 setting 6-40, 6-41
 types 6-40
Bring MATLAB to Front 8-41
browser
 Web, in MATLAB 2-29
bugs, reporting to The MathWorks 4-38
built-in editor 6-4
C
C/C++
   editing files in Editor 6-12
caching
   M-files 6-30
   search path 5-25
calc zones
   defining 8-25
   ensuring workspace consistency in M-books 8-22
   evaluating 8-30
   output from 8-30
callbacks
   in shortcuts 2-21
calling from MATLAB 3-6
capitalization in MATLAB 3-8
case sensitivity in MATLAB 3-8
cell groups
   converting to input cells 8-47
   creating 8-24
   definition of 8-23
   evaluating 8-28
   output from 8-28
cell markers
   defined 8-23
   hiding 8-45
   printing 8-33
cell mode 6-63
cell scripts 6-63
cells
   beep 6-70
   defining in M-files 6-64
   evaluating 6-69
   removing 6-68
   cells in M-File Editor 6-63
character set
   preference for MAT-files 2-56
checkin
   on PC platforms 9-15
   on UNIX platforms 9-30
checking in files
   on PC platforms 9-11
   on UNIX platforms 9-29
checking out files
   on PC platforms 9-8
   on UNIX platforms 9-31
   undoing on PC platforms 9-18
   undoing on UNIX platforms 9-32
checkout
   on PC platforms 9-11
   on UNIX platforms 9-32
c1c 3-20
clear 5-7
ClearCase source control system
   configuring on UNIX platforms 9-29
clearing
   Command Window 3-20
   clicking on multiple items 2-38
   clipboard 2-39
closing
   desktop tools 2-6
   MATLAB 1-13
   M-files 6-32
cmopts
   on PC platforms 9-4
   on UNIX platforms 9-28
code examples 6-2
code iteration 6-63
code resources 6-2
code samples
   sample code 6-2
   Collatz problem 6-36
Index

color
  general preferences 2-52
  indicators for syntax 3-9
  modes for printing M-books 8-38
  printing M-book 8-33

colors
  in M-files 6-19
  preferences in MATLAB 2-50

column numbers 6-20

COM
  entries for MATLAB 1-6
  startup options for MATLAB 1-6

command flags 1-4

Command History
  about 3-30
  deleting entries in 3-35
  file 3-31
  preferences 3-37
  printing 3-35
  running functions from 3-32

command line
  defined 3-2
  editing 3-10

command name
  automatic completion 3-12

command switches 1-4

Command Window
  bringing to front in Notebook 8-41
  clearing 3-20
  editing in 3-10
  help 4-3
  paging of output in 3-18
  preferences 3-27
  printing contents of 3-20
  prompt 3-2
  width 3-28

Command Window scroll buffer 3-28

commands
  executing a group of 2-21
  on multiple lines 3-11
  to operating system 3-6

comments
  adding/removing with any text editor 6-15
  adding/removing with Editor 6-14
  block 6-15
  color indicators 2-52
  creating in Editor 6-14
  multiline statements 6-17
  purpose 6-14
  using ... (ellipsis) 6-17
  within a line 6-17

comp.soft-sys.matlab 4-39

cmp-fs-nnn 4-39

comparing working copy to source control version
  on PC platforms 9-22

completing statements automatically 3-12

compression
  MAT-files 2-56

conditional breakpoints 6-58

configuration management
  See source control system interface on PC
  platforms or source control system
  interface on UNIX platforms

configuration, desktop 2-5

configuring Notebook 8-39

console mode 3-28

content of M-files, searching 5-40

Contents in Help browser
  synchronizing preference 4-28

Contents tab in Help browser
  description 4-7
  synchronizing with display 4-9

context menus 2-31

continuing long statements 3-11
control keys
  editing commands 3-16
  Editor 6-21
conversion
  Word document to M-book 8-21
cropping graphics
  in M-books 8-38
cssm 4-39
current directory
  at startup for MATLAB 1-3
  changing 5-32
  contents of 5-32
  field in toolbar 5-30
  relevance to MATLAB 5-29
  tool 5-30
Current Directory browser 5-30
  preferences 5-42

data consistency
  calc zones in M-books 8-22
  evaluating M-books 8-22
  in M-book 8-22
datatips 6-31
  example 6-47
dbclear 6-53
dbstop
  example 6-43
Debugger 6-1
debugger
  option for UNIX 1-8
debugging
  ending 6-51
  example 6-36
  features 6-39
M-files 6-33
  options 6-4
Notebook 8-22
  prompt 6-44
  stepping 6-45
  techniques 6-33
decimal places in output 3-19
defaults
  preferences for MATLAB 2-44
  setting in startup file for MATLAB 1-4
Define Autinit Cell 8-41
Define Calc Zone 8-42
Define Input Cell 8-42
delete 5-35
delete function
  preference for recycling 2-55
deleting
  files 5-35
deleting files 2-55
desktop
  color preferences 2-50
  configuration 2-5
  description 2-2
  font preferences for 2-44
  starting without 1-9
tools
  closing 2-6
  opening 2-4
windows
  closing 2-6
  opening 2-4
development environment for MATLAB 2-2
diary 3-21
dir 5-32
directories
   copying 5-36
   creating 5-34
   deleting 5-35
MATLAB
   caching 6-30
   renaming 5-35
   searching contents of 5-30
   *See also* current directory, search path
disabling
   breakpoints 6-52
display pane in Help browser 4-19
displaying
   output 3-18
displaying source control properties of a file 9-24
dividers for cells 6-64
documentation
   printing 4-30
   problems, reporting 4-40
   viewing 4-19
dots (...) 3-11
downloading
   M-files 4-38
dragging in the desktop 2-39

e
edit
   creating new M-file in Editor 6-8
editing
   in Command Window 3-10
   M-files 6-1
   outside of MATLAB 6-4
   without running MATLAB 6-10
Editor
   arranging documents 6-10
   closing files 6-32
go to
   bookmark 6-22
   function 6-23
   line number 6-22
   modifying values 6-22
   navigation keys 6-21
   opening files 6-8
   other text files 6-12
   rule displayed 6-20
   status bar
   function 6-20
   viewing values 6-31
editor
   built-in 6-4
   Editor, stand-alone (Windows) 6-10
   Editor/Debugger 6-1
      closing 6-12
      description 6-6
      example 6-36
      indenting 6-6
      preferences 6-11
   EDU>> prompt in Command Window 3-2
   ellipses (...) in statements 3-11
   Emacs key bindings in Editor 6-21
   Embed Figures in M-book 8-36
   embedding graphics
      in M-book 8-36
   encoding
      preference when saving 2-56
   ending MATLAB 1-13
   environment settings at startup 1-4
   error breakpoints
      stop for errors 6-60
   error message identifiers 6-62
   error messages
      in Command Window 3-4
error style
  definition 8-34
errors
  color indicators 2-52
  finding in M-files 6-33
  run-time 6-33
  syntax 6-33
Evaluate Calc Zone 8-43
Evaluate Cell 8-43
Evaluate Loop 8-44
Evaluate Loop dialog box 8-31
Evaluate M-Book 8-44
evaluating
  M-books, ensuring data consistency 8-22
  selection in Command History 3-32
  selection in Command Window 3-6
evaluating sections of M-file 6-69
example code 6-2
examples
  in documentation, index of 4-8
  running from Help browser 4-22
exe 3-6
executables
  running from MATLAB 3-6
executing
  group of statements 2-21
  M-files in Editor 6-31
existing code 6-2
exit
  confirmation 1-13
exiting MATLAB 1-13
ext startup option 1-8

F
  f button 6-23
  F Inc Search field 6-25
favorites in Help browser 4-18
feedback to The MathWorks 4-40
file exchange
  for M-files 4-38
file management system
  See source control system interface on PC
    platforms or source control system
    interface on UNIX platforms
filebrowser 5-30
files
  contents, viewing 5-38
  copying 5-36
  creating in the Current Directory browser 5-34
  deleting 5-35
  editing M-files 6-6
  log 1-6
  MATLAB related, listing 5-32
  naming 5-19
  opening 5-37
  operations in MATLAB 5-29
  renaming 5-35
  running 5-39
  source control on PC platforms 9-2
  source control on UNIX platforms 9-27
  viewing contents of 5-38
Find Files dialog box 5-40
finding
  files using Current Directory browser 5-40
  M-files 5-40
  string in M-files 5-40
  text in Command History 3-34
  text in Command Window 3-22
  text in current file 6-23
  text in M-files 6-24
  text in page of Help browser 4-21
finish.m file running when quitting 1-13
firewall 2-31
flags
  for startup 1-4
folders. See directories
font
  Help browser 4-29
  Help browser display pane 4-29
  Help Navigator 4-29
  preferences in MATLAB 2-44
format
  controlling numeric format in M-book 8-35
  preferences 3-28
format 3-19
  in M-book 8-35
function name
  automatic completion 3-12
function workspace 5-8
functions
  color indicators 2-52
  executing a group of 2-21
  help for 4-32
  reference page 4-2
  long (on multiple lines) 3-11
  multiple in one line 3-11
  naming 5-19

G
get 9-17
get latest version of file on PC platforms 9-15
graphical debugger 6-1
graphics
  controlling output in M-book 8-37
  embedding in M-book 8-36
  in M-books 8-35
graphing
  variables from the Workspace browser 5-8

H
gray breakpoint icons 6-42
  Group Cells 8-44

H
help
  functions 4-32
  in Command Window 4-34
  M-file 4-3
help 4-34
Help browser
  contents listing 4-7
  copying information from 4-22
  display pane 4-19
  font preferences 4-29
  index 4-10
  navigating 4-20
  printing help 4-30
  running examples from 4-22
  searching 4-12
Help Navigator 4-5
helpbrowser 4-4
Hide Cell Markers 8-45
highlighted search terms 4-13
history
  automatic log file 1-6
  source control on PC platforms 9-20
history 9-21
history of statements 3-30
history.m file 3-31
home 3-20
HTML
  editing files in Editor 6-12
  source, viewing in Help browser 4-22
HTML viewer in MATLAB 2-29
Index

I
incremental searching
   in Editor 6-25
indenting
   in Command Window 3-9
   in Editor 6-19
   nested functions 6-20
index
   examples in documentation 4-8
   Help browser 4-10
      results 4-11
      tips 4-11
initiation (init) file for MATLAB 1-4
input
   to MATLAB in Command Window 3-2
input cells
   controlling evaluation 8-30
   controlling graphic output 8-37
   converting autoinit cell to 8-42
   converting text to 8-42
   converting to autoinit cell 8-41
   converting to cell groups 8-47
   converting to text 8-26
   defining in M-books 8-23
   evaluating 8-27
   evaluating cell groups 8-28
   evaluating in loop 8-31
   maintaining consistency 8-22
   timing out during evaluation 8-44
   use of Word Normal style 8-26
Input style
   definition of 8-34
Internet proxy server 2-31
interrupting a running program 3-5
invalid breakpoints 6-42
isdiff 9-23
iterative programming 6-63

J
Java
   editing files in Editor 6-12
Java VM
   starting without 1-10

K
K>> prompt
   debugging mode 6-44
   keyboard statement 6-35
K>> prompt in Command Window 3-2
keyboard 6-35
keyboard shortcuts and accelerators 2-35
keys
   editing in Command Window 3-16
   Editor 6-21
   keyword
      color indicator 2-53
   keywords
      color indicators 2-52
      in documentation 4-10

L
license information 4-40
line
   in Editor 6-20
line breaks
   adding for long statements 3-11
line numbers 6-20
   going to 6-22
line wrapping 3-28
links
   in Help browser 4-21
load 5-6
locking files 9-31

Index-8
Index

log
  automatic 1-6
  file 1-6
  session 3-21
  statements 3-30
logfile startup option 1-6
long lines 3-11
lookfor 5-42
looping
to evaluate input cells 8-31
lowercase usage in MATLAB 3-8

evaluating all input cells 8-30
  modifying style template 8-33
  opening 8-20
  printing 8-33
  sizing graphic output 8-37
  styles 8-33
measuring performance of M-files 7-35
mvim, stand-alone MATLAB editor 6-10
membership Web page 2-42
message identifiers 6-62
M-file
  navigating 6-21
M-file help 4-3
  viewing in Current Directory browser 5-39
M-files
  appearance 6-19
  automatic saving 6-30
  colors in 6-19
  content, viewing 5-38
  creating 6-4
    from Command History 3-33
    in MATLAB directory 5-25
  creating new file 6-7
  debugging 6-1, 6-33
    options 6-4
  editing 6-1
    options 6-4
  editing without running MATLAB 6-10
  file association (Windows) 6-10
  finding 5-40
  naming 5-19
  opening 6-8
  pausing 6-35
  performance of 7-35
  printing 6-32
  profiling 7-35
  replacing content 6-24

M
MAT-files
  compatibility 2-56
  compression preference 2-56
  creating 5-4
  defined 5-4
  loading 5-6
  preferences 2-56
Mathtools.net 4-39
MATLAB
  commands, executing in a Word document 8-27
  files, listing 5-32
  path 5-18
  quitting 1-13
matlab.mat 5-6
matlabrc.m, startup file 1-4
matrices
  editing 5-10
M-books
  creating 8-18
  data consistency 8-22
  data integrity 8-22
  entering text and commands 8-21
M-files (continued)
running
at startup 1-6
at startup (UNIX) 1-10
from Command Window 3-5
from Current Directory browser 5-39
saving 6-29
search path 5-18
searching contents of 5-40
source control on PC platforms 9-2
source control on UNIX platforms 9-27
user-contributed 4-38
viewing description 5-44
Microsoft Word
converting document to M-book 8-21
specifying version and location 8-40
mkdir 5-35
M-Lint
suppressing messages 7-27
model files
source control on PC platforms 9-2
source control on UNIX platforms 9-27
more 3-18
mouse, right-clicking 2-31
multiple item selection 2-38
multiple lines for statements 3-11
multiprocessing 3-5
newsgroup 4-39
newsletters 4-39
nodesktop startup option 1-9
nojvm startup option 1-10
Normal style (Microsoft Word)
default style in M-book 8-33
defaults 8-34
used in undefined input cells 8-26
nosplash startup option 1-10
Notebook
configuring 8-39
devbugging 8-22
options 8-45
overview 8-18
platforms supported 8-18
notebook
basics 8-18
configuring 8-39
Notebook menu
Word menu bar 8-18
numbering lines 6-20
numeric format
controlling in M-book 8-35
output 3-19
preferences 3-28
O
open 5-38
opening files
Current Directory browser 5-37
openvar 5-11
operating system commands 3-6
optimizing performance of M-files 7-35
options
shutdown 1-13
startup 1-4
N
naming functions and variables 5-19
navigating
M-File 6-21
nested comments 6-16
nested functions
indenting 6-20
newsgroup 4-39
output
display
  format 3-19
  hidden 3-18
  hiding 3-18
in Command Window 3-2
paging 3-18
  spaces per tab 3-29
  spacing of 3-28
  suppressing 3-18
output cells
  converting to text 8-32
  purging 8-32
Output style
  definition 8-34

P
paging in the Command Window 3-18
path
  adding directories to 5-32
  changing 5-22
  description 5-18
  order of directories 5-19
  problems and recovering 5-27
  saving changes 5-25
  saving for future sessions 5-25
  viewing 5-22
  pathdef.m 5-20
    location 5-25
  pathtool 5-20
pausing execution of M-file 6-40
pcode
  error checking 6-34
PDF
  printing documentation files 4-30
  reader, preference for Help browser 4-28
performance
  improving for M-files 7-35
periods (...) 3-11
pop-up menus 2-31
precision
  output display 3-19
preferences
MATLAB, general 2-54
printing
  Command History 3-35
  Command Window contents 3-20
  documentation 4-30
  help 4-30
  M-files 6-32
printing an M-book
  cell markers 8-33
  color 8-33
  color modes 8-38
  defaults 8-33
problems, reporting to The MathWorks 4-38
product filter in Help browser
  preference 4-27
product version 2-43
profile 7-51
  example 7-53
profiling 7-35
programs
  running from MATLAB 3-6
  stopping while running 3-5
prompt
  in Command Window 3-2
  when debugging 6-44
properties
  source control on PC platforms 9-24
  tab completion 3-15
properties 9-25
publishing
   cells
      platforms supported 8-15
Purge Output Cells 8-45
purging output cells 8-32

Q
quitting
   confirmation 1-13
quitting MATLAB 1-13

R
R Inc Search field 6-25
rapid development 6-63
recall previous lines 3-12
recover deleted files 2-55
recycle function
   preference 2-55
red breakpoint icons 6-42
redo
   in desktop 2-39
reference pages 4-2
regserver startup option 1-6
remove 9-20
removing files from source control system 9-19
requirements
   MATLAB 1-2
results in MATLAB, displaying 5-11
revision control system
   See source control system interface on PC
   platforms or source control system
   interface on UNIX platforms
right-hand text limit 6-20
roadmap for documentation 4-8
rule
   in Editor 6-20
running
   M-files 5-39
   M-files in Editor 6-31
runsc 9-26
run-time errors 6-33

S
save 5-6
saving
   automatically in Editor 6-30
   MAT-files
      preferences 2-56
   M-files 6-29
script for startup 1-4
scroll buffer for Command Window 3-28
scrolling in Command Window 3-18
SCS
   See source control system interface on PC
   platforms or source control system
   interface on UNIX platforms
search path 5-18
problems and recovering 5-27
saving for future sessions 5-25
searching
   for M-files 5-40
   Help browser 4-12
      Boolean 4-15
      results 4-12
      text in page 4-21
   M-file content
      across files 5-40
searching (continued)
  text
    Command History 3-34
    Command Window 3-22
  text in current file 6-23
  text in M-files 6-24
  text, incrementally 6-25
section breaks
  in calc zones 8-42
selecting multiple items 2-38
semicolon (;)
  after functions 3-18
  between functions 3-11
separator in functions 3-11
session
  automatic log file 1-6
session log
  Command History 3-30
  diary 3-21
setting breakpoints 6-40
shadowed functions 5-19
shell escape 3-6
shortcut
  for MATLAB in Windows 1-2
  keys in MATLAB 2-35
shortcut keys
  Command Window editing 3-16
  Editor 6-21
shortcuts
  categories 2-28
  creating
    Command History 3-33
  defined 2-21
  deleting 2-28
  editing 2-28
  Editor 6-21
  file 2-24
  labels, hiding 2-28
  moving 2-28
  organizing 2-28
  toolbar 2-25
shortcuts.xml 2-24
Show Cell Markers 8-45
show file history on PC platforms 9-20
showdiff 9-23
shutdown
  MATLAB 1-13
  options 1-13
Simulink
  interfacing files with source control systems on
    PC platforms 9-2
  interfacing files with source control systems on
    UNIX platforms 9-27
smart recall 3-12
snapshot 7-8
source control system interface on PC platforms
  adding file 9-4
  checking in files 9-11
  checking out files 9-8
  comparing working copy to source control
    version 9-22
  displaying file properties 9-24
  get latest version of file 9-15
  preferences 9-3
  removing files 9-19
  selecting current system 9-3
  showing file history 9-20
  specifying 9-3
  starting source control system 9-26
  undoing file check-out 9-18
source control system interface on UNIX platforms 9-29
checking in files 9-29
checking out files 9-31
configuring ClearCase source control system 9-29
preferences 9-27
selecting current system 9-27
specifying 9-27
supported systems 9-27
undoing file check-out 9-32
spaces in MATLAB commands 3-8
spacing
output in Command Window 3-28
tabs in Command Window 3-29
splash screen
UNIX startup option 1-10
Windows startup option 1-6
stack
in Editor 6-44
viewing 5-8
stand-alone Editor 6-10
Start button 2-19
adding toolboxes 2-21
starting MATLAB
DOS 1-2
UNIX 1-3
Windows 1-2
starting source control system
on PC platforms 9-26
startup
directory for MATLAB 1-3
files for MATLAB 1-4
options for MATLAB 1-4
script 1-4
startup.m
location 1-5
startup file 1-4
Stateflow files
source control on PC platforms 9-2
source control on UNIX platforms 9-27
statements
defined 3-3
executing a group of 2-21
long (on multiple lines) 3-11
stepping through M-file 6-45
stopping a running program 3-5
stops
in M-files 6-40
stops (...) 3-11
strings
across multiple lines 3-11
color indicators 2-52
saving as Unicode 2-56
structures
tab completion 3-14
style preferences for text 2-44
styles in M-book
modifying 8-33
subfunction
displayed in Editor status bar 6-20
subfunctions
going to in M-file 6-23
suggestions to The MathWorks 4-40
support
technical 4-38
suppressing M-Lint messages 7-27
suppressing output 3-18
switches
for startup 1-4
Index

syntax
  color indicators 2-52
  color preferences in MATLAB 2-50
  coloring and indenting 3-9
  errors 6-33
  highlighting 6-19
system requirements
  MATLAB 1-2

T
  tab
    indenting in Editor 6-19
    spacing in Command Window 3-29
  tab completion 3-12
  table of contents for help 4-7
Technical Support
  contacting 4-38
  Web page 2-42
templates
  M-book 8-33
temporary directory
  for deleted files 2-55
  terminating a running program 3-5
text
  converting to input cells 8-42
  finding in page in Help browser 4-21
  preferences in MATLAB 2-44
  styles in M-book 8-33
text editors for M-files 6-4
text files
  editing in Editor 6-12
  opening in Editor 6-8
time
  measured for M-files 7-35
time-out message
  while evaluating multiple input cells in an
    M-book 8-44
tmp/MATLAB_Files directory 2-55
Toggle Graph Output for Cell 8-46
toolbars
  desktop 2-32
  shortcuts 2-25
toolbox path cache
  preferences 1-11
tools in desktop
  description 2-2
tooltips 2-32
tooltips for data 6-47
Trash Can 2-55
type ahead feature 3-12
  Command feature 3-12

U
  UNC pathname 7-12
  uncheckout 9-19
  uncomment 6-14
  Undefine Cells 8-46
  undo
    in desktop 2-39
    undo in Editor 6-19
  undoing file check-out
    on PC platforms 9-18
    on UNIX platforms 9-32
  Ungroup Cells 8-47
  Unicode
    preference when saving 2-56
  unregserver startup option 1-6
  updates 2-43

Index-15
updates to products 2-42
uppercase usage in MATLAB 3-8
Use 16-Color Figures 8-38
utilities
  running from MATLAB 3-6

V
values
  examining 6-46
  viewing in Editor 6-31
variables
  clearing 5-7
  displaying values of 5-11
  editing values 5-10
  graphing from the Workspace browser 5-8
  naming 5-19
  saving 5-4
  viewing during execution 6-46
  workspace 5-2
version 2-43
  information for MathWorks products 4-40
  startup option for UNIX 1-8
version control system
  See source control system interface on PC platforms or source control system
  interface on UNIX platforms
viewing desktop tools 2-6
Visible figure property
  embedding graphics in M-book 8-36

W
warning breakpoints 6-60
warning message identifiers 6-62
Web
  accessing from MATLAB 2-42
  site for The MathWorks 2-42
Web browser
  font 2-31
  in MATLAB 2-29
  proxy server 2-31
what 5-32
who 5-4
whos 5-4
width of Command Window 3-28
windows in desktop
  about 2-2
  arrangement 2-5
  closing 2-6
  opening 2-6
  winword.exe 8-40
Word documents
  converting to M-book 8-21
work directory 1-3
working directory 5-30
workspace
  base 5-8
  clearing 5-7
  defined 5-2
  functions 5-8
  initializing in M-book 8-25
  loading 5-6
  M-book contamination 8-22
  opening 5-6
  protecting integrity 8-22
  saving 5-4
  tool 5-2
  viewing 5-3
  viewing during execution 6-46
  Workspace browser
    description 5-2
    preferences 5-9
    wrapping lines 3-28
    wrapping long lines 3-11
X
Xserver option for UNIX 1-8

Y
yellow highlighting in M-file 6-65