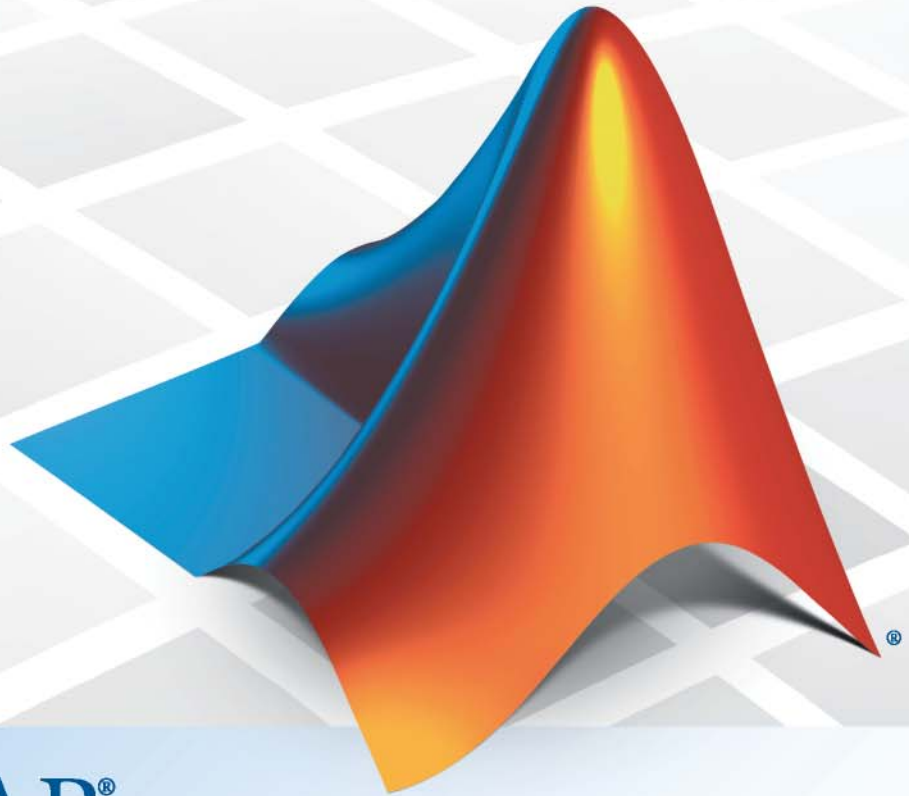


MATLAB® 7

Desktop Tools and Development Environment



MATLAB®

How to Contact MathWorks



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



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



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

MATLAB® Desktop Tools and Development Environment

© COPYRIGHT 1984–2010 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

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

Patents

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

Revision History

June 2004	First printing	New for MATLAB 7.0 (Release 14). Formerly part of Using MATLAB.
October 2004	Online only	Revised for MATLAB 7.0.1 (Release 14SP1)
March 2005	Online only	Revised for MATLAB 7.0.4 (Release 14SP2)
March 2005	Second printing	Revised for MATLAB 7.0.4 (Release 14SP2)
June 2005	Third printing	Minor revision for MATLAB 7.0.4 (Release 14SP2)
September 2005	Online only	Revised for MATLAB 7.1 (Release 14SP3)
March 2006	Online only	Revised for MATLAB 7.2 (Release 2006a)
September 2006	Online only	Revised for MATLAB 7.3 (Release 2006b)
March 2007	Online only	Revised for MATLAB 7.4 (Release 2007a)
September 2007	Online only	Revised for MATLAB 7.5 (Release 2007b)
March 2008	Online only	Revised for MATLAB 7.6 (Release 2008a)
October 2008	Online only	Revised for MATLAB 7.7 (Release 2008b)
March 2009	Online only	Revised for MATLAB 7.8 (Release 2009a)
September 2009	Online only	Revised for MATLAB 7.9 (Release 2009b)
March 2010	Online only	Revised for MATLAB 7.10 (Release 2010a)
March 2010	Online Only	Revised for MATLAB 7.10 (Release 2010a)
September 2010	Online only	Revised for Version 7.11 (R2010b)

Startup and Shutdown

1

Starting the MATLAB Program on Windows	
Platforms	1-1
Associating Files with MATLAB on Windows Platforms ..	1-2
Starting the MATLAB Program on Linux Platforms ...	1-5
Starting the MATLAB Program on Macintosh	
Platforms	1-6
Limitation	1-6
Startup Error Log Reporter	1-7
Startup Folder for the MATLAB Program	1-8
What Is the Startup Folder?	1-8
Startup Folder on Windows Platforms	1-9
Startup Folder on Linux Platforms	1-10
Startup Folder on Macintosh Platforms	1-10
Changing the Startup Folder	1-10
Startup Options	1-14
Specifying MATLAB Startup Options	1-14
Commonly Used Startup Options	1-16
Passing Perl Variables on Startup	1-17
Startup and Calling Java Software from the MATLAB Program	1-18
Toolbox Path Caching in the MATLAB Program	1-19
About Toolbox Path Caching in the MATLAB Program ...	1-19
Using the Cache File Upon Startup	1-19
Updating the Cache and Cache File	1-19
Additional Diagnostics with Toolbox Path Caching	1-22

Quitting the MATLAB Program	1-23
Ways to Quit the MATLAB Program	1-23
Confirm Quitting the MATLAB Program	1-23
Running a Script When Quitting the MATLAB Program ..	1-24
Abnormal Termination	1-24

Desktop

2

Desktop Overview	2-2
About the Desktop	2-2
Summary of Desktop Tools	2-4
Opening and Arranging Desktop Tools	2-5
Opening Desktop Tools	2-5
Navigating Among Desktop Tools and Documents	2-7
Closing Desktop Tools	2-8
Resizing Desktop Tools	2-9
Moving Tools Within the Desktop	2-10
Undocking Tools to Move Them Outside the Desktop	2-13
Moving Undocked Tools Back onto the Desktop	2-14
Grouping Desktop Tools Together	2-14
Maximizing Available Space on the Desktop	2-16
Maximizing Tools Within the Desktop	2-17
Minimizing Tools Within the Desktop	2-17
Opening and Arranging Desktop Documents	2-20
Opening Documents	2-20
Navigating Among Open Documents Using the Document Bar	2-22
Adjusting the Document Bar	2-23
Positioning Documents	2-24
Moving and Resizing Documents	2-34
Closing Documents	2-34
Moving Documents Outside of the Desktop (Undocking) ..	2-35
Docking Documents and Tools	2-36
Grouping Documents in a Tool Outside the Desktop	2-36
Managing Desktop Layouts	2-37

Overview of Desktop Layouts	2-37
Saving a Desktop Layout	2-37
Reusing a Saved or Predefined Desktop Layout	2-38
Renaming a Saved Desktop Layout	2-38
Deleting a Saved Desktop Layout	2-39
Restoring the Default Desktop Layout	2-39
Examples of Desktop Arrangements	2-40
About These Examples	2-41
Tool Outside of Desktop and Other Tools Grouped Inside Desktop Example	2-41
Maximized Tool in Desktop Example	2-43
Minimized Tools in Desktop Example	2-44
Tiled Documents in Desktop Example	2-48
No Empty Document Tiles Example	2-49
Maximized Documents Outside of the Desktop Example ..	2-52
Floating (Cascaded) Figures in Desktop Example	2-53
Undocked Tools and Documents Example	2-55
Running Frequently Used Statement Groups with MATLAB Shortcuts	2-57
What Is a MATLAB Shortcut?	2-57
When to Use MATLAB Shortcuts	2-57
Creating MATLAB Shortcuts — Tutorials	2-58
Running MATLAB Shortcuts	2-61
Editing and Organizing MATLAB Shortcuts	2-62
Customizing MATLAB Toolbar Shortcuts	2-63
Performing Desktop Actions Using the Keyboard	2-66
Keyboard Key Combinations	2-66
Performing Desktop Actions Using Keyboard Shortcuts	2-69
Overview of Keyboard Shortcuts	2-69
Choosing a Set of Keyboard Shortcuts	2-70
Comparing Sets of Keyboard Shortcuts	2-74
Displaying Keyboard Shortcuts	2-75
Customizing Keyboard Shortcuts	2-79
Evaluating and Resolving Keyboard Shortcut Conflicts ...	2-85
Examples of Creating, Modifying, and Deleting Keyboard Shortcuts	2-87
Deleting a Set of Keyboard Shortcuts	2-90

Using Keyboard Shortcuts Settings Files Created on Other Systems	2-91
Keyboard Shortcut Restrictions	2-91
Accessing Tools with the Start Button	2-94
Viewing Products and Tools with the Start Button	2-94
Adding Your Own Toolboxes to the Start Button	2-96
Using Web Browsers in MATLAB	2-101
About Web Browsers in MATLAB	2-101
Displaying Pages in Web Browsers	2-103
Web Preferences	2-104
Other Features for Managing the Desktop	2-108
Using Menus and Context Menus	2-108
Using Toolbar Features	2-110
Viewing Status in the Status Bar	2-111
Sizing, Arranging, and Sorting Columns in Desktop Tools	2-111
Selecting Multiple Items	2-113
Cut, Copy, Paste, and Move	2-114
Printing and Page Setup Options for Desktop Tools	2-115
Accessing MathWorks on the Web	2-119
Managing Your Licenses	2-121
Check for Updates	2-123
Specifying Options for MATLAB Using Preferences ...	2-124
Setting Preferences for MATLAB	2-124
Summary of Preferences	2-125
Where MATLAB Stores Preferences	2-126
Preferences Folder and Files MATLAB Uses When Multiple MATLAB Releases Are Installed	2-127
Setting General Preferences for the MATLAB	
Application	2-129
General Preferences	2-129
MAT-Files Preferences	2-131
Confirmation Dialogs Preferences	2-132
Source Control Preferences	2-136

Java Heap Memory Preferences	2-136
Customizing the Desktop Using Preferences	2-138
Setting Keyboard Preferences for Desktop Tools	2-138
Setting Fonts Preferences for Desktop Tools	2-141
Setting Colors Preferences	2-150
Setting Color Preferences for Programming Tools	2-154
Setting Toolbars Preferences for Desktop Tools	2-156
Accessibility	2-159
Software Accessibility Support	2-159
Documentation Accessibility Support	2-160
Assistive Technologies	2-161
Installation Notes for Accessibility Support	2-162
Troubleshooting	2-165
Macintosh Platform — Differences	2-172
GUI Conventions in the Documentation and Macintosh Platforms	2-172
Pointer Device Instructions and Macintosh Platforms	2-172
Using File Browser GUIs on Macintosh Platforms to Navigate Within the MATLAB Root Folder	2-172

Running Functions — Command Window and History

3

Using the Command Window	3-2
About the Command Window	3-2
Opening the Command Window	3-2
Using the Command Window Prompt	3-3
Changing How the Command Window Looks	3-4
Running Functions and Programs, and Entering Variables	3-5
Running Statements at the Command-Line Prompt	3-5
Stopping Execution	3-8
Running External Programs	3-8

Evaluating or Opening a Selection	3-11
Displaying Hyperlinks in the Command Window	3-12
Entering Statements in the Command Window	3-17
Case and Space Sensitivity	3-17
Cut, Copy, Paste, and Undo Features	3-18
Entering Multiple Lines Without Running Them	3-18
Entering Multiple Functions in a Line	3-20
Entering Multiple-Line (Long) Statements Using Line Continuation	3-20
Recalling Previous Lines in the Command Window	3-21
Navigating Above the Command Line	3-22
See Also	3-22
Assistance While Entering Statements	3-23
Highlighting Syntax to Help Ensure Correct Entries	3-23
Matching Delimiters (Parentheses)	3-24
Completing Statements in the Command Window — Tab Completion	3-24
Viewing Function Syntax Hints While Entering a Statement	3-33
Getting Help for a Function Shown in the Command Window or Editor	3-38
Finding Functions Using the Function Browser	3-40
See Also	3-46
Controlling Output in the Command Window	3-47
Echoing Execution	3-47
Suppressing Output	3-47
Paging of Output in the Command Window	3-47
Formatting and Spacing Numeric Output	3-48
Number of Characters in Command Window Display	3-49
Clearing the Command Window	3-50
Printing Command Window Contents	3-50
Keeping a Session Log	3-51
Finding Text in the Command Window	3-52
Introduction	3-52
Finding Text Currently Displayed in the Command Window	3-52
Increasing the Amount of Information Available for Searching in the Command Window	3-53

Using Incremental Search in the Command Window	3-53
Preferences for the Command Window	3-60
Text, Display, Accessibility, and Tab Size Preferences	3-60
Additional Settings That Affect the Command Window	3-64
Using the Command History Window	3-66
Overview of the Command History Window	3-66
Viewing Statements in the Command History Window	3-68
Performing Actions on Statements in the Command History Window	3-68
Searching in the Command History Window	3-70
Printing the Command History Window	3-76
Deleting Entries from the Command History Window	3-76
Preferences for Command History	3-78
Overview of Command History Preferences	3-78
Settings	3-78
Saving	3-79
See Also	3-80

Getting Help and Product Information

4

Overview of Help	4-2
Using the Help Browser	4-4
About the Help Browser	4-4
Getting Help for Functions and Blocks	4-8
Accessing a Specific Page	4-12
See Also	4-13
Searching the Documentation	4-14
Performing a Simple Search	4-14
Improving Search Results	4-14
Advanced Search Techniques	4-20
Searching Within a Page	4-23

Learning from Demos	4-25
About Demos	4-25
Types of Demos	4-25
Accessing Demos	4-26
Running Demos	4-26
Configuring the Help Browser	4-28
Adjusting the Help Browser Layout	4-28
Specifying Which Documentation to Display	4-28
Accessing English Documentation on Japanese Systems ..	4-29
Customizing Help Browser Fonts and Colors	4-30
Preferences for Configuring Help Windows, Search History, and PDF Readers	4-32
Using Printed Documentation	4-35
Printing from the Help Browser	4-35
Accessing and Printing PDF Documentation	4-35
Obtaining Printed Manuals	4-36
Additional Help and Learning Resources	4-37
Obtaining Information About your Installation	4-37
Obtaining Technical Support	4-38
Product Documentation at the MathWorks Web Site	4-39
Newsgroup for MathWorks Products	4-40
File Exchange — Files Created By Other Users	4-40
Blogs for MathWorks Products	4-40
Newsletters for MathWorks Products	4-40
Seminars and Webinars for MathWorks Products	4-40
Training for MathWorks Products	4-41

Customizing Help and Demos

5

Getting Help for Files Created by Others	5-2
About Help for Files Created by Others	5-2
Getting Command-Line Help for Externally Supplied Program Files	5-2
Viewing a Help Summary for Externally Supplied Files ..	5-3
Accessing Help for Externally Supplied Class Files	5-3

Accessing Externally Supplied Documentation in the Help Browser	5-6
Accessing Externally Supplied Demos in the Help Browser	5-7
Providing Your Own Help and Demos	5-8
About Providing Help and Demos	5-8
Adding Help for Your Program Files	5-9
Adding HTML Help Files to the Help Browser	5-17
Adding Demos to the Help Browser	5-52
About Creating Demos	5-52
Providing Demos to Others	5-60
Addressing Validation Errors for info.xml Files	5-61
About XML File Validation	5-61
Entities Missing or Out of Order in info.xml	5-61
Unrelated info.xml File	5-62
Invalid Constructs in info.xml File	5-62
Outdated info.xml File for a MathWorks Product	5-62

Workspace Browser and Variable Editor

6

MATLAB Workspace	6-2
About the Workspace	6-2
Opening the Workspace Browser	6-2
Viewing and Editing Values in the Current Workspace ...	6-4
Saving the Current Workspace	6-5
Viewing and Loading a Saved Workspace and Importing Data	6-7
Changing and Copying Variable Names	6-8
Deleting Workspace Variables	6-9
Viewing Base and Function Workspaces Using the Stack	6-9
Creating Plots from the Workspace Browser	6-10
Opening Variables and Objects for Viewing and Editing ..	6-21
Setting Workspace Browser Preferences	6-21

Viewing and Editing Workspace Variables with the Variable Editor	6-24
About the Variable Editor	6-24
Opening the Variable Editor	6-24
Working with Different Types of Data in the Variable Editor	6-27
Navigating and Editing Shortcut Keys for the Variable Editor	6-34
Changing Size, Content, and Format of Variables in the Variable Editor	6-35
Cut, Copy, Paste, and Clear Contents in the Variable Editor	6-36
Other Variable Editor Operations	6-40
Creating Graphs and Variables, and Data Brushing in the Variable Editor	6-41
Preferences for the Variable Editor	6-46

Managing Files in MATLAB

7

Introduction to Managing Files in MATLAB	7-2
Ways to Manage MATLAB Files	7-2
Tools for Managing Files	7-2
Understanding File Locations in MATLAB	7-4
Important MATLAB Folders	7-4
Path Names in MATLAB	7-7
Working with Files and Folders	7-12
Viewing Folder Contents	7-12
Using the Current Folder Browser	7-18
Finding Files and Folders	7-27
Finding Files and Folders by Name in the Current Folder	7-27
Simple Search for File and Folder Names in the Current Folder Browser	7-27
Advanced Search for Files — Find Files Tool	7-30

Locating a File or Folder in the Operating System	
Browser	7-34
Finding Files and Folders Using Functions	7-35
Additional Ways to Find Files	7-35
Creating, Opening, Changing, and Deleting Files and Folders	7-36
Creating New Files and Folders	7-36
Copying, Renaming, and Deleting Files and Folders	7-42
Opening and Running Files	7-45
Comparing Files and Folders	7-50
Comparing Files and Folders	7-50
Comparing Text Files	7-52
Comparing Files with Autosave Version or Version on Disk	7-56
Comparing MAT-Files	7-57
Comparing Binary Files	7-59
Comparing Folders and ZIP Files	7-60
Using Features of the Comparison Tool	7-63
Function Alternative for Comparing Files and Folders ...	7-65
Making Files and Folders Accessible to MATLAB	7-66
Files and Folders That MATLAB Can Access	7-66
How to Make Files Accessible	7-66
Determining if MATLAB Can Access a File	7-68
Ensuring MATLAB Uses the File You Want	7-70
Using the MATLAB Search Path	7-72
What Is the Search Path?	7-72
Viewing Files and Folders on the Search Path	7-74
Changing the Search Path	7-75
Using the Search Path with Different MATLAB Installations	7-80
Recovering from Problems with the Search Path	7-81
Handling Errors and Unexpected Behavior When Updating Folders	7-83
Related Topics for Managing Files	7-84

File Exchange — Finding and Getting Files Created by Other Users

8

Before Using File Exchange	8-2
What Is File Exchange?	8-2
What You Need to Use File Exchange	8-2
Ways to Access the File Exchange Repository	8-3
How To Use the File Exchange Desktop Tool	8-5
Steps for Using File Exchange	8-5
Example — Finding and Downloading a File in File Exchange	8-6
Finding Files in File Exchange — Searching and Using Tags	8-13
About Finding Files in File Exchange	8-13
Using Search to Find Files in File Exchange	8-13
Finding Files by Product, Author, and Other Attributes in File Exchange	8-14
Using Tags to Find Files in File Exchange	8-14
Clearing Your Criteria	8-26
Getting Better Results Using Search and Tags	8-26
Viewing and Sorting the List of Files in File Exchange	8-28
Viewing the List of Files in File Exchange	8-28
Sorting the List of Files in File Exchange	8-29
Viewing Details About a File	8-30
Viewing the File Details Page	8-30
Viewing the Contents of a File	8-30
Downloading Files from the File Exchange Repository	8-32
About Downloading Files	8-32
Downloading from the List of Files	8-32
Downloading from the File Details Page to a Location You Choose	8-33
The Default Folder for Downloaded Files	8-33

Which Location Should You Choose When Downloading Files?	8-33
Downloading a Submission that Consists of Multiple Files	8-34
Viewing and Locating Files You Downloaded	8-34
Best Practices for Using Files Provided by Other Users	8-37
Ensure MATLAB Can Access the File	8-37
Consult the File Details Page	8-37
Look for Updates to the File	8-37
Read the File	8-38
Ask Questions	8-38
Contributing to the File Exchange Repository	8-39
How You Can Contribute to the Repository	8-39
Adding Tags to a File	8-39
Removing Tags from a File	8-40
Rating a File	8-40
Providing Comments About a File	8-41
Submitting Your Files to the Repository	8-41
Frequently Asked Questions About File Exchange	8-42
What Is File Exchange?	8-42
How Do I Use File Exchange?	8-42
How Does the File Exchange Desktop Tool Relate to File Exchange on the Web Site?	8-43
Why Do I See Only 50 Files and How Can I See More? ...	8-43
What Are Tags and How Do I Use Them?	8-44
What Are the Tags Above the List of Files?	8-44
How Can I See Other Tags?	8-44
Why Are the Tags Changing?	8-45
Is Search Looking Inside Files?	8-45
How Can I Start Over When Looking for Files?	8-45
How Can I Choose Where to Download a File To?	8-46
How Do I Contribute My Files to the Repository?	8-46

MATLAB Code Files	9-2
What Are MATLAB Code Files?	9-2
Creating Files from the Command Window and Command History	9-2
Use Existing MATLAB Code and Examples	9-2
 Ways to Edit, Evaluate, and Debug Code	 9-4
 Starting, Creating Files, and Closing the Editor	 9-6
Starting the Editor	9-6
Creating New Files in the Editor	9-7
Opening Existing Files Using the Editor	9-9
Arranging Editor Documents	9-11
Closing the Editor	9-11
 Customizing the Editor by Setting Preferences	 9-13
Overview of Setting Editor/Debugger Preferences	9-13
Setting General Preferences for the Editor/Debugger	9-15
Setting Display Preferences	9-16
Setting Tab and Indent Preferences	9-19
Setting Language Preferences	9-20
Setting MATLAB Language Preferences	9-21
Setting TLC Language Preferences	9-28
Setting VHDL Language Preferences	9-28
Setting Verilog Language Preferences	9-29
Setting C/C++ Language Preferences	9-30
Setting Java Language Preferences	9-32
Setting XML/HTML Language Preferences	9-33
Setting Code Folding Preferences	9-34
Setting Autosave Preferences	9-35
Additional Information About Editor/Debugger Preferences	9-37
 Entering Statements in the Editor	 9-38
Using Command Window Features in the Editor	9-38
Entering Text in Insert or Overwrite Mode	9-39
Changing the Case of Selected Text	9-39
Undoing and Redoing Editor Actions	9-40

Adding Comments	9-40
Completing Statements in the Editor — Tab Completion ..	9-46
Making MATLAB Code Files More Readable	9-53
Syntax Highlighting	9-53
Indenting	9-53
Function Indenting	9-55
Line and Column Numbers	9-55
Highlight Current Line	9-55
Right-Side Text Limit	9-56
Class, Function, or Subfunction	9-57
Code Folding — Expanding and Collapsing File Constructs	9-57
Displaying Two Parts of a File Simultaneously	9-67
Navigating an Open File in the Editor	9-71
Navigating to a Specific Location	9-71
Using Bookmarks	9-75
Navigating Backward and Forward in Files	9-75
Opening a File or Variable from Within a File	9-76
Finding Text in Files	9-78
Finding Any Text in the Current File	9-78
Finding and Replacing Functions or Variables in the Current File	9-78
Finding and Replacing Any Text	9-80
Finding Text in Multiple File Names or Files	9-81
Function Alternative for Finding Text	9-82
Performing an Incremental Search in the Editor	9-82
Saving, Printing, and Closing Files in the Editor	9-83
Saving Files	9-83
Printing Files	9-85
Closing Files	9-86
Running MATLAB Files in the Editor	9-87
Running Files with No Input Arguments in the Editor ...	9-87
Using Run Configurations to Run Files with Input Arguments in the Editor	9-88
Create and Use a Run Configuration	9-88
Create and Execute Multiple Run Configurations for a File	9-93

About the run_configurations.m File	9-96
Find Configurations	9-96
Remove Configurations	9-98
Reassociate and Rename Configurations	9-99
Other Ways to Run Files from the Editor	9-103
Finding Errors, Debugging, and Correcting MATLAB	
Files	9-104
Preventing and Identifying Coding Problems	9-107
Ways to Prevent and Check for Coding Problems	9-107
Code Analysis Options	9-107
Determining Scope and Usage of Functions and	
Variables	9-135
Debugging Process and Features	9-141
Ways to Debug MATLAB Files	9-141
Preparing for Debugging	9-141
Setting Breakpoints	9-144
Running a File with Breakpoints	9-148
Stepping Through a File	9-150
Examining Values	9-152
Correcting Problems and Ending Debugging	9-158
Using Conditional Breakpoints	9-166
Breakpoints in Anonymous Functions	9-168
Breakpoints in Methods That Overload Functions	9-169
Error Breakpoints	9-170
Evaluating Subsections of Files Using Code Cells	9-175
What Are Code Cells?	9-175
Scenarios for Evaluating Sections of Code	9-176
Process for Evaluating Sections of Files	9-177
Defining Code Cells	9-178
Understanding Nested Code Cells	9-185
Navigating Among Code Cells in a File	9-193
Evaluating Code Cells	9-194
Debugging Functions	9-202

Tuning and Managing MATLAB Code Files

10

Using MATLAB Reports	10-2
Refining and Improving Files Using Reports	10-2
Identifying Files with Reminder Annotations	10-4
Generating a Summary View of the Help Components in Functions and Scripts	10-8
Displaying and Updating a Report on the Contents of a Folder	10-11
Displaying Dependencies Among MATLAB Code Files ...	10-15
Identifying How Much of a File Ran When Profiled	10-20
Using the Code Analyzer Report	10-22
Running the Code Analyzer Report	10-22
Changing Code Based on Messages	10-24
Other Ways to Access Messages	10-25
Profiling for Improving Performance	10-27
What Is Profiling?	10-27
Profiling Process and Guidelines	10-28
Using the Profiler	10-30
Profile Summary Report	10-36
Profile Detail Report	10-38
The profile Function	10-46

Publishing MATLAB Code

11

Overview of Publishing MATLAB Code	11-2
What Is Meant by Publishing MATLAB Code?	11-2
Using Code Cells	11-2
Process for Publishing MATLAB Code	11-3
Example of Published MATLAB Code	11-4
Adding the Markup for the Example	11-10
Marking Up MATLAB Comments for Publishing	11-17

Overview of Marking Up MATLAB Comments for Publishing	11-18
Creating Document Titles and Introductory Text for Publishing an Existing File	11-19
Specifying Preformatted Text in MATLAB Files for Publishing	11-25
Specifying Bulleted or Numbered Lists in MATLAB Files for Publishing	11-27
Specifying Graphics in MATLAB Files for Publishing	11-30
Using HTML Markup Tags in MATLAB Files for Publishing	11-34
Using LaTeX Markup for Publishing	11-36
Including Inline LaTeX Math Symbols in MATLAB Files for Publishing	11-39
Including Blocks of LaTeX Math Symbols in MATLAB Files for Publishing	11-40
Forcing a Snapshot of Output in MATLAB Files for Publishing	11-42
Including Bold, Italic, and Monospaced Text in MATLAB Files for Publishing	11-43
Including Trademarks in MATLAB Files for Publishing ..	11-45
Including Hyperlinks in MATLAB Files for Publishing ...	11-46
Cleaning Up Text Markup Before Publishing MATLAB Files	11-54
Summary of Markup for Publishing MATLAB Files	11-57
Marking Up MATLAB Code for Publishing	11-60
Overview of Marking Up MATLAB Code for Publishing ..	11-60
Specifying the Display of Code Output	11-60
Example of Marking Up Code	11-60
Specifying Output Preferences for Publishing	11-64
About Publishing Configurations	11-64
Creating a Publish Configuration for a MATLAB File	11-66
Running an Existing Publish Configuration	11-92
Creating Multiple Publish Configurations for a File	11-93
About the publish_configurations.m File	11-104
Finding Publish Configurations	11-105
Removing Publish Configurations	11-105
Reassociating and Renaming Publish Configurations	11-105
Summary of Options for Presenting Your Code to Others	11-106

Creating a MATLAB Notebook to Publish to Microsoft Word

12

About MATLAB Notebooks	12-2
Contents of MATLAB Notebooks	12-2
Creating or Opening a MATLAB Notebook	12-2
Entering Commands in a MATLAB Notebook	12-9
Protecting the Integrity of Your Workspace in MATLAB Notebooks	12-10
Ensuring Data Consistency in MATLAB Notebooks	12-10
Debugging and MATLAB Notebooks	12-11
Defining MATLAB Commands as Input Cells for a MATLAB Notebook	12-12
Defining Input Cells for a MATLAB Notebook	12-12
Defining Cell Groups for a MATLAB Notebook	12-13
Defining Autoinit Input Cells for a MATLAB Notebook ...	12-14
Defining Calc Zones for a MATLAB Notebook	12-14
Converting an Input Cell to Text in a MATLAB Notebook	12-15
Evaluating MATLAB Commands in a MATLAB Notebook	12-17
Evaluating Input Commands	12-17
Evaluating Cell Groups	12-18
Evaluating a Range of Input Cells	12-20
Evaluating a Calc Zone	12-20
Evaluating an Entire MATLAB Notebook	12-20
Using a Loop to Evaluate Input Cells Repeatedly	12-21
Converting Output Cells to Text	12-22
Deleting Output Cells	12-22
Printing and Formatting a MATLAB Notebook	12-23
Printing a MATLAB Notebook	12-23
Modifying Styles in the MATLAB Notebook Template	12-23
Choosing Loose or Compact Format	12-24
Controlling Numeric Output Format	12-25
Controlling Graphic Output	12-25
Configuring MATLAB notebook	12-28

Notebook Feature Reference	12-30
Bring MATLAB to Front	12-30
Define Autoinit Cell	12-31
Define Calc Zone	12-31
Define Input Cell	12-32
Evaluate Calc Zone	12-32
Evaluate Cell	12-33
Evaluate Loop	12-34
Evaluate M-Book	12-34
Group Cells	12-34
Hide Cell Markers	12-35
Notebook Options	12-35
Purge Selected Output Cells	12-35
Toggle Graph Output for Cell	12-36
Undefine Cells	12-36
Ungroup Cells	12-37

Source Control Interface

13

Source Control Interface on Microsoft Windows	13-2
 Setting Up the Source Control Interface on Microsoft	
Windows	13-3
Create Projects in Source Control System	13-3
Specify Source Control System with MATLAB Software ..	13-5
Register Source Control Project with MATLAB Software ..	13-7
Add Files to Source Control	13-10
 Checking Files Into and Out of Source Control from the	
MATLAB Desktop on Microsoft Windows	13-11
Check Files Into Source Control	13-11
Check Files Out of Source Control	13-12
Undoing the Checkout	13-13
 Additional Source Control Actions on Microsoft	
Windows	13-14
Getting the Latest Version of Files for Viewing or	
Compiling	13-14

Removing Files from the Source Control System	13-15
Showing File History	13-16
Comparing the Working Copy of a File to the Latest Version in Source Control	13-18
Viewing Source Control Properties of a File	13-20
Starting the Source Control System	13-21
Performing Source Control Actions from the Editor, Simulink, or Stateflow File Menu on Microsoft Windows	13-23
Troubleshooting Source Control Problems on Microsoft Windows	13-24
Source Control Error: Provider Not Present or Not Installed Properly	13-24
Restriction Against @ Character	13-25
Add to Source Control Is the Only Action Available	13-25
More Solutions for Source Control Problems	13-25
Source Control Interface on UNIX Platforms	13-26
Specifying the Source Control System on UNIX Platforms	13-27
MATLAB Desktop Alternative	13-27
Function Alternative	13-28
Setting a View and Checking Out a Folder with ClearCase Software on UNIX Platforms	13-29
Checking Files Into the Source Control System on UNIX Platforms	13-30
Checking In One or More Files Using the Current Folder Browser	13-30
Checking In One File Using the Editor, or the Simulink or Stateflow Products	13-31
Function Alternative	13-32
Checking Files Out of the Source Control System on UNIX	13-33
Checking Out One or More Files Using the Current Folder Browser	13-33

Checking Out a Single File Using the Editor, or the Simulink or Stateflow Products	13-34
Function Alternative	13-34
Undoing the Checkout on UNIX Platforms	13-36
Impact of Undoing a File Checkout	13-36
Undoing the Checkout for One or More Files Using the Current Folder Browser	13-36
Undoing the Checkout for a Single File Using the Editor, or the Simulink or Stateflow Products	13-36
Function Alternative	13-37

Internationalization

14

How the MATLAB Process Uses Locale Settings	14-2
Windows Platform-Specific Behavior	14-3
Macintosh Platform-Specific Behavior	14-3
Setting the Locale	14-4
Setting Locale on Windows Platforms	14-4
Setting Locale on Linux Platforms	14-6
Setting Locale on Macintosh Platforms	14-7
Troubleshooting I18n Messages and Settings	14-9
Asian Characters Incorrectly Displayed on Linux Systems	14-9
Characters Incorrectly Displayed on Windows Systems ..	14-10
datenum Might Not Return Correct Value	14-10
Numbers Display Period for Decimal Point	14-10
MATLAB Displays Messages in English	14-11
File or Folder Names Incorrectly Displayed	14-11

Index

Startup and Shutdown


The way you start the MATLAB® program depends on which platform you use. The following topics describe how to startup and shutdown MATLAB software on all supported platforms, with information about how you can customize startup and shutdown.

- “Starting the MATLAB Program on Windows Platforms” on page 1-1
- “Starting the MATLAB Program on Linux Platforms” on page 1-5
- “Starting the MATLAB Program on Macintosh Platforms” on page 1-6
- “Startup Error Log Reporter” on page 1-7
- “Startup Folder for the MATLAB Program” on page 1-8
- “Startup Options” on page 1-14
- “Toolbox Path Caching in the MATLAB Program” on page 1-19
- “Quitting the MATLAB Program” on page 1-23

Starting the MATLAB Program on Windows Platforms

There are several ways to start the MATLAB program on a Microsoft® Windows® platform:

- Select **Start > Programs > MATLAB > R2010b > MATLAB R2010b**.
- Double-click a file with certain file extensions in the Windows Explorer tool. The installer sets up associations between certain file types and MathWorks products during installation. For example, double-clicking a file with a `.m` extension in the Windows Explorer tool starts MATLAB and opens the file in the MATLAB Editor. For more information, see “Associating Files with MATLAB on Windows Platforms” on page 1-2.

- Open the DOS window, `cd` to the folder in which you want to start MATLAB and type `matlab` at the DOS prompt.
- If you chose to have the installer create a shortcut, double-click the MATLAB shortcut  on your Windows desktop.

After starting MATLAB, the desktop opens. Desktop components that were open when you last shut down MATLAB will be opened on startup. For more information, see Chapter 2, “Desktop”. You can specify other startup options, such the current folder upon startup—for more information, see “Startup Folder for the MATLAB Program” on page 1-8 and “Startup Options” on page 1-14.

If you have trouble starting MATLAB, see troubleshooting information in the “Troubleshooting”.

Associating Files with MATLAB on Windows Platforms

When you install MATLAB software on Windows platforms, the installer sets up associations between certain file types and MathWorks products. When you double-click a particular file type, identified by its file extension, Windows starts MATLAB and opens the file in the appropriate tool. The following table lists some of the file extensions the installer associates with MathWorks products and the behavior that results from this association. To learn how to change this behavior, see “Managing File Associations for MATLAB on Windows Systems” on page 1-3.

File Extension and Resulting Action

File Extension	Result
<code>.fig</code>	Opens file in figure window
<code>.m</code>	Opens file in Editor
<code>.mat</code>	Opens Import Wizard to load the data into the MATLAB workspace.
<code>.mdl</code>	Opens file in a Simulink [®] model window

File Extension and Resulting Action (Continued)

File Extension	Result
.mex ¹	Displays icon for MATLAB in Windows Explorer tool
.p	Displays icon for MATLAB in Windows Explorer tool

File associations for the Windows Explorer tool do not affect what happens when you open one of these file types from *within* MATLAB. MATLAB acts on the file using the MATLAB tool associated with that file type. For example, even if your system associates .mat files with the Access™ application, when you open a MAT-file from within MATLAB, it opens the Import Wizard to load the data.

Managing File Associations for MATLAB on Windows Systems

You can associate any file types with MATLAB from the Windows Explorer tool. For example, you can associate the .xml extension with MATLAB so that when you double-click an XML file, it opens in the MATLAB Editor.

In addition, you can resolve conflicts with other applications that want to associate with the same file extension. For example, the Microsoft® Access™ application might associate files having a .mat extension. Then, double-clicking a MAT-file opens the Access application rather than MATLAB.

To manage file associations with MATLAB, follow this procedure:

Note These instructions might not exactly apply to your version of the Windows operating system. If you encounter differences or problems, see your Windows documentation.

- 1** Open a Windows Explorer window.
- 2** Select **Tools > Folder Options**.

1. MEX-file extensions are platform specific. See “Using Binary MEX-Files”.

- 3 Select the **File Types** tab.
- 4 Browse the list of file types and select the file extension you want to associate with MATLAB. If you do not see the extension in the list, click **New** to add it. For example, select the **MAT** file extension.
- 5 In the **Details for 'MAT' extension** area of the dialog box, click **Change**.
- 6 Select MATLAB from the list of programs and click **OK**.

If the list does not include MATLAB, click **Browse** to select the MATLAB executable file and then click **Open**. The MATLAB executable file (`matlab.exe`) is located in the `bin` folder of your MATLAB installation folder, for example, `C:\Program Files\MATLAB\R2010b\bin\win32`.

If you want to specify what will be when you double-click on a file with a particular extension, click the **Advanced** button.

- 7 Click **Close** to close the Folder Options dialog box.

After associating a file type with MATLAB, you can open other applications that have the same extension via the context menu. For example, if you want to open a MAT-file with the Access application, right-click `myfile.mat`, and from the context menu, select **Open With**. The Access application should be one of the options.

Utilities to Set Up File Associations on Windows Platforms

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

- Run utility to associate files with `.fig` extension with MATLAB
- Run utility to associate files with `.m` extension with MATLAB
- Run utility to associate files with `.mat` extension with MATLAB
- Run utility to associate files with `.mdl` extension with MATLAB
- Run utility to associate MATLAB with MEX-files

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

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

Starting the MATLAB Program on Linux Platforms

To start the MATLAB program on Linux® platforms, type `matlab` at the operating system prompt.

If you did not set up symbolic links in the installation procedure, you must enter the full pathname to start MATLAB, `matlabroot/bin/matlab`, where `matlabroot` is the name of the folder in which you installed MATLAB.

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

If you have trouble starting MATLAB, see troubleshooting information in the “Troubleshooting”.

You can specify the current folder upon startup as well as other options—for more information, see “Startup Folder for the MATLAB Program” on page 1-8 and “Startup Options” on page 1-14.

Starting the MATLAB Program on Macintosh Platforms

There are several ways to start the MATLAB program on a Microsoft Windows platform:

- Double-click the MATLAB icon in the Applications folder.
- Open a Terminal window, navigate to your MATLAB installation folder, and type `matlab` at the operating system prompt.

```
/Applications/MATLAB_R2010b.app/bin/matlab
```

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

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

If you have trouble starting MATLAB, see troubleshooting information in the “Troubleshooting”.

You can specify the current folder upon startup as well as other options—for more information, see “Startup Folder for the MATLAB Program” on page 1-8 and “Startup Options” on page 1-14.

Limitation

On Macintosh® platforms, if you run MATLAB remotely, for example using `rlogin`, you must run with `nodisplay`, `noawt`, and `nojvm` startup options—for more information, see “Startup Options” on page 1-14.

Startup Error Log Reporter

Upon startup, if the MATLAB program detects an error log generated by a serious problem encountered during the *previous* session, an Error Log Reporter prompts you to e-mail the log to MathWorks for analysis. Click **Send Report** to e-mail the log, or click **Help** for more information. After sending the log, a confirmation message appears in the Command Window. For more information, see “Abnormal Termination” on page 1-24.

Startup Folder for the MATLAB Program

In this section...

“What Is the Startup Folder?” on page 1-8

“Startup Folder on Windows Platforms” on page 1-9

“Startup Folder on Linux Platforms” on page 1-10

“Startup Folder on Macintosh Platforms” on page 1-10

“Changing the Startup Folder” on page 1-10

What Is the Startup Folder?

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

Accepting the default value for *userpath* and using it as the startup folder offers these benefits:

- You can store the MATLAB files you work with in one, appropriately-named location, such as Documents/MATLAB.
- Your MATLAB files are readily available upon startup, because the current folder is always the same, for example, Documents/MATLAB.
- You can always run your files because MATLAB automatically adds the *userpath* folder to the top of the search path upon startup.
- The first time you run a new version of MATLAB, MATLAB automatically creates the *userpath* folder if it does not exist.
- When you upgrade to a newer version of MATLAB, MATLAB automatically continues to use the same MATLAB folder and your existing files, with all of its other benefits.

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

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

There are other ways to change the startup folder as well as the folders on your search path. For more information, see “Changing the Startup Folder” on page 1-10 and “Viewing Files and Folders on the Search Path” on page 7-74.

Startup Folder on Windows Platforms

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

How Started	Startup Folder
Double-click the MATLAB shortcut on your Windows desktop	The startup folder is set to the <code>userpath</code> value, whose default value is My Documents\MATLAB, or Documents\MATLAB on Windows Vista platforms. The <code>userpath</code> folder is automatically added to the search path. If there is a value specified in the Start in field of the Properties dialog box for the MATLAB program, that value is the startup folder, although the <code>userpath</code> is added to the search path. If MATLAB does not find a valid <code>userpath</code> value upon startup, and the Start in field is empty, the startup folder is the Windows desktop.
Double-click a file type associated with MATLAB	The folder in which the file resides is the startup folder. The <code>userpath</code> folder is automatically added to the search path.

How Started	Startup Folder
In a DOS window	The folder in which you ran the <code>matlab</code> command is the startup folder. The <code>userpath</code> folder is automatically added to the search path.

Startup Folder on Linux Platforms

On Linux platforms, the default startup folder is the folder from which you started MATLAB.

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

Startup Folder on Macintosh Platforms

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

When you start MATLAB in a shell, the default startup folder is the folder from which you started MATLAB.

Changing the Startup Folder

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

- “Changing the Startup Folder Via the `userpath` Function” on page 1-11
- “Changing the Startup Folder Using the Shortcut — Windows Platforms Only” on page 1-11
- “Changing the Startup Folder Using the `startup.m` File” on page 1-13

Changing the Startup Folder Via the userpath Function

Use the userpath function to change the startup folder as well as to add the startup folder to the search path upon startup. For more information, see the userpath reference page and “Startup Folder for the MATLAB Program” on page 1-8.

Changing the Startup Folder Using the Shortcut – Windows Platforms Only

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

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

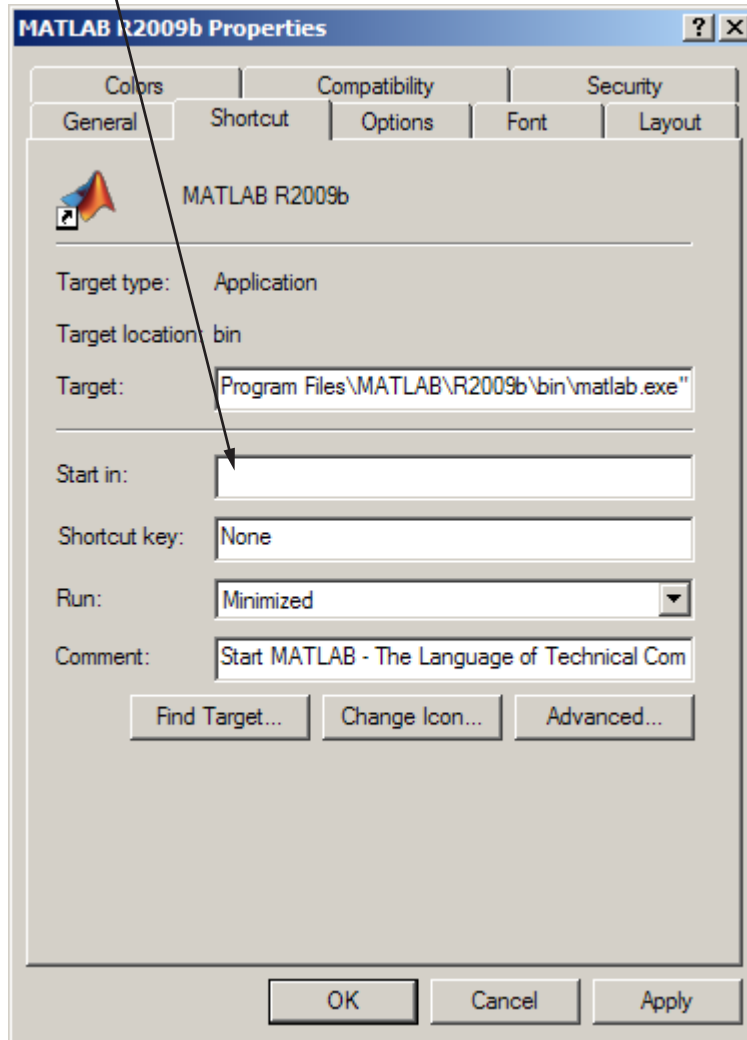
The Properties dialog box for MATLAB opens to the **Shortcut** pane.

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

By default, the startup folder is My Documents\MATLAB or Documents\MATLAB on Windows Vista platforms; for more information, see “Startup Folder on Windows Platforms” on page 1-9.

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

Enter full path to MATLAB startup folder.



The next time you start MATLAB using that shortcut icon, the current folder will be the one you specified in step 2.

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

Changing the Startup Folder Using the `startup.m` File

Use the `startup.m` file to specify the startup folder as well as other startup options—for details, see “Specifying Startup Options in the MATLABStartup File” on page 1-15.

Startup Options

In this section...

“Specifying MATLAB Startup Options” on page 1-14

“Commonly Used Startup Options” on page 1-16

“Passing Perl Variables on Startup” on page 1-17

“Startup and Calling Java Software from the MATLAB Program” on page 1-18

Specifying MATLAB Startup Options

You can specify startup options (also called command flags or command line switches) that instruct the MATLAB program to perform certain operations when you start it. On all platforms, you specify the options as arguments to the `matlab` command when you start is at the operating system prompt. For example, the following starts MATLAB and suppresses the display of the splash screen.

```
matlab -nosplash
```

On Windows platforms, you can precede a startup option with either a hyphen (-) or a slash (/). For example, `-nosplash` and `/nosplash` are equivalent.


On all platforms, you can also specify startup options using a MATLAB startup file—see “Specifying Startup Options in the MATLABStartup File” on page 1-15

On Windows platforms, you can specify startup options in the MATLAB shortcut—see “Including Startup Options in a Shortcut on Windows Systems” on page 1-14.

Including Startup Options in a Shortcut on Windows Systems

You can add selected startup options (also called command flags or switches for the command line) to the target path for your shortcut in the Windows environment for MATLAB. For more information about the options, see “Commonly Used Startup Options” on page 1-16.

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

- 1** Right-click the shortcut icon for MATLAB  and select **Properties** from the context menu. The Properties dialog box for MATLAB opens to the **Shortcut** pane.
- 2** In the **Target** field, after the target path for `matlab.exe`, add the startup option, and click **OK**. For example, adding `-r "filename"` runs the MATLAB code file `filename` after startup.

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

```
C:\Program Files\MATLAB\R2010b\bin\matlab.exe -r "results"
```

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

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

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

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

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

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

Specifying Startup Options in the MATLABStartup File

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

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

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

Place the `startup.m` file in the default or current startup folder, which is where MATLAB first looks for it. For more information, see “Startup Folder for the MATLAB Program” on page 1-8.

Commonly Used Startup Options

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

Platform	Option	Description
All	<code>-c licensefile</code>	Set <code>LM_LICENSE_FILE</code> to <code>licensefile</code> . It can have the form <code>port@host</code> .
All	<code>-h</code> or <code>-help</code>	Display startup options (without starting MATLAB).
All	<code>-logfile</code> "logfilename"	Automatically write output from MATLAB to the specified log file.
Windows platforms	<code>-minimize</code>	Start MATLAB with the desktop minimized. Any desktop tools or documents that were undocked when MATLAB was last closed will not be minimized upon startup.

Platform	Option	Description
UNIX platforms	-nojvm	Start MATLAB without loading the Sun Microsystems JVM™ software. This minimizes memory usage and improves initial startup speed, but restricts functionality. With <code>nojvm</code> , you cannot use the desktop, figures, or any tools that require Java software. For example, you cannot set preferences if you start MATLAB with the <code>-nojvm</code> option. However, you can start MATLAB once <i>without</i> the <code>-nojvm</code> option, set the preference, and quit MATLAB. MATLAB remembers that preference when you start it again, even if you use the <code>-nojvm</code> option.
All	-nosplash	Start MATLAB without displaying its splash screen.
All	-r "statement"	Automatically run the specified statement immediately after MATLAB starts. This is sometimes referred to as calling MATLAB in batch mode. Files you run must be in the startup folder for MATLAB or on the search path. Do not include pathnames or file extensions. Enclose the statement in double quotation marks (" <i>statement</i> "). Use semicolons or commas to separate multiple statements
All	-singleCompThread	Limit MATLAB to a single computational thread. By default, Windows makes use of the multithreading capabilities of the computer on which it is running.

Passing Perl Variables on Startup

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

```
function test(x)
```

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

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

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

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

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

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

- 1 Create a Perl script such as

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

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

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

Startup and Calling Java Software from the MATLAB Program

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

Toolbox Path Caching in the MATLAB Program

In this section...

“About Toolbox Path Caching in the MATLAB Program” on page 1-19

“Using the Cache File Upon Startup” on page 1-19

“Updating the Cache and Cache File” on page 1-19

“Additional Diagnostics with Toolbox Path Caching” on page 1-22

About Toolbox Path Caching in the MATLAB Program

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

Using the Cache File Upon Startup

Upon startup, MATLAB gets information from a cache file to build the toolbox folder cache. Because of the cache file, startup is faster, especially if you run MATLAB from a network server or if you have many toolbox folders. 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 folders, which is slower than using the cache file.

Updating the Cache and Cache File

How the Toolbox Path Cache Works

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

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

When to Update the Cache

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

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

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

MATLAB displays this message:

```
Undefined function or variable
```

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

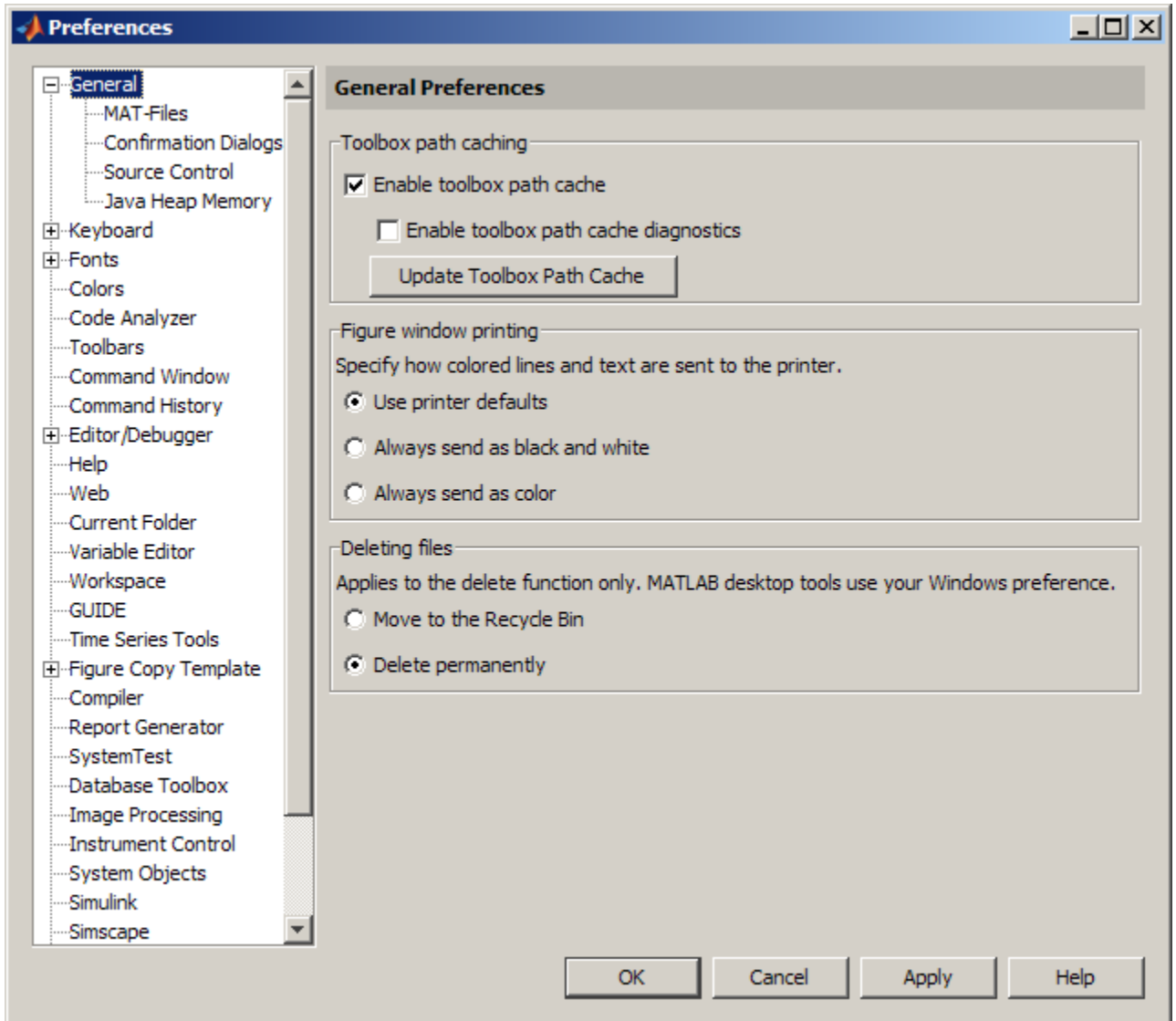
Steps to Update the Cache

To update the cache and the cache file,

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

The **General Preferences** pane is displayed.

- 2** Click **Update Toolbox Path Cache** and click **OK**.



Function Alternative

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

Additional Diagnostics with Toolbox Path Caching

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

Quitting the MATLAB Program

In this section...

“Ways to Quit the MATLAB Program” on page 1-23


“Confirm Quitting the MATLAB Program” on page 1-23

“Running a Script When Quitting the MATLAB Program” on page 1-24

“Abnormal Termination” on page 1-24

Ways to Quit the MATLAB Program

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

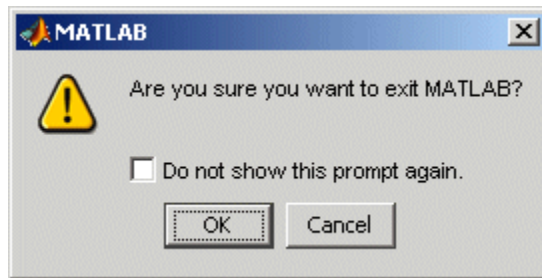
- Click the Close box  in the MATLAB desktop.
- Select **Exit MATLAB** from the desktop **File** menu.
- Type `quit` at the Command Window prompt.

MATLAB closes after

- Prompting you to confirm quitting, if that preference is specified (see “Confirm Quitting the MATLAB Program” on page 1-23)
- Prompting you to save any unsaved files
- Running the `finish.m` script, if it exists in the current folder or on the search path (see “Running a Script When Quitting the MATLAB Program” on page 1-24)

Confirm Quitting the MATLAB Program

To set a preference that displays a confirmation dialog box when you quit MATLAB, select **File > Preferences > General > Confirmation Dialogs**, select the **Confirm before quitting** check box, and click **OK**. MATLAB then displays the following dialog box when you quit.



For more information, see “Confirmation Dialogs Preferences” on page 2-132.

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

Running a Script When Quitting the MATLAB Program

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

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

For more information, see the `finish` reference page.

Abnormal Termination

- “When the MATLAB Program Terminates Unexpectedly” on page 1-25
- “Error Log Reporting” on page 1-26
- “Recovering Data After an Abnormal Termination” on page 1-26

When the MATLAB Program Terminates Unexpectedly

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

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

- 1 If you want to view the stack trace for the problem, click **Details**.
- 2 Click **Close** to terminate MATLAB.
- 3 Restart MATLAB. If the Error Log Reporter dialog box opens, select the option to send a report to MathWorks.

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

- 1 If you want to view the stack trace for the problem, click **Details**.
- 2 Click **Attempt to Continue**. MATLAB tries to return to the Command Window or tool you were using.

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

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

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

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

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

Error Log Reporting

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

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

Send e-mail to segv@mathworks.com with this file attached:

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

Recovering Data After an Abnormal Termination

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

- Use the Command History or the file on which it is based, `history.m`, to run statements from the previous session. You might be able to approximately recreate data as it was prior to the termination. For more information, see “Overview of the Command History Window” on page 3-66.
- If you used the `diary` function or `-logfile` startup option for the session in which MATLAB terminated unexpectedly, you might be able to recover output.

- If you saved the workspace to a MAT-file during the session, you can recover it by loading the MAT-file. For more information, see “Viewing and Loading a Saved Workspace and Importing Data” on page 6-7, and “Saving the Current Workspace” on page 6-5.
- If you were editing a file in the Editor when MATLAB terminated unexpectedly, and you had the autosave preference enabled, you should be able to recover changes you made to files you had not saved.
- If you were in a Simulink session when a segmentation violation occurred, and you have the Simulink **Autosave Options** preference selected, note that the last autosave file for the model reflects the state of the autosave data prior to the segmentation violation. Because Simulink models might be corrupted by a segmentation violation, a model is not autosaved after a segmentation violation occurs.

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

Desktop

- “Desktop Overview” on page 2-2
- “Opening and Arranging Desktop Tools” on page 2-5
- “Opening and Arranging Desktop Documents” on page 2-20
- “Managing Desktop Layouts” on page 2-37
- “Examples of Desktop Arrangements” on page 2-40
- “Running Frequently Used Statement Groups with MATLAB Shortcuts” on page 2-57
- “Performing Desktop Actions Using the Keyboard” on page 2-66
- “Performing Desktop Actions Using Keyboard Shortcuts” on page 2-69
- “Accessing Tools with the Start Button” on page 2-94
- “Using Web Browsers in MATLAB” on page 2-101
- “Other Features for Managing the Desktop” on page 2-108
- “Managing Your Licenses” on page 2-121
- “Check for Updates” on page 2-123
- “Specifying Options for MATLAB Using Preferences” on page 2-124
- “Setting General Preferences for the MATLAB Application” on page 2-129
- “Customizing the Desktop Using Preferences” on page 2-138
- “Accessibility” on page 2-159
- “Macintosh Platform — Differences” on page 2-172

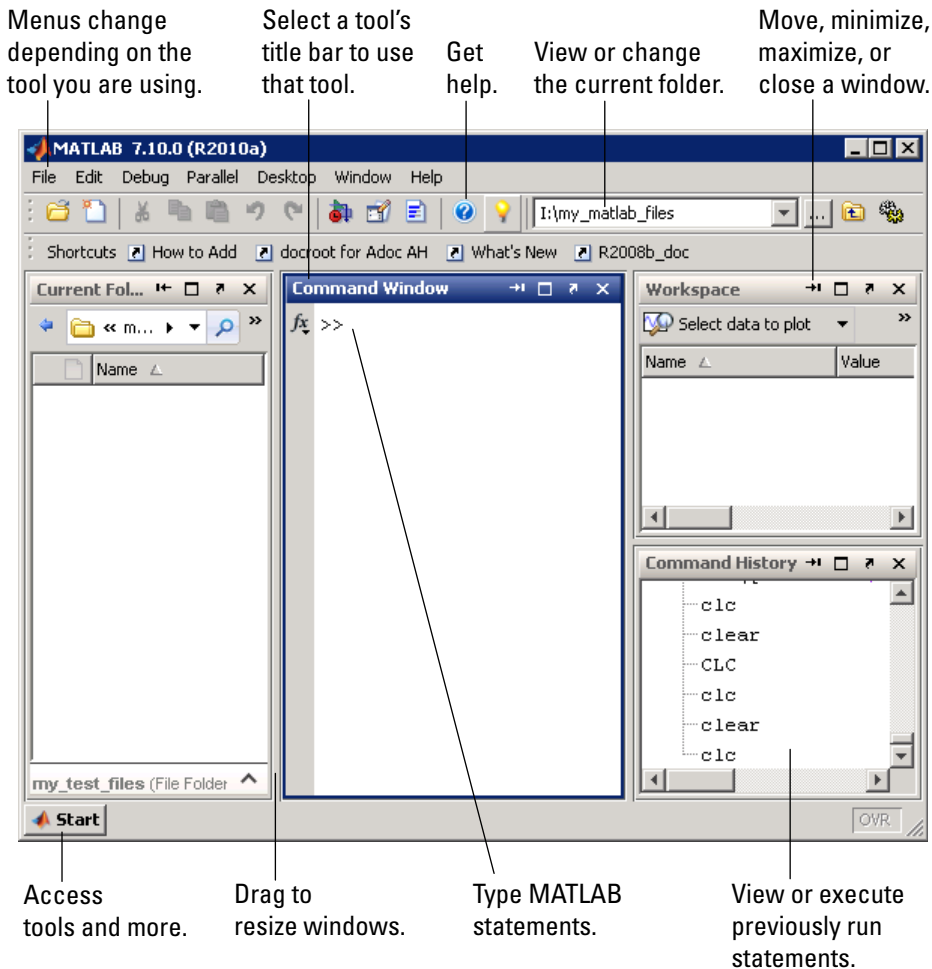
Desktop Overview

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

About the Desktop

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

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



You can change the desktop arrangement to meet your needs, including resizing, moving, and closing tools. The desktop manages tools differently from documents. The Command History and Editor are examples of tools. A file in the Editor and a variable in the Variable Editor are examples of documents. For details, see “Opening and Arranging Desktop Tools” on page 2-5 and “Opening and Arranging Desktop Documents” on page 2-20.

Summary of Desktop Tools

The MATLAB desktop manages the tools listed in the table that follows. Not all of the tools appear by default when you first start MATLAB. If you prefer a command-line interface, you can often use functions to accomplish the same results. The documentation for each tool provides instructions for using functions to perform the task. These instructions are typically labeled as Function Alternatives.

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

Opening and Arranging Desktop Tools

In this section...

- “Opening Desktop Tools” on page 2-5
- “Navigating Among Desktop Tools and Documents” on page 2-7
- “Closing Desktop Tools” on page 2-8
- “Resizing Desktop Tools” on page 2-9
- “Moving Tools Within the Desktop” on page 2-10
- “Undocking Tools to Move Them Outside the Desktop” on page 2-13
- “Moving Undocked Tools Back onto the Desktop” on page 2-14
- “Grouping Desktop Tools Together” on page 2-14
- “Maximizing Available Space on the Desktop” on page 2-16
- “Maximizing Tools Within the Desktop” on page 2-17
- “Minimizing Tools Within the Desktop” on page 2-17

See also “Examples of Desktop Arrangements” on page 2-40.

Opening Desktop Tools

To open a tool, select it from the **Desktop** menu. A check mark in front of the tool name on the menu indicates that the tool is open. The tool opens in the location it occupied the last time you used it. The dimensions of other open tools adjust to accommodate the newly opened tool.

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

- Variable Editor — Open it by double-clicking a variable in the Workspace browser.
- Editor — Open it by creating or opening an existing text file. For instructions, see “Starting, Creating Files, and Closing the Editor” on page 9-6.
- Figures — Create figures using `plot` and other graphics functions.

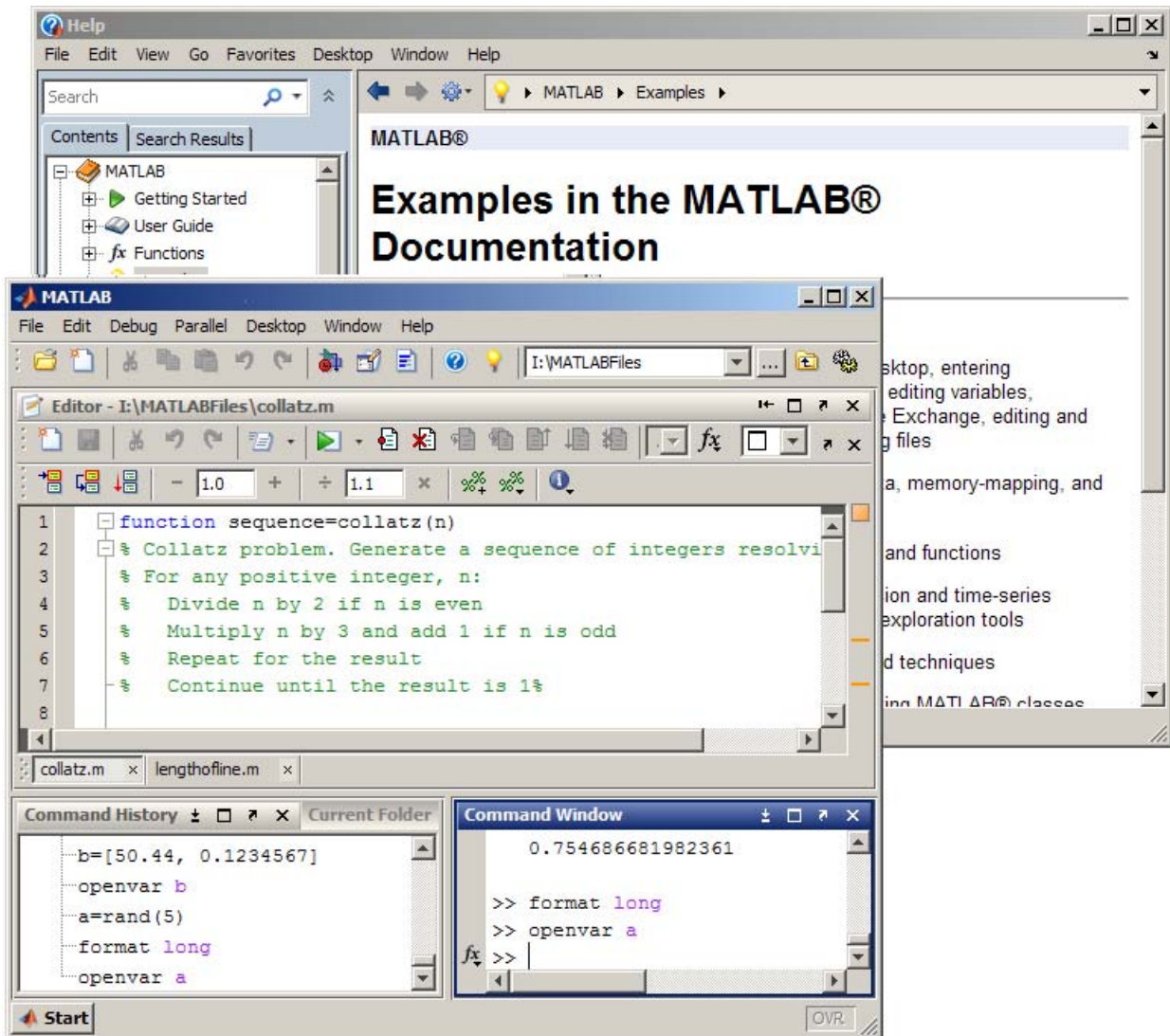
You also can open most desktop tools by:

- Clicking the desktop **Start** button, selecting **Desktop Tools**, and then clicking the tool you want to open.
- Using a function. For example, type `helpbrowser` to open the Help browser. For information on how to open a given tool using a function, see the documentation for that tool.

The following example shows how the MATLAB desktop can look when the following are open:

- Command Window
- Command History window
- Help browser
- The Editor with two open files, `collatz.m` and `lengthoffline.m`

Because the Command Window is the active window, its title bar is dark blue.



Navigating Among Desktop Tools and Documents

- “Navigating Among Desktop Tools and Documents” on page 2-8

- “Making a Tool or Document the Active Window” on page 2-8

Navigating Among Desktop Tools and Documents

You can navigate among desktop tools and documents by:

- Choosing an entry in the **Window** menu
- Using a function that opens the tool or document
- Clicking the entry for an undocked tool or document on the Microsoft Windows task bar (or the equivalent for your platform)

Making a Tool or Document the Active Window

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

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

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

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

See “Keyboard Key Combinations” on page 2-66

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

Closing Desktop Tools

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

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

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

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

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

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

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




Resizing Desktop Tools

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

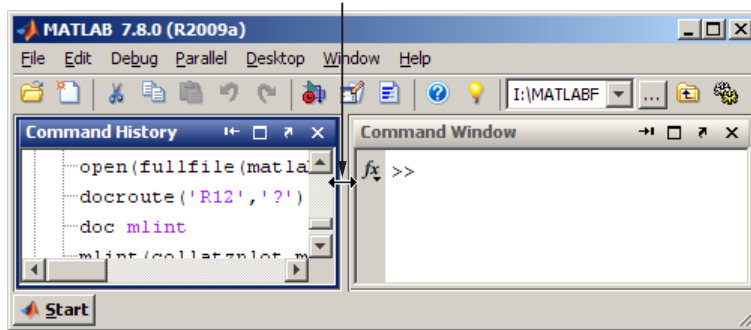
- “Resizing Desktop Tools Using the Mouse” on page 2-9
- “Resizing Desktop Tools Using the Keyboard” on page 2-10

Resizing Desktop Tools Using the Mouse

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

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

Drag the separator bar to resize tools in the desktop.



Resizing Desktop Tools Using the Keyboard

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

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

- 1 Click in the Command Window.
- 2 Press **Alt+D, Z**. This action is the mnemonic equivalent of selecting **Desktop > Resize Command Window**.

The pointer shape becomes an arrow.

- 3 Use the keyboard arrow keys to change the size of the Command Window.
- 4 Press **Enter** to accept the new size, or press **Esc** to return the Command Window to its original size.

Moving Tools Within the Desktop

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

- “Moving Tools Using the Mouse” on page 2-11
- “Moving Tools Using the Keyboard” on page 2-13

Moving Tools Using the Mouse

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

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

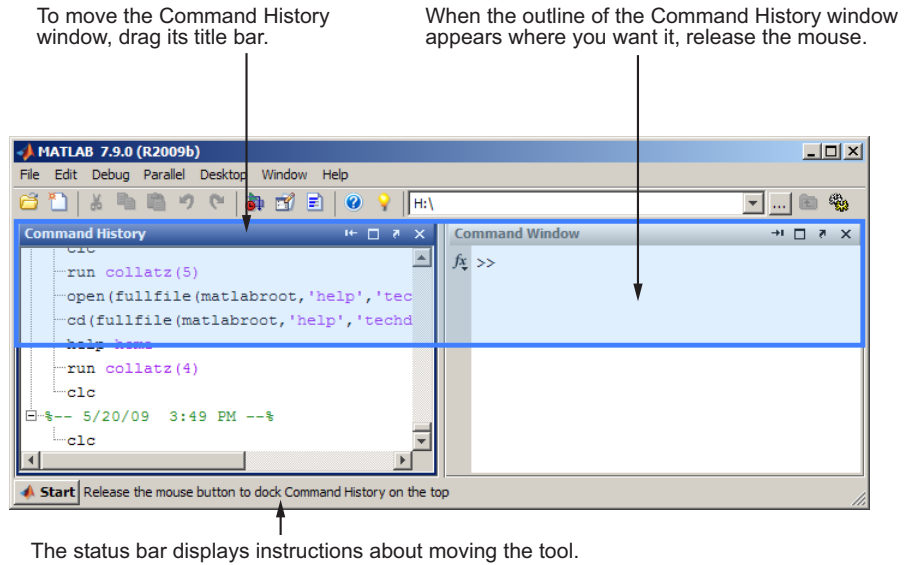
As you drag the tool, an outline of it appears. The status bar indicates where the tool moves if you release the mouse. For instance, it can display:

- Release the mouse to dock the Editor on the top.
- Release the mouse to tab-dock the Current Folder.
- Release the mouse to leave the Editor in the current location.

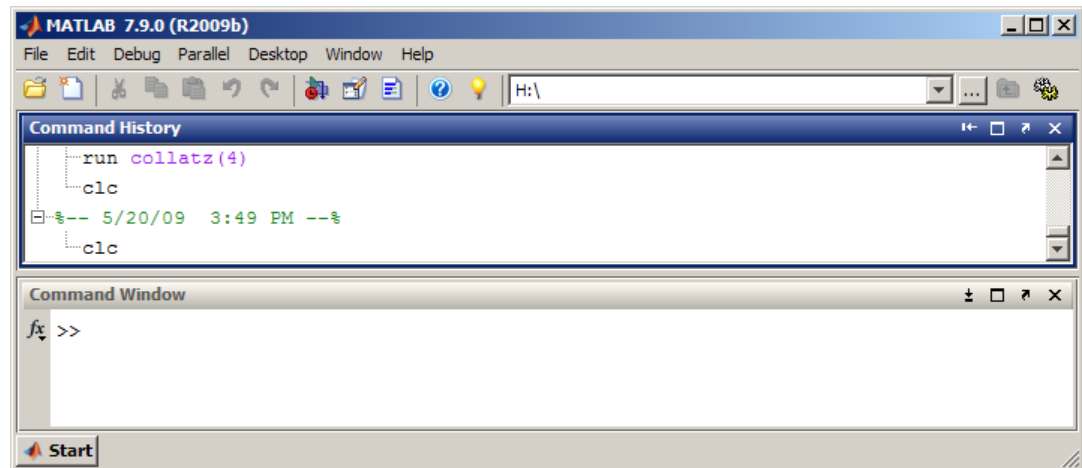
- 2 When the outlined position is where you want the tool to be, release the mouse button.

The tool stays at the new location.

The following illustration shows how it looks as you drag the Command History tool above the Command Window. When you begin dragging the Command History tool, the outline appears around the tool. When you drag it across the boundary separating the two tools, the outline indicates the top-bottom arrangement. If you release the mouse button, you change the arrangement from side-by-side to top-bottom.



Other tools on the desktop automatically resize to accommodate the new configuration. The following example shows how the desktop looks after you move the Command History tool above the Command Window.



Moving Tools Using the Keyboard

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

- 1 Make the Command Window the active tool by pressing **Ctrl+0**.
- 2 Press **Alt+D, V**, which is the mnemonic equivalent for selecting **Desktop > Move Command Window**.


The pointer shape becomes an arrow.

- 3 Use the arrow keys to move the outline of the Command Window to a new location.
- 4 Press **Enter** to keep the tool at the new location, or press **Esc** to return the Command Window to its original position.

Undocking Tools to Move Them Outside the Desktop

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


To move a tool outside the desktop:

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

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

Moving Undocked Tools Back onto the Desktop

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

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

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

Grouping Desktop Tools Together

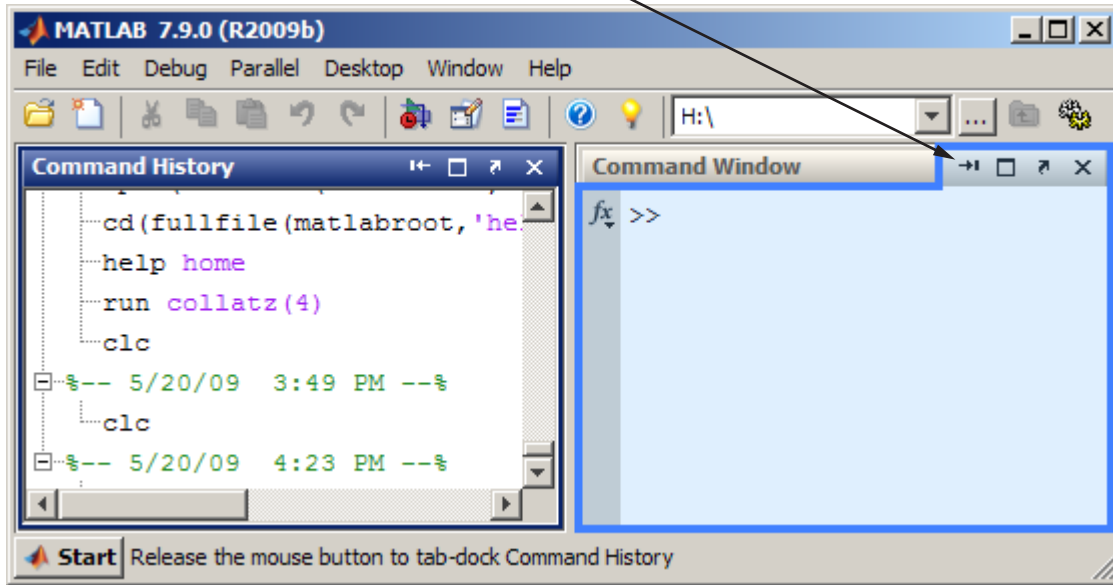
You can group tools so that they occupy the same location on the MATLAB desktop. Basically, you are stacking one tool on top of another. Then, you can access the individual tools using the tool name on the title bar:

To group tools:

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

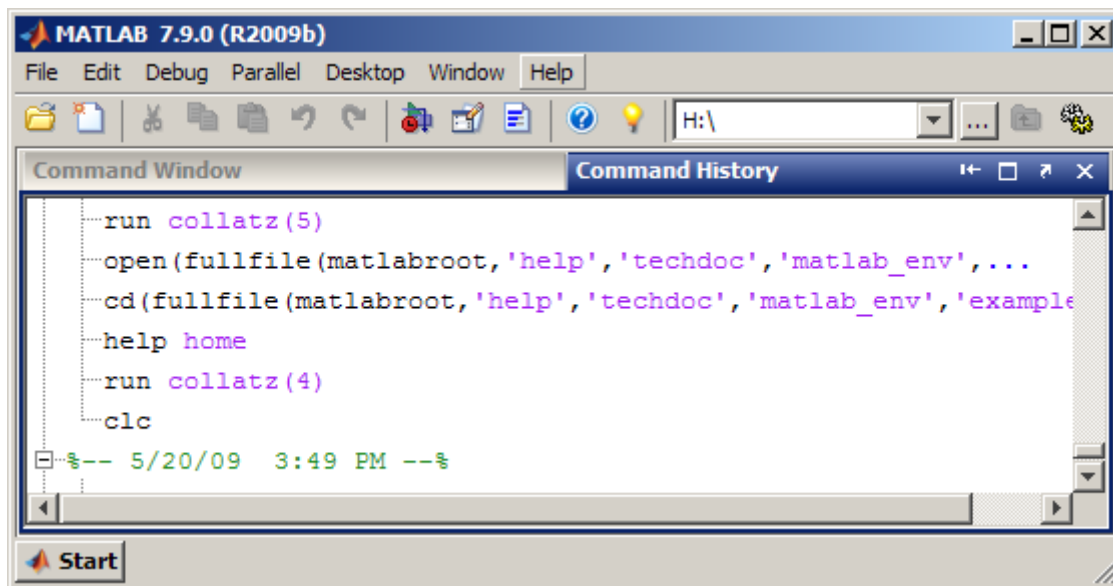
An outline of the tool you are dragging overlies the target tool.

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




2 Release the mouse.

Both tools occupy the same space. Labeled tabs appear at the top of that space.



To view a grouped tool, click the title bar for the tool. The selected tool moves to the foreground and becomes the currently active window.

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


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


Maximizing Available Space on the Desktop

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

Maximizing Tools Within the Desktop

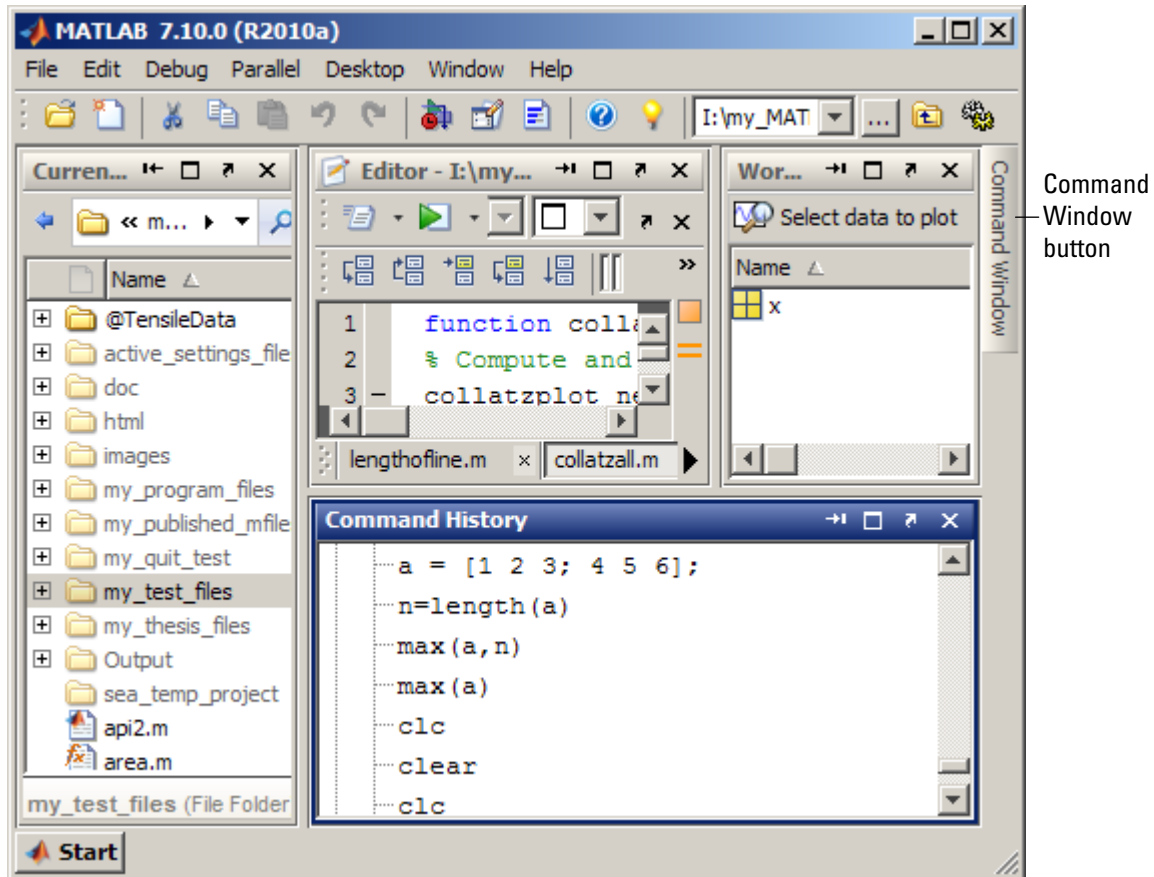
MATLAB software provides multiple ways to maximize tools on the desktop. It also has multiple ways of restoring the desktop to the layout in place before you resized it. For example:

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


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

Minimizing Tools Within the Desktop


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



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

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

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

- To view or use a minimized tool, hover over or click the button for the tool. This action temporarily opens the tool on the desktop. When you finish using the tool, click the Minimize button or another tool. The tool appears again as a button along the edge of the desktop.
- To return the tool to the position it occupied before you minimized it, do one of the following:
 - Double-click the button for the tool.
 - Right-click the button for the tool, and then select **Restore > Toolname**.
 - Hover over or click the button for the tool, and then click the Restore button  on the title bar of the tool.
- To move the button for the tool, drag it to a different edge.

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

Opening and Arranging Desktop Documents

In this section...

“Opening Documents” on page 2-20

“Navigating Among Open Documents Using the Document Bar” on page 2-22

“Adjusting the Document Bar” on page 2-23

“Positioning Documents” on page 2-24

“Moving and Resizing Documents” on page 2-34

“Closing Documents” on page 2-34

“Moving Documents Outside of the Desktop (Undocking)” on page 2-35

“Docking Documents and Tools” on page 2-36

“Grouping Documents in a Tool Outside the Desktop” on page 2-36


Opening Documents

Use the document bar to go to a document that is open, but not in view. The names of all open documents appear on the document bar. Click the document name to open the document. If the document bar is not open, select **Desktop > Document Bar > Bar Position** and select the position for it, for example, **Right**. For more information, see “Navigating Among Open Documents Using the Document Bar” on page 2-22.

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

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

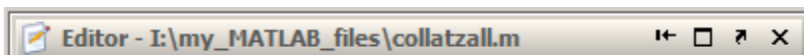
How you open a document depends on the document type, as described in the following table.





Document Type and Tool	How to Open Document	Where Document Appears by Default	Other Techniques to Open Document
Text file in the Editor	Click the Open file button  on the desktop toolbar and select the file.	In the last location of the Editor. The default location for the Editor is outside the desktop.	“Opening Existing Files Using the Editor” on page 9-9
Variable in the Variable Editor	Double-click a variable in the Workspace browser.	In the last location of the Variable Editor. The default location of the Variable Editor is docked on the desktop.	“Opening the Variable Editor” on page 6-24
HTML or similar page in the Web browser	Double-click the file name in the Current Folder browser.	In the last location of the Web browser, replacing the existing Web browser document.	“Using Web Browsers in MATLAB” on page 2-101
Figure	Use the plot function.	In a figure window, outside the desktop.	Any other function or tool that creates a figure window.

Example of Working with Documents on the Desktop

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




- Select a document from the document bar, making it the active open document.
- Use the **Window** menu or equivalent toolbar buttons to position documents.
- Use buttons in the titlebar for a tool. The following image shows the Editor titlebar, for example.



To Accomplish This:	Do This Using the Titlebar for a Tool:
Minimize the tool	Click  .
Maximize the tool	Click  .
Undock the tool from the desktop	Click  .
Close the tool, including all documents in the tool	Click  .

- Use buttons in a toolbar for a tool to arrange documents within a tool. The following image shows the Editor toolbar.



To Accomplish This:	Do This Using the Editor Toolbar:
Arrange documents in the Editor	Use  , as described in “Positioning Documents” on page 2-24.
Undock a document	Click  .
Close and save the document currently displaying	Click  .
Close and not save the document currently displaying	Click Ctrl + X .

See also “Examples of Desktop Arrangements” on page 2-40.

Navigating Among Open Documents Using the Document Bar

When you have more than one document open in a tool, each document appears either maximized (the default), tiled, or floating (cascading). Tiled


and floating arrangements make multiple documents visible simultaneously. The document bar shows the names for all open documents docked together in a tool.

Making a Document Active

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

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

- Select the document from the document bar.

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

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

See also “Performing Desktop Actions Using the Keyboard” on page 2-66.


Adjusting the Document Bar

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

To Accomplish This:	Do This:
Show the document bar.	Select Desktop > Document Bar > Bar Position , and then select a location, for example, Right .
Hide the document bar.	Select Document Bar > Bar Position > Hide from the Window menu or the document bar context menu.

To Accomplish This:	Do This:
Move the document bar.	Do one of the following: <ul style="list-style-type: none"> • Drag it to another location. • Select a new location from the Desktop > Document Bar > Bar Position submenu.
Alphabetize the names of the documents on the document bar. Alphabetizing is useful if you have many documents open at once.	Do one of the following: <ul style="list-style-type: none"> • Right-click on the document bar and select Alphabetize. • Select Desktop > Document Bar > Alphabetize.
Reorder document names on the document bar.	Do one of the following: <ul style="list-style-type: none"> • Drag a document name to a different position on the document bar. • Select Move documentname On Bar and select a direction. For example, select to Beginning from either the Desktop > Document Bar menu or from the document bar context menu.
Widen or narrow document names on the document bar. If document names are long, or if you have many documents open, the entire document name does not display. Instead, you see the first few characters followed by ellipsis (...).	Do one of the following: <ul style="list-style-type: none"> • When the document bar is on the top or bottom drag the separator bar between two names on the bar. (Do this, for example, to see an entire document name.) • When the document bar is on the left or right, change the width of the bar by dragging its left or right edge.

Positioning Documents

You can position open documents so that one document or multiple documents are in view from within a tool. Select the arrangement from the **Window** menu or use the Arrange Documents drop-down menu , as described in

the sections that follow. When you *tile* documents, they are all visible within the tool, arranged in a grid pattern.

- “Viewing One Document (Default)” on page 2-25
- “Viewing All Open Documents, Layered on Top of One Another” on page 2-25
- “Viewing Documents, Side-By-Side” on page 2-26
- “Viewing Open Documents, One Above the Other” on page 2-26
- “Viewing Open Documents, Tiled Within the Tool” on page 2-26
- “Viewing a Subset of Open Documents, Tiled Within the Tool” on page 2-30
- “Replacing a Tiled Document with an Out-Of-View Document” on page 2-31

Viewing One Document (Default)

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

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

The illustration in “Example of Working with Documents on the Desktop” on page 2-21 shows this arrangement.

Viewing All Open Documents, Layered on Top of One Another


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

- Select **Window > Float**.
- In the **Arrange Documents** drop-down menu, choose the **Float** option .

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

Viewing Documents, Side-By-Side


To use the **Tile** option to view two documents side-by-side, use one of the following methods:

- Select **Window > Left/Right Tile**.
- In the Arrange Documents drop-down menu, choose the Left/Right Tile option .

See also “Displaying Two Parts of a File Simultaneously” on page 9-67 that enables you to view two different parts of the same file simultaneously.

Viewing Open Documents, One Above the Other


You can use the **Top/Bottom Tile** option to view two documents stacked one above the other by using one of the following methods:

- Select **Window > Top/Bottom Tile**.
- In the Arrange Documents drop-down menu, choose the Top/Bottom Tile option .

See also “Displaying Two Parts of a File Simultaneously” on page 9-67 that enables you to view two different parts of the same file simultaneously.

Viewing Open Documents, Tiled Within the Tool

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

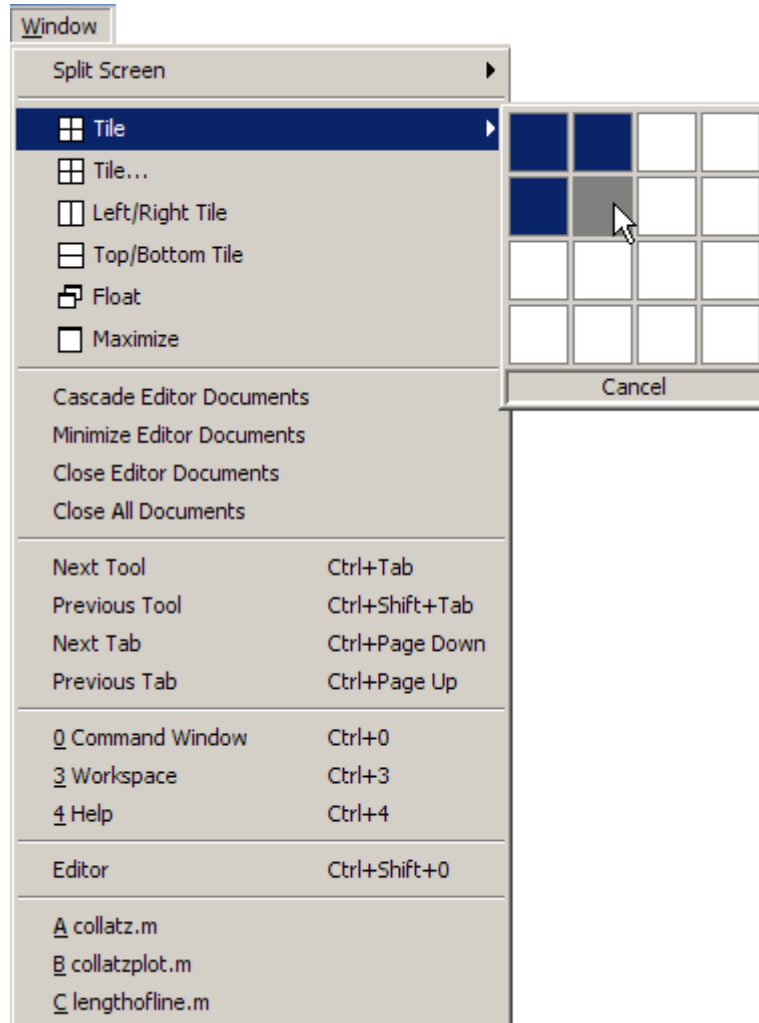
- 1 Select the tiling option using one of these methods:
 - Select **Window > Tile**.
On the Apple Macintosh platform, this option might be unavailable, so use the drop-down menu instead.
 - In the **Arrange Documents** drop-down menu, choose the Tile option .

A four-by-four grid displays.

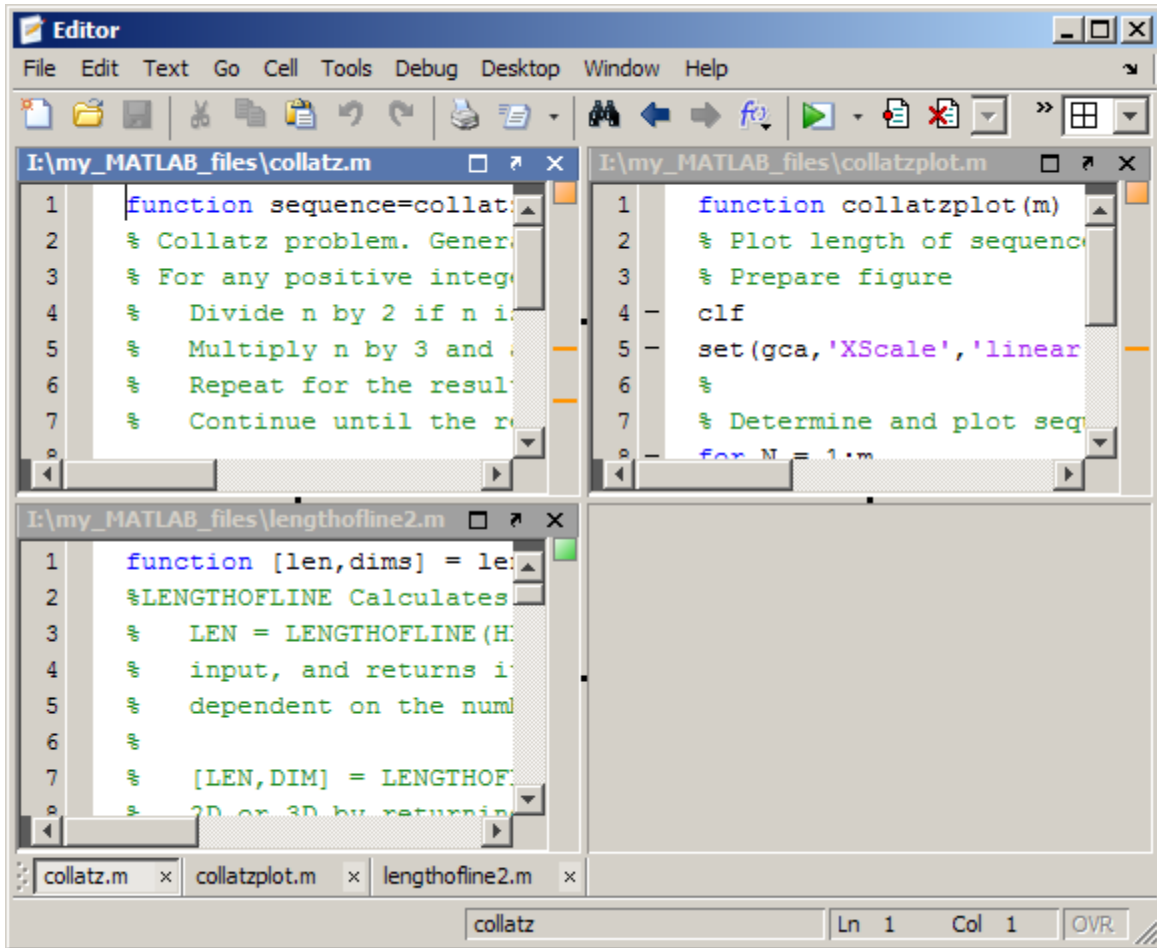
- 2 Move the pointer across the grid to define the number and position of the tiles, as shown in the following illustration.

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



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



The following arrangement shows three documents tiled in the Editor. The Editor is undocked from the desktop.



Closing an Empty Tile.

- 1 Move the pointer over the handle  on the separator bar.
- 2 Click the Close box  that appears where the handle previously appeared.

Covering One Document with Another. Click the handle between two open documents to move one on top of the other.

Viewing a Subset of Open Documents, Tiled Within the Tool

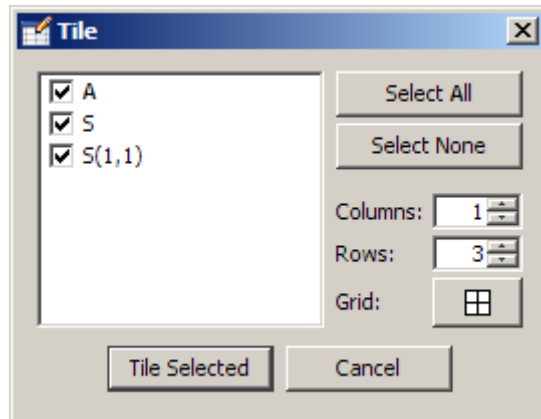
To view only a subset of all your open documents displayed in tile format, follow these steps:

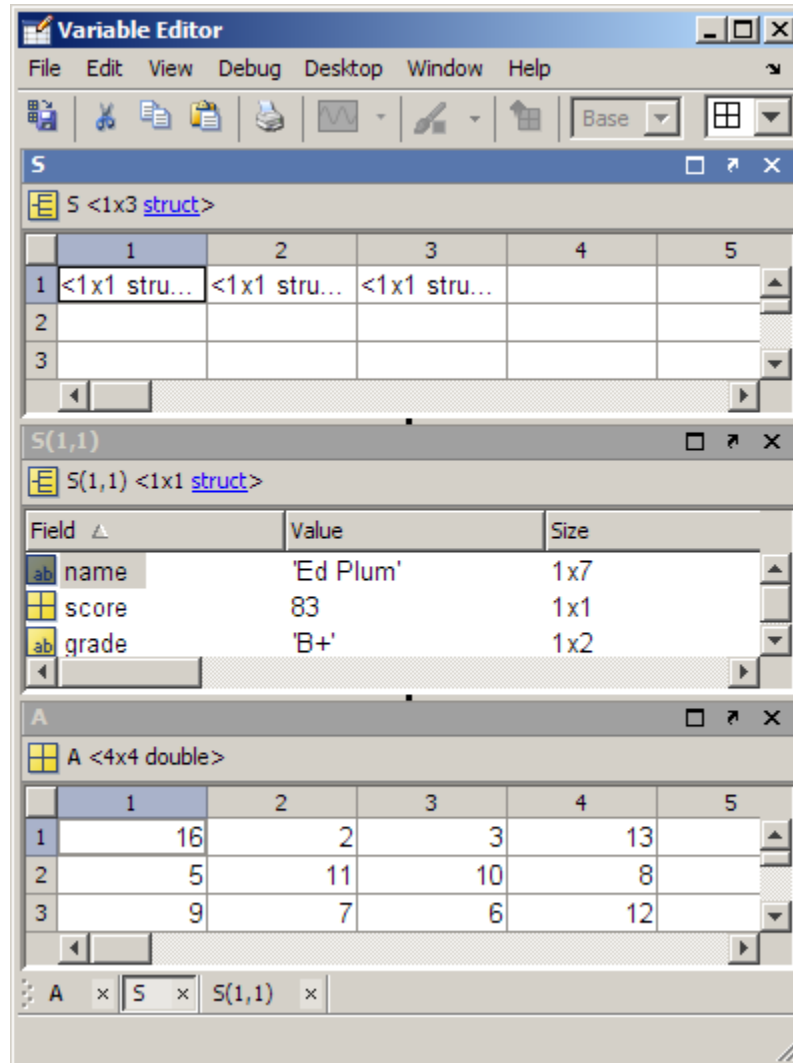
- 1** Select **Window > Tile**.

The Tile dialog box opens.

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

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

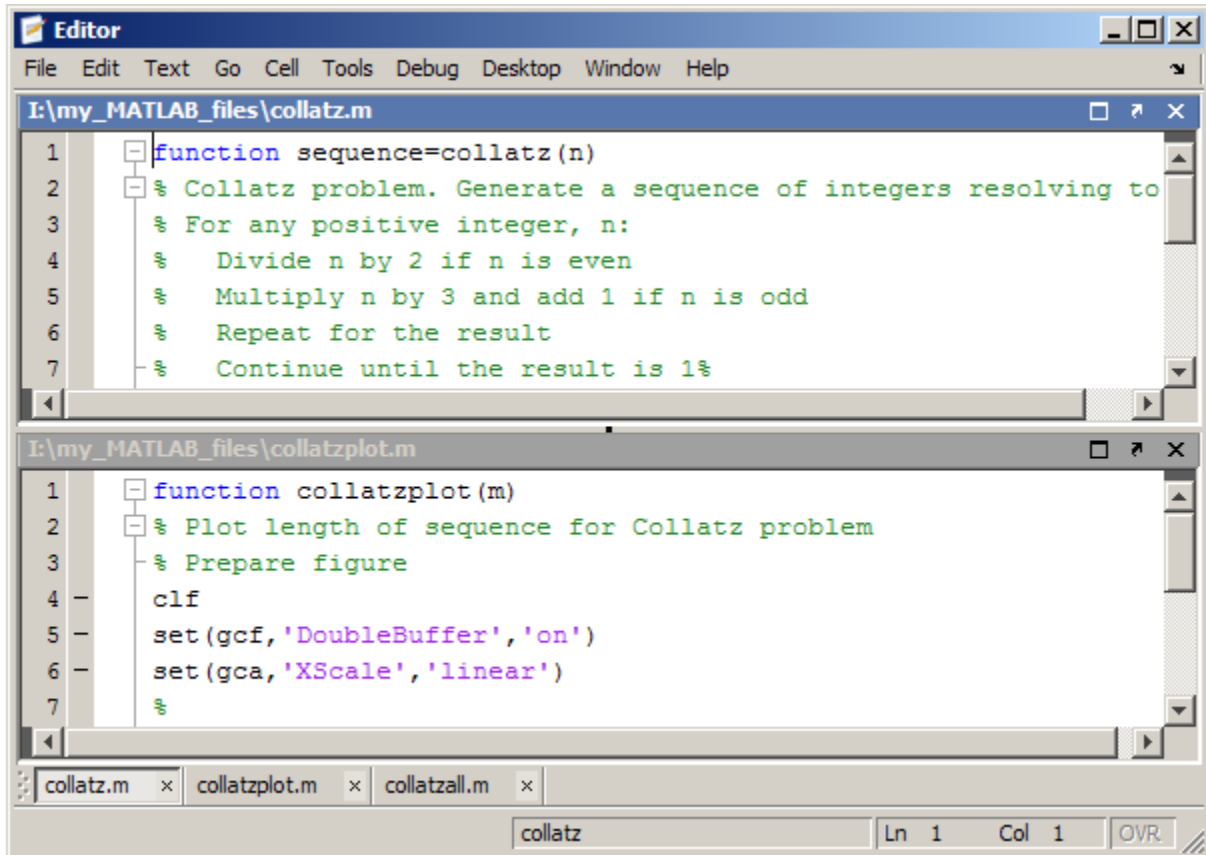




Replacing a Tiled Document with an Out-Of-View Document

You can replace a currently tiled document with another that is on the document bar, but not in currently in view. However, once you open a document in a particular tile, that document always displays in the same tile unless you drag the document to a new tile.

For example, suppose you have three documents open in the Editor — `collatz.m`, `collatzplot.m`, and `collatzall.m`. The first two documents are in view, as shown in the following image.

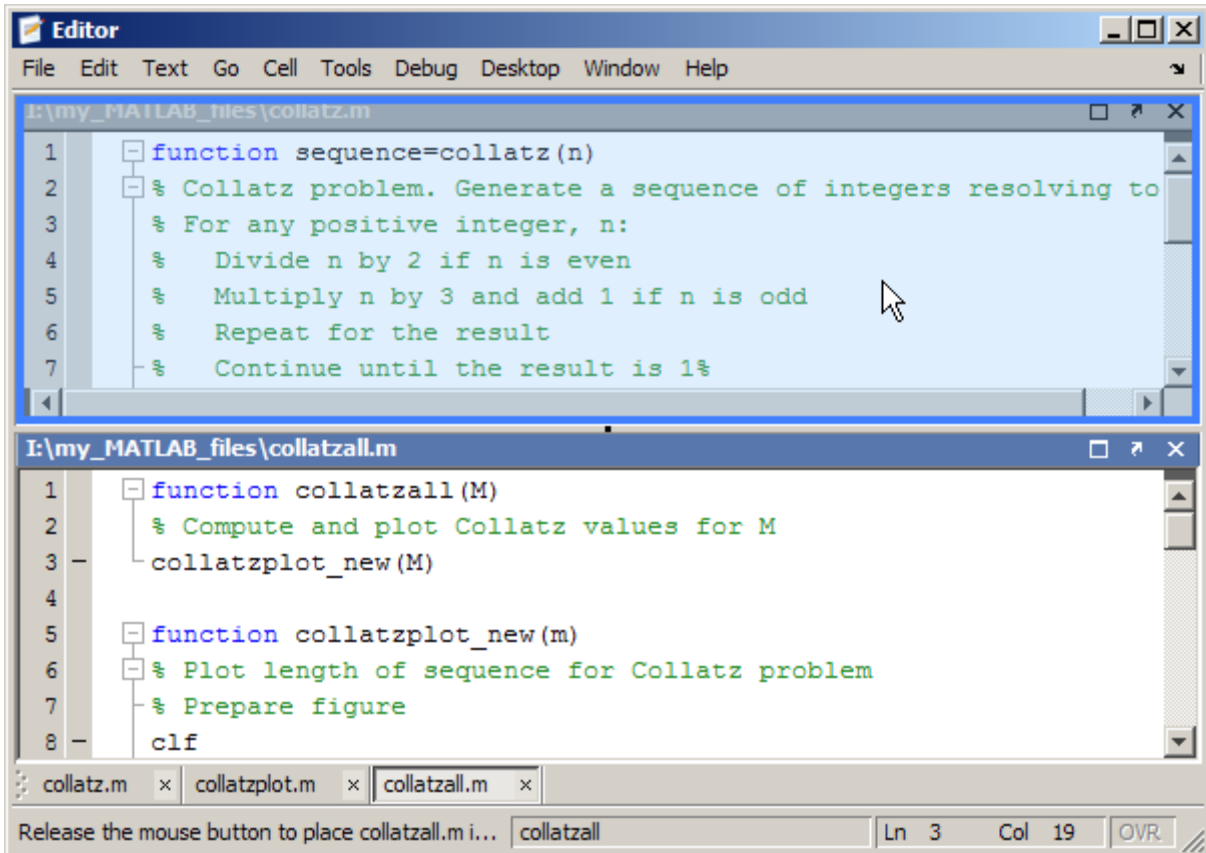


Suppose you want to view `collatzall.m` in the top tile. Follow these steps:

- 1 If you did not already select it, click the title bar of the file in the top tile, `collatz.m`.
- 2 On the document bar, click the name of the file you want to view instead, `collatzall.m`. Assuming that you have not previously viewed `collatzall.m` in the bottom tile, `collatzall.m` displays in the top tile.

However, if you previously viewed `collatzall.m` in the bottom tile, `collatzall.m` displays in the bottom tile, regardless of which title bar you click in step 1.

- 3 If `collatzall.m` displays in the bottom tile, drag its title bar to the top tile to get the arrangement you want.



Now, `collatzall.m` displays in the top tile and `collatzplot.m` displays in the bottom tile.

Moving and Resizing Documents


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

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

Closing Documents

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

- Click the Close box **✕** on the title bar for the tool.
MATLAB closes all the documents open within the tool. Any undocked documents remain open.
- Right-click the title bar for the document, and then select the **Close** option.


- Right-click the name of the document on the document bar, and then select one of the **Close** options.
- Click the Close box  next to the name of the document on the document bar.
- To close a document or documents without saving changes, hold the **Ctrl** key while clicking the Close box.
- Place the mouse pointer over the name of the document on the document bar, and then click the middle mouse button. (Works on Microsoft Windows and Linux only.)
- Select **Window > Close *ToolName* Documents**.
- Select **File > Close *Current_Document_Name***.
- For an undocked document or tool, right-click the Windows task bar entry (or equivalent for your platform) and select **Close**.
- When there are open documents, undocked from within their tools, close all open documents and the tool by selecting **Window > Close All Documents** from the desktop.

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

Moving Documents Outside of the Desktop (Undocking)

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


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

- Click the Undock arrow  on the title bar of the tool you want to move outside the desktop.
- Make the tool you want to move outside the desktop active. Then for that tool, select **Desktop > Undock**.

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

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

Docking Documents and Tools

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

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

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

Grouping Documents in a Tool Outside the Desktop

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

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

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

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

- 2** Undock the tool.

Managing Desktop Layouts

In this section...

“Overview of Desktop Layouts” on page 2-37

“Saving a Desktop Layout” on page 2-37

“Reusing a Saved or Predefined Desktop Layout” on page 2-38

“Renaming a Saved Desktop Layout” on page 2-38

“Deleting a Saved Desktop Layout” on page 2-39

“Restoring the Default Desktop Layout” on page 2-39

Overview of Desktop Layouts

When you end a session, MATLAB saves the current desktop arrangement. The next time you start MATLAB, the desktop appears like the way you left it. However, tools such as the Help browser, Web browser, and Array Editor do not reopen automatically, even if they were open when you ended the last session. You can use startup options to specify tools that you want to open on startup. For example, to have the Help browser open each time you start MATLAB, add `helpbrowser` to a `startup.m` file. For more information, see “Startup Options” on page 1-14.

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

Saving a Desktop Layout

To save your current desktop arrangement:

- 1 Select **Desktop > Save > Layout**.
- 2 Assign a name to the layout in the resulting dialog box, and then click **OK**.

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

Reusing a Saved or Predefined Desktop Layout

Select **Desktop > Desktop Layout**, and then select the name of the layout you want to use.

MATLAB includes the following predefined layouts:

- **Default** — Contains the Current Folder, Command Window, Workspace Browser, and Command History windows.
- **Command Window Only** — Contains the Command Window only.
- **History and Command Window** — Contains the Command History window and Command Window.
- **All Tabbed** — Contains all desktop tools, opened, maximized, and tabbed together.
- **All but Command Window Minimized** — Contains all tools, opened and minimized in the desktop, except for the Command Window and sometimes the Editor. The Command Window and the Editor (if it contains a document) remain maximized.

When you select a saved or predefined layout, document tools already open in the desktop remain open.

Renaming a Saved Desktop Layout

Rename a desktop layout that you have previously created and saved as follows:

- 1** Select **Desktop > Organize Layouts**.
- 2** In the resulting dialog box, select a layout, click the **Rename** button.
- 3** Type the new name over the existing name.
- 4** Click **Close**.

You can rename desktop layouts that you created only.

Deleting a Saved Desktop Layout

Delete a desktop layout that you have previously created and saved as follows:

- 1** Select **Desktop > Organize Layouts**.
- 2** In the resulting dialog box, select a layout, click the **Delete** button, and then click **Close**.

You can delete desktop layouts that you created only.

Restoring the Default Desktop Layout

If you are dissatisfied with your current desktop arrangement, you can restore it to the default arrangement as follows:

Select **Desktop > Desktop Layout > Default**

The default arrangement is that which appeared when you first installed MATLAB.

Examples of Desktop Arrangements

In this section...

“About These Examples” on page 2-41

“Tool Outside of Desktop and Other Tools Grouped Inside Desktop Example” on page 2-41

“Maximized Tool in Desktop Example” on page 2-43

“Minimized Tools in Desktop Example” on page 2-44

“Tiled Documents in Desktop Example” on page 2-48

“No Empty Document Tiles Example” on page 2-49

“Maximized Documents Outside of the Desktop Example” on page 2-52

“Floating (Cascaded) Figures in Desktop Example” on page 2-53

“Undocked Tools and Documents Example” on page 2-55


About These Examples

Scan the illustrations in the following examples for a desktop arrangement like what you want, and then follow the brief instructions to achieve the arrangement. There are many different ways to accomplish the result; these instructions present just one way. The instructions might not apply exactly, depending on how your desktop looks before you start. For details, see “Opening and Arranging Desktop Tools” on page 2-5 and “Opening and Arranging Desktop Documents” on page 2-20.

Tool Outside of Desktop and Other Tools Grouped Inside Desktop Example

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

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

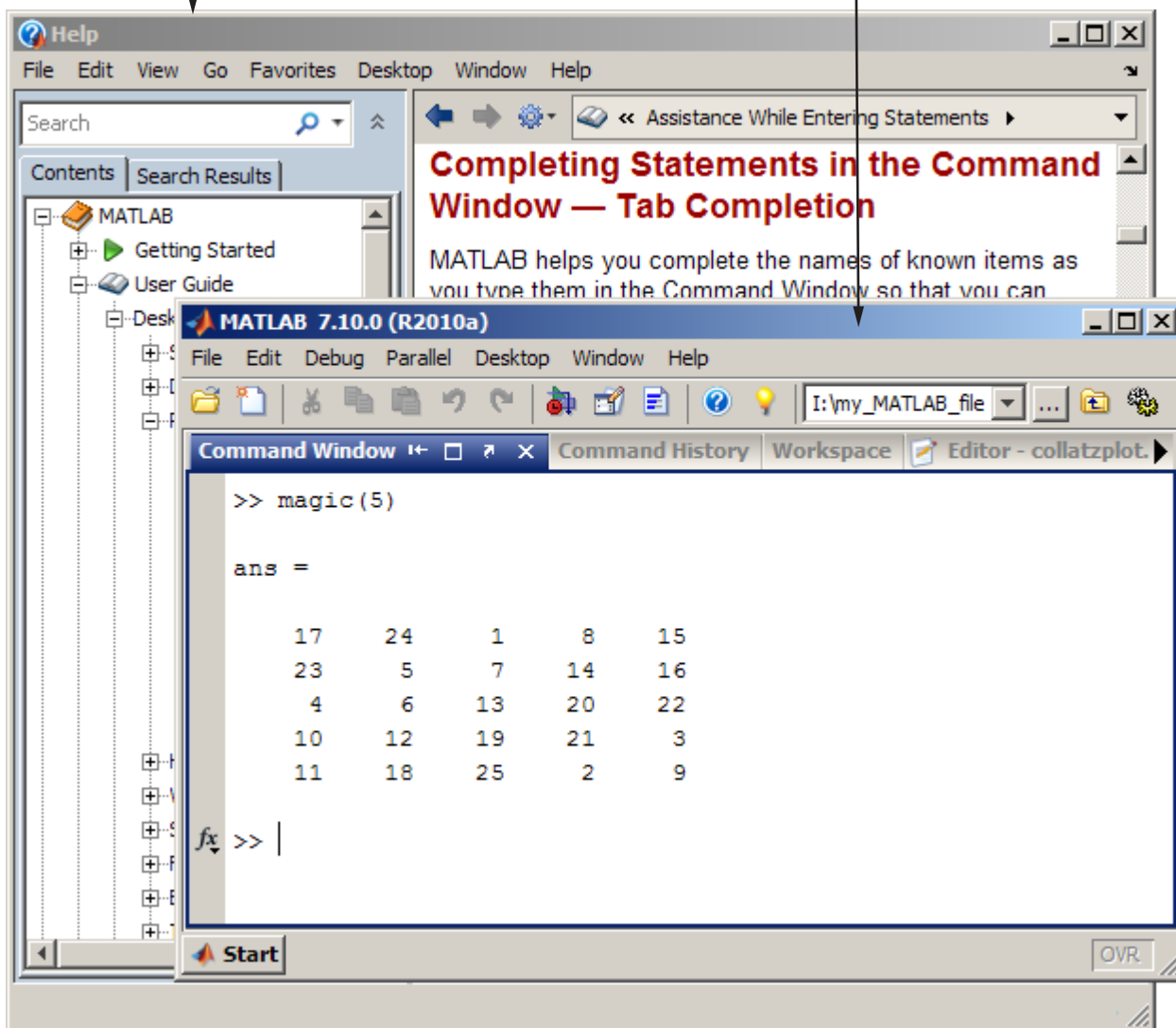
In the illustration that follows, the Help browser is outside of the desktop and made larger. To move a tool outside of the desktop, click the Undock button  on the title bar of the tool when the tool is in the desktop.

- Group tools inside the desktop, and then access a particular tool by clicking the name of that tool on the title bar.

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


Help browser undocked from desktop.

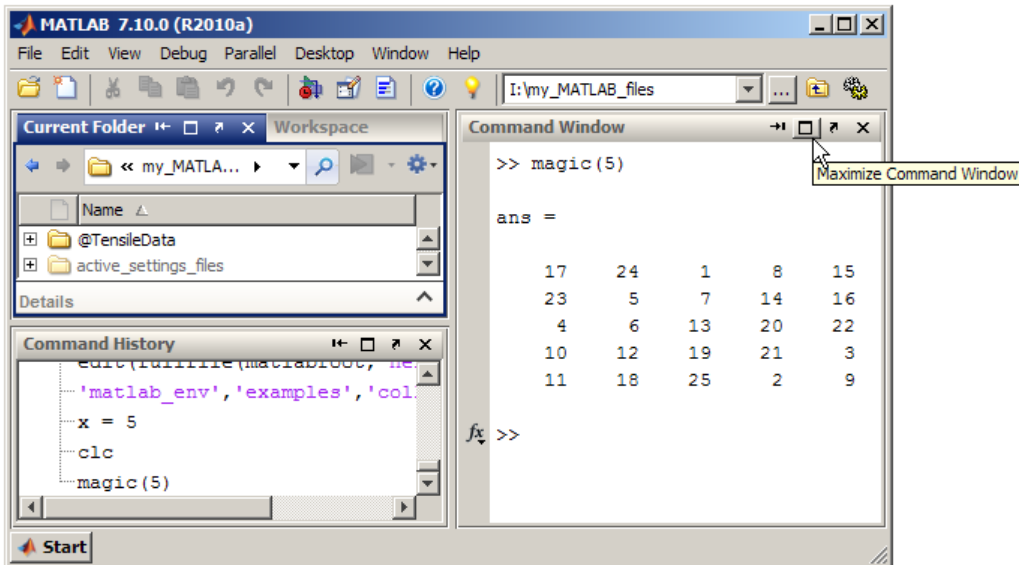
Four desktop tools grouped together.




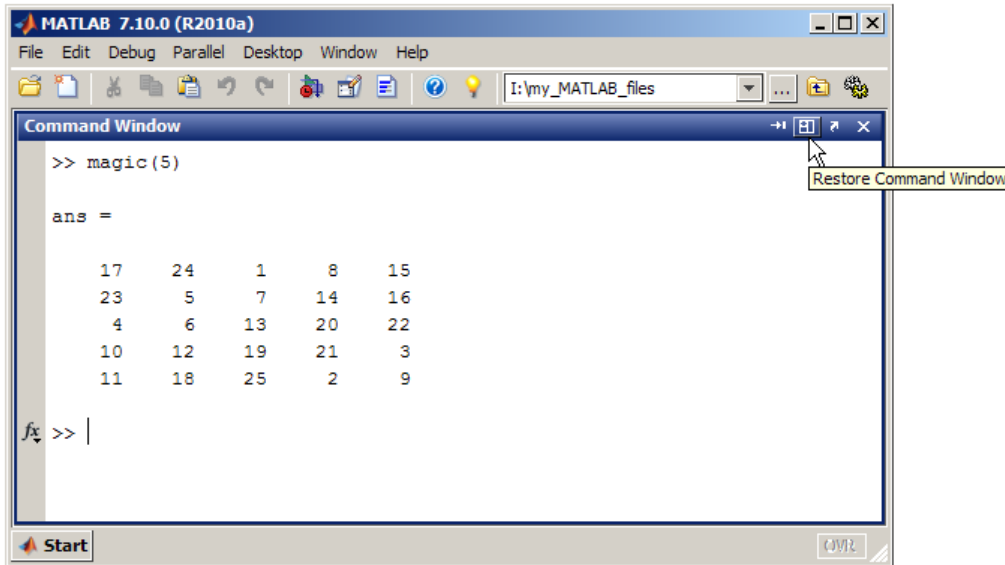
Maximized Tool in Desktop Example

This example shows a way to increase the size of a tool temporarily so that it occupies the entire area of the desktop.

- 1 Click the Maximize button  on the Command Window title bar.



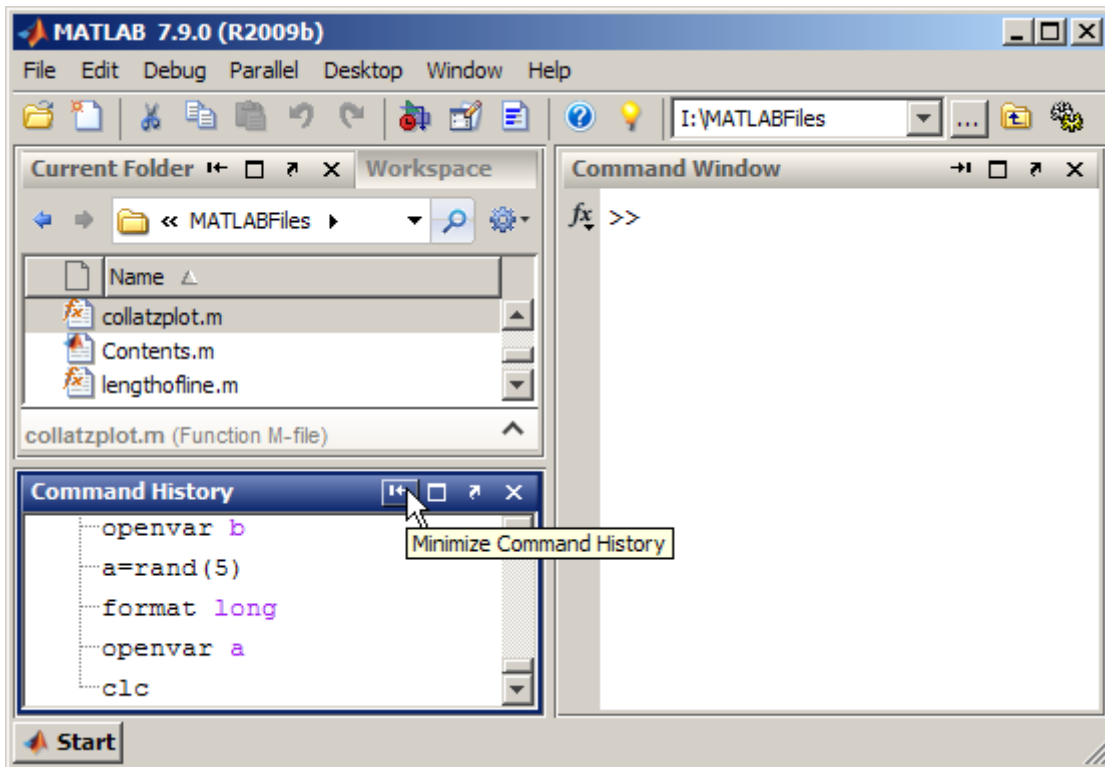
- 2 Return the maximized Command Window to its previous size and position in the desktop by clicking the Restore button  on the Command Window title bar.



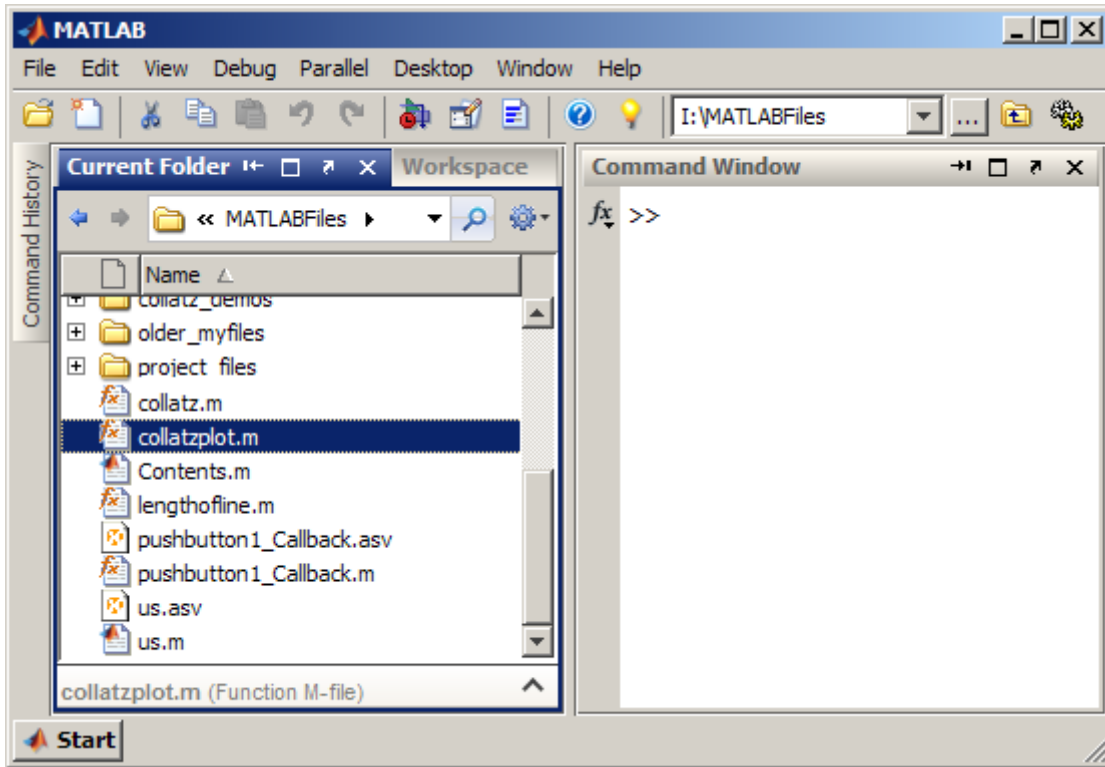
Minimized Tools in Desktop Example

Minimize a tool in the desktop to give the remaining desktop tools more space in the desktop. Minimizing is available on Microsoft Windows and UNIX² platforms. This image shows the button and associated tooltip for minimizing the Command History window to the left edge of the desktop.

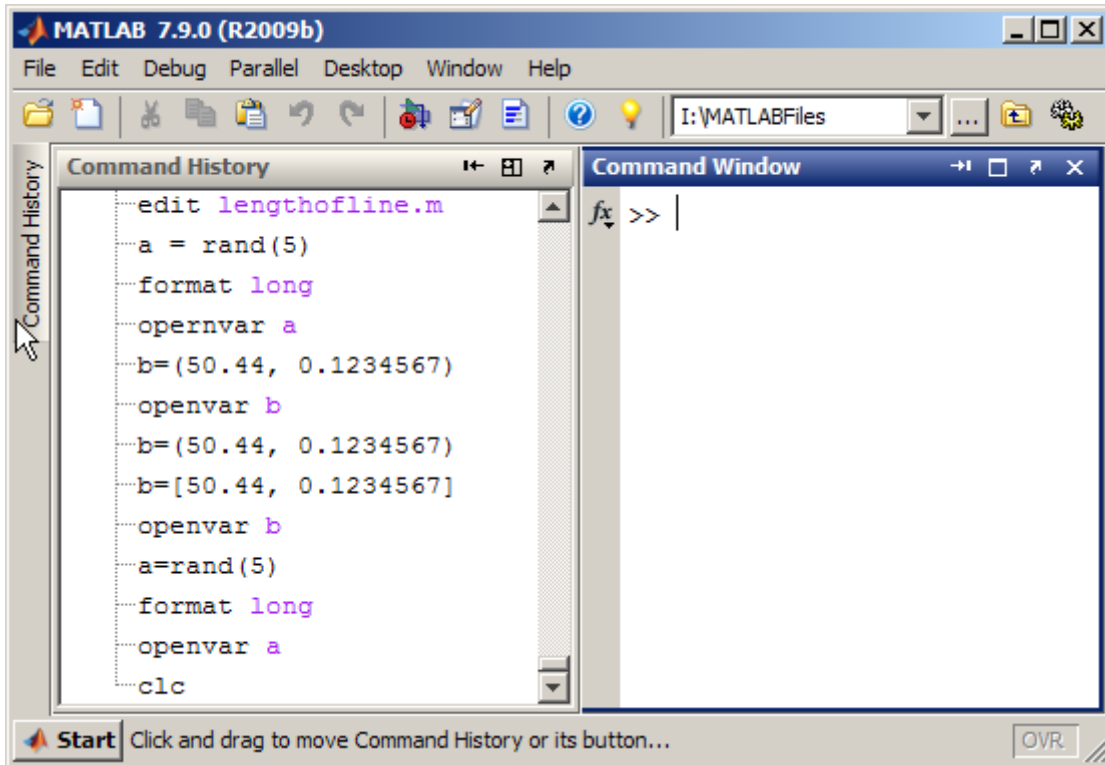
2. UNIX is a registered trademark of The Open Group in the United States and other countries.



This image shows the Command History minimized. It appears as a button along the left edge.




To view or use a minimized tool temporarily, hover over or click the button representing the minimized tool. MATLAB temporarily displays the tool. This illustration shows the minimized Command History temporarily open, as a result of hovering over the button.



When you select another tool, the tool on temporary display becomes minimized again.


To return the Command History to the position and size it occupied in the desktop before minimizing, do one of the following:


- Click the button representing the minimized tool, and then click the **Restore** button 
- Right-click the button representing the minimized tool, and then choose the **Restore** option.

Tiled Documents in Desktop Example

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

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

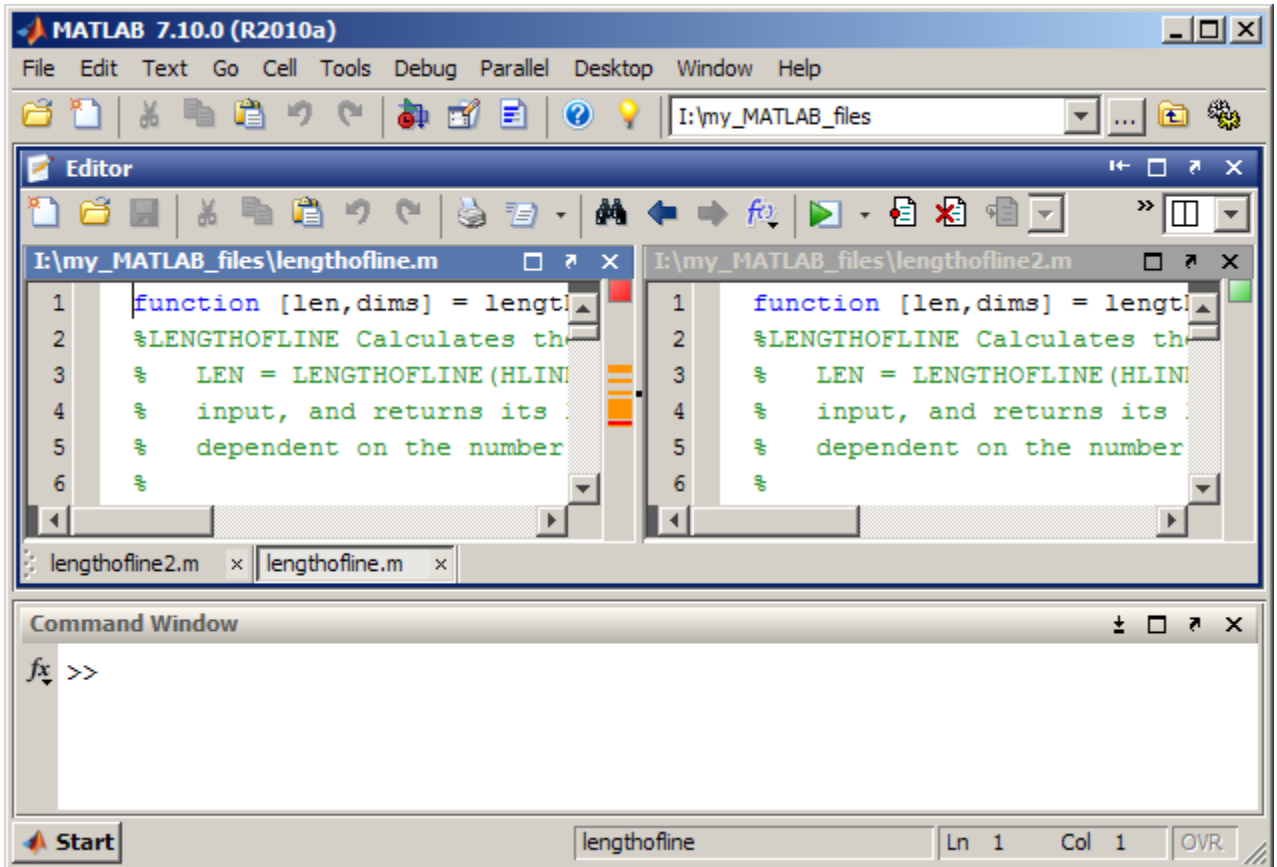
After changing the arrangement from the default, you can restore it by selecting **Window > Maximize** (or the  toolbar button).

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

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


- To hide (or show) a toolbar, select **Desktop > Toolbar name**.
- To see or move the document bar, select **Desktop > Document Bar > Bar Position**, and choose its location, for example, **Top**.

The following example shows two files, side-by-side in the Editor, and the desktop shortcuts toolbar hidden.



No Empty Document Tiles Example

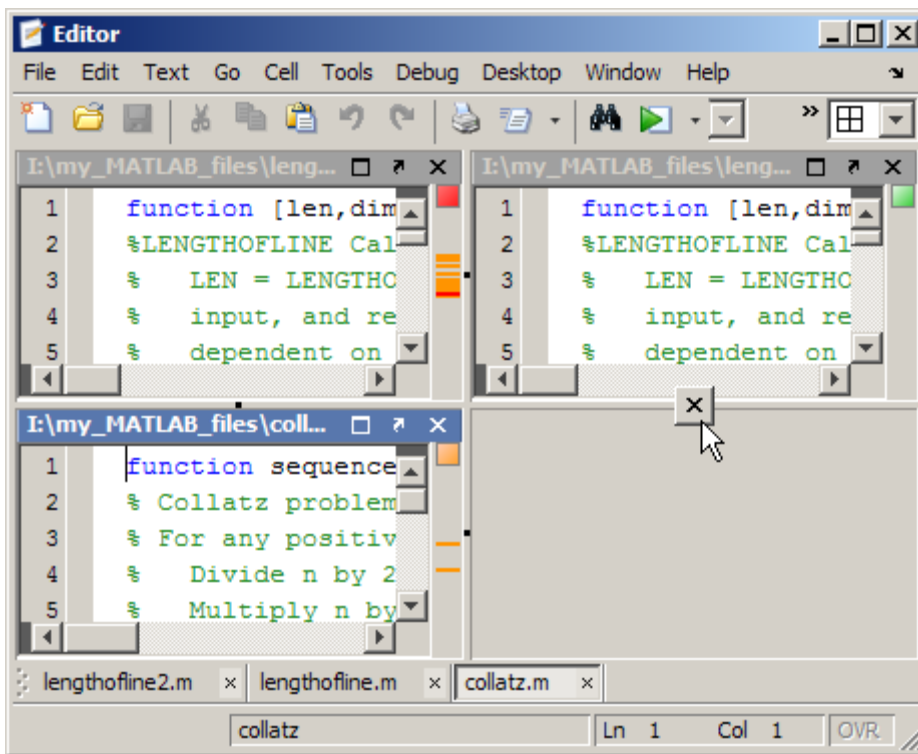
The following example illustrates many of the options described in this list for creating and manipulating tiled documents.

- To see more than two documents at once:
 - 1** Click the Arrange Document down arrow button  and select the 4x4 grid pattern.
 - 2** In the grid that appears, move the pointer to select the number of tiles you want.

If you choose more tiles than the number of currently open documents, the extra tiles appear as empty grey tiles.

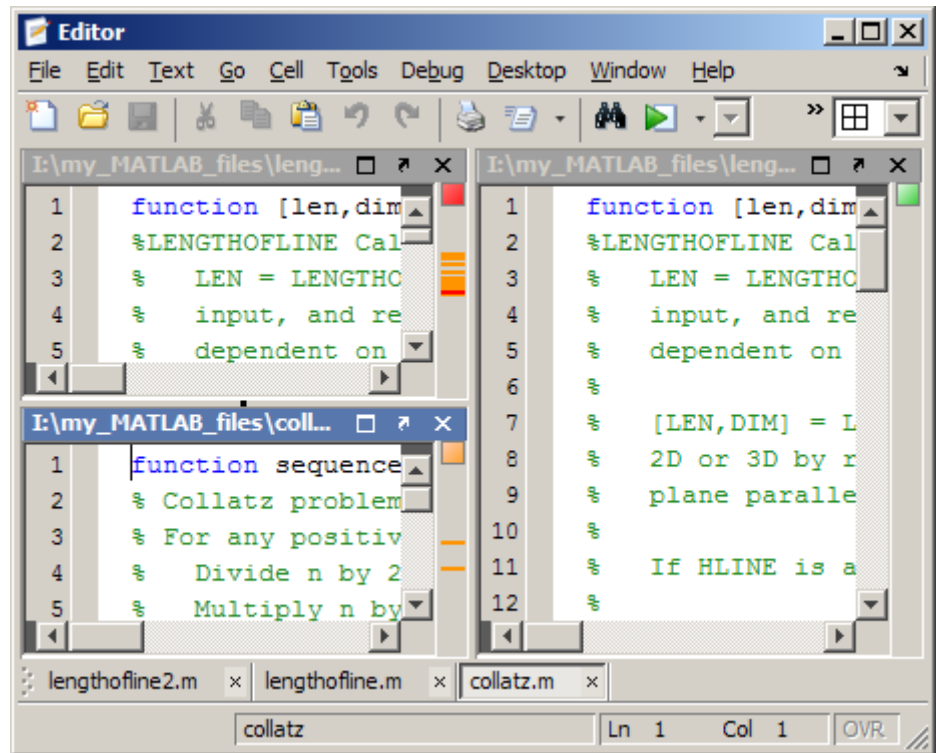
- To move a document to any empty tile, drag its title bar to the new location.
- To close an empty tile:
 - 1 Position the pointer over the handle **.** on the separator bar.

The handle becomes a Close box.



- 2 Click the Close box.

The empty tile closes, and the neighboring document expands.



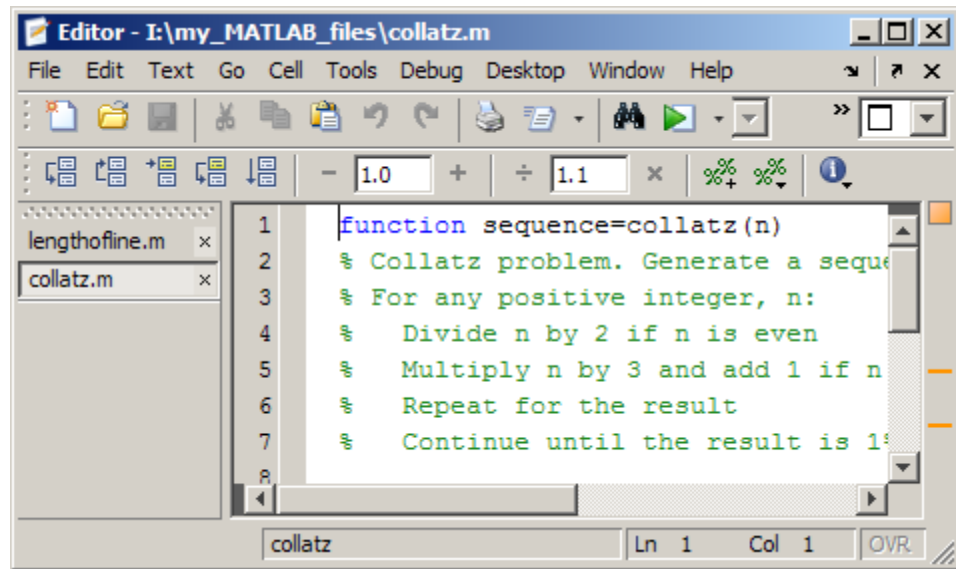
- To hide one document behind another, click the Close box between two tiles containing documents.
One document becomes hidden.

Maximized Documents Outside of the Desktop Example

This example illustrates a way to provide a large area for multiple documents, in this case, files maximized in the undocked Editor. To:


- Group all documents in the Editor, select **Desktop > Dock All in Editor** from any Editor document.
- Move all Editor documents outside of the desktop, select **Desktop > Undock Editor** when the Editor is the active window.
- Make a document occupy the full area in the Editor, click the Maximize button in the Editor toolbar, or select **Window > Maximize**.
- Display the cell toolbar, select **Desktop > Cell Toolbar**. This menu item is available only when the current document has a .m file extension. The cell toolbar displays below the Editor toolbar.
- Display the document bar on the left side of the Editor, select **Desktop > Bar Position > Document Bar > Left** from the Editor.

The following image shows how making the previous choices affects the Editor display.

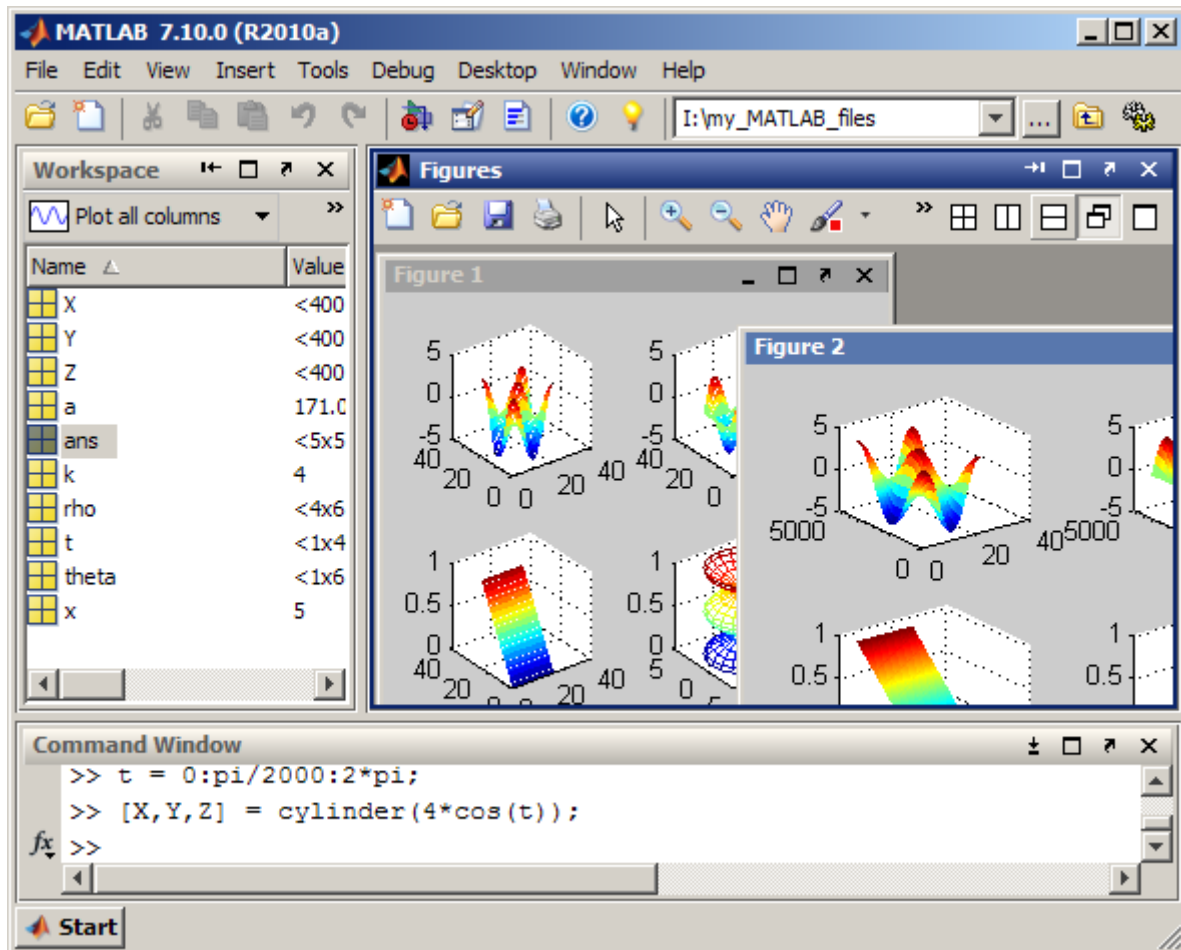


Floating (Cascaded) Figures in Desktop Example

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

- To move the figures into the desktop, click the Dock button on the menu bar of each figure.
- To float (also called cascade) the figures, select **Window > Float**, or click the Float button .
- To get more screen area for the figures, hide the document bar by selecting **Desktop > Document Bar > Bar Position > Hide**.

The following image shows how making the previous choices can affect the Desktop display.



Undocked Tools and Documents Example

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

To undock a tool and its documents:

- 1 Select **Desktop > Undock *Toolname***.
- 2 Select **Desktop > Undock *Documentname*** from the tool.

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

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

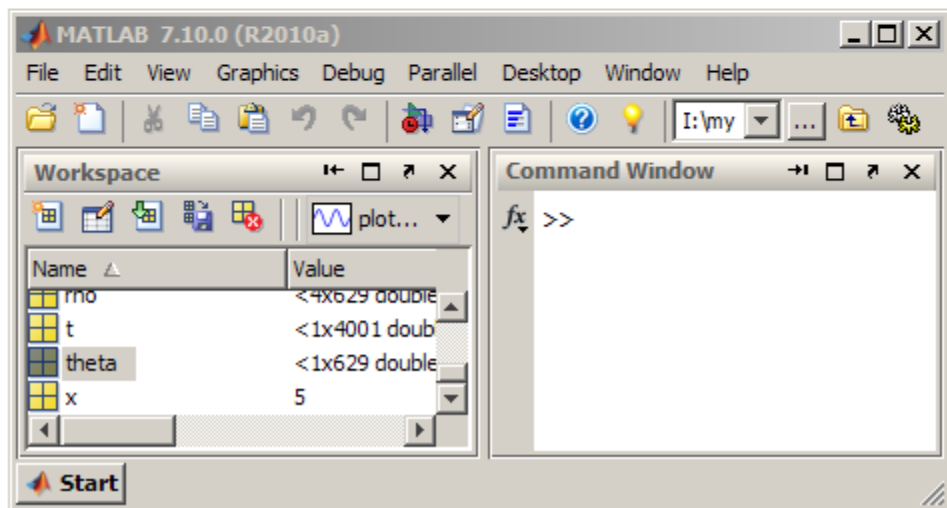
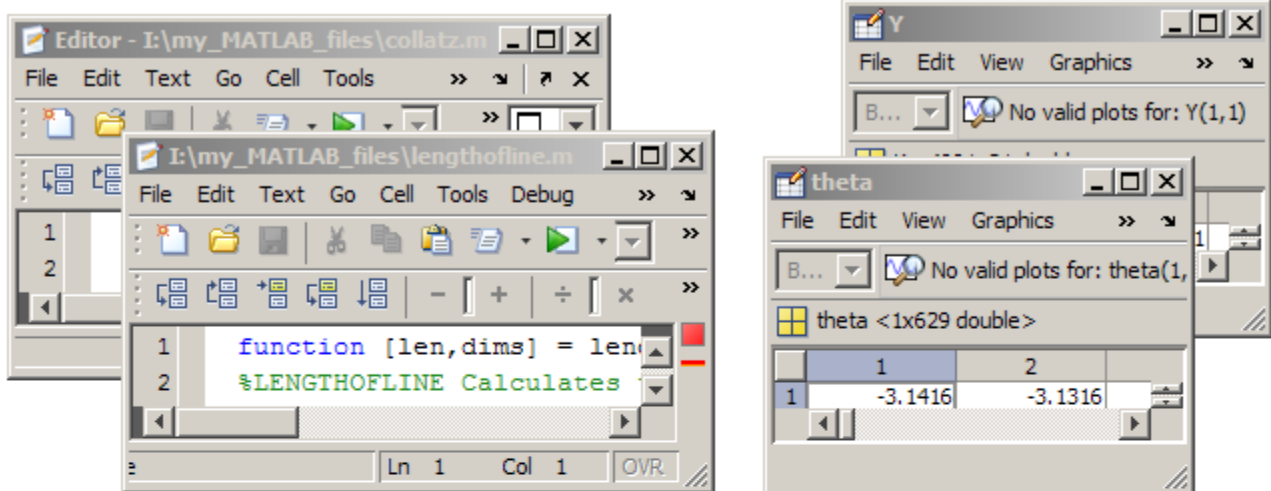
Notice the following in this example:

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

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

- Neither of the Variable Editor documents (which appear to the right of the Editor documents) includes the name of the tool.

This happens when you undock the Variable Editor from the desktop, undock the variables from the Variable Editor, and then close the empty Variable Editor window. The undocked documents in the tool remain open.



Running Frequently Used Statement Groups with MATLAB Shortcuts

In this section...

“What Is a MATLAB Shortcut?” on page 2-57

“When to Use MATLAB Shortcuts” on page 2-57

“Creating MATLAB Shortcuts — Tutorials” on page 2-58

“Running MATLAB Shortcuts” on page 2-61

“Editing and Organizing MATLAB Shortcuts” on page 2-62

“Customizing MATLAB Toolbar Shortcuts” on page 2-63

What Is a MATLAB Shortcut?

A MATLAB shortcut is an easy way to run a group of MATLAB language statements that you use regularly. Although like a MATLAB script, MATLAB does not save it as a script.

You can create shortcuts that you run from the **Start** button or the shortcut toolbar, depending on your preferences.

MATLAB maintains shortcut information in the `shortcuts.xml` file. This file is located in the folder displayed when you type `prefdir` in the Command Window. It is unlikely that you will need to access this file, because MATLAB updates it automatically.

When to Use MATLAB Shortcuts

Create MATLAB shortcuts to:

- Run a group of functions you use frequently.

For example, use a shortcut to set up your environment when you start working. This practice is useful when you do not use a `startup` file, or if there are statements you do not want to include in the `startup` file.

- Set the same properties for figures you create, such as adding a legend and setting the background color.

- Run a long statement, such as changing the current folder (`cd`) when the path names are long.
- Run a single function that you use frequently, such as `clc` to clear the Command Window.
- Run a statement that you use frequently, but have trouble remembering.

Creating MATLAB Shortcuts – Tutorials

You can create a MATLAB shortcut to run from the desktop **Start** button, or from the shortcuts toolbar, as described in the tutorials that follow. Both tutorials create a shortcut for a project called the Sea Temperature project. For these tutorials, you set up your environment in a certain way by running a series of MATLAB language statements. You create a shortcut called `sea_temp_env`, which contains these statements. Then, you run the shortcut to execute all the statements with a single click. The statements are:

```
more on
format long e
cd I:/my_MATLAB_files/sea_temp_project
clear
workspace
filebrowser
clc
```


Creating MATLAB Start Button Shortcuts

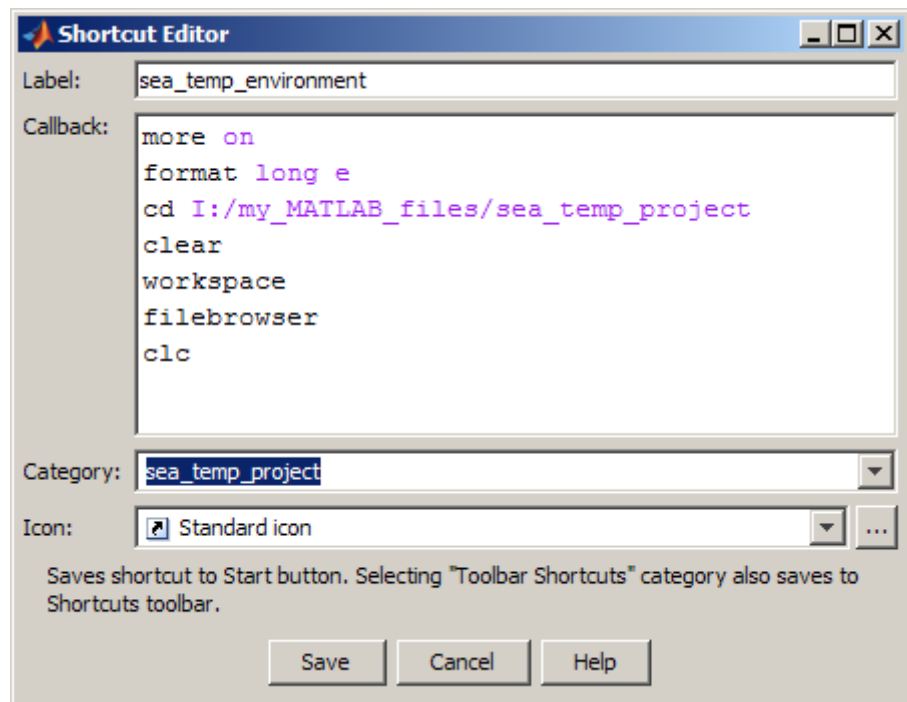
To create a start button shortcut, follow these steps:

- 1** From the **Start** button, select **Shortcuts > New Shortcut**.
- 2** Complete the Shortcut Editor dialog box.
 - a** Provide a shortcut name in the **Label** field, for example, `sea_temp_environment`.
 - b** Put the statements in the **Callback** field as shown in the following illustration. You can:
 - Enter them by typing.

- Copy and paste them from a desktop tool—if prompts (>>) from the Command Window appear, MATLAB automatically removes them when you save the shortcut.
- Drag them from a desktop tool.
- Edit them, if necessary.

The **Callback** field uses the Editor preferences for keyboard shortcuts, colors, and fonts.

- Assign a **Category**, which is like a folder for organizing shortcuts. For this example, specify `sea_temp_project`.
- Use the default shortcuts icon , or select your own.
- Click **Save**.



MATLAB adds the shortcut to the **Shortcuts** entry in the **Start** button.

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

Creating MATLAB Toolbar Shortcuts

This example assumes that you have the following statements in the Command Window:

```
more on
format long e
cd I:/my_MATLAB_files/sea_temp_project
clear
workspace
filebrowser
clc
```

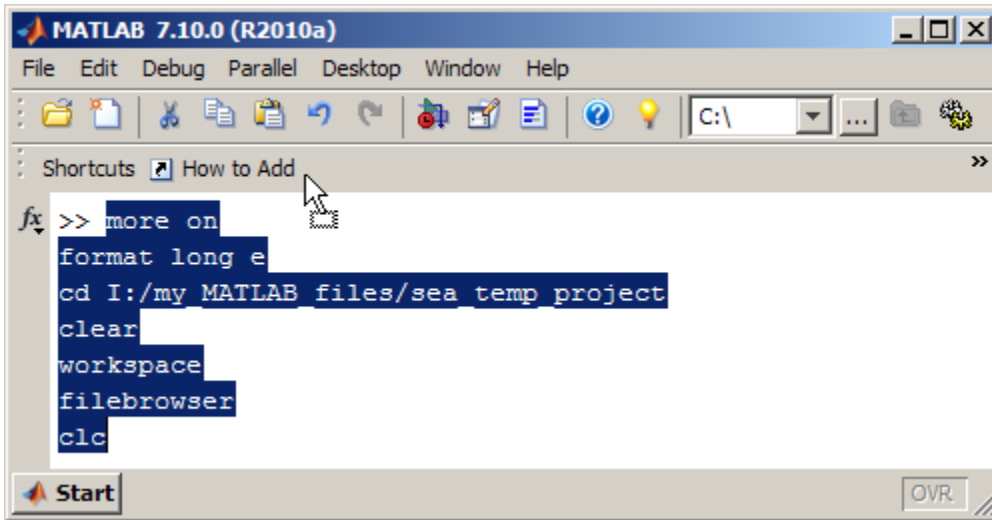
Follow these steps:

- 1** If the shortcuts toolbar is not currently on the desktop, choose **Desktop > Toolbars > Shortcuts**.
- 2** Select statements from the Command Window.

You can also select MATLAB statements from the Command History Window or a file.

- 3** Drag the selection to the desktop **Shortcuts** toolbar.

This illustration shows highlighted statements being dragged from the Command Window to the toolbar.



4 Create the shortcut by completing the Shortcut Editor dialog box:

- a In the **Label** field, enter a name for the shortcut.
- b In the **Callback** field, edit the selected statements, if necessary.

If prompts (>>) from the Command Window appear, MATLAB automatically removes them when you save the shortcut.

- c Do not change the **Category** field value, **Toolbar Shortcuts**.

For the shortcut to appear on the toolbar, this value must remain as-is.

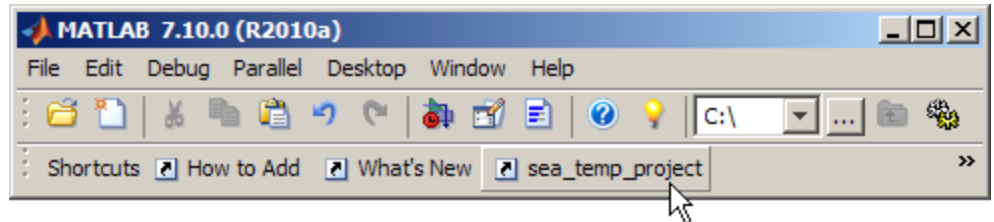
- d In the **Icon** field, select an icon, or keep the default.
- e Click **Save**.

The shortcut icon and label appear on the toolbar. If you have more shortcuts on the toolbar than the desktop can display concurrently, use the drop-down list to access them all.

Running MATLAB Shortcuts

To run a shortcut, do one of the following:

- To run a Start menu shortcut, select **Start > Shortcuts**. Then select the shortcut name, or a category submenu, followed by the shortcut name.
- To run a toolbar shortcut, click its icon on the shortcuts toolbar.



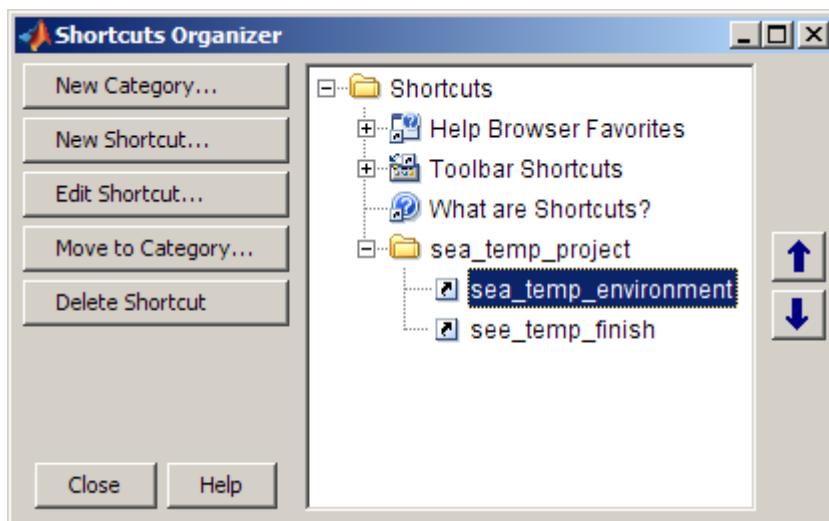
All the statements in the shortcut **Callback** field execute. It is as if you ran those statements from the Command Window, although they do not appear in the Command History window.

Editing and Organizing MATLAB Shortcuts

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

- 1 Open the Shortcuts Organizer dialog box, by doing one of the following:
 - Click the **Start** button and select **Shortcuts > Organize Shortcuts**.
 - From the shortcuts toolbar context menu, choose **Organize Shortcuts**.

The Shortcuts Organizer dialog box appears. When you select a shortcut category in the dialog box, the **Edit Shortcut** button replaces the **Rename Category** button.



2 To edit and organize shortcuts and categories, do one of the following:

- Click buttons in the dialog box.
- Right-click an item and select an action from the context menu.

Changes take effect immediately.

3 Click **Close**.

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

Customizing MATLAB Toolbar Shortcuts

For step-by-step instructions on how to add a shortcut to the toolbar, see “Creating MATLAB Toolbar Shortcuts” on page 2-60.

Default Toolbar Shortcuts

By default, the **Shortcuts** toolbar includes these two shortcuts:

- **How to Add**—Provides help about shortcuts and adding them to the **Shortcuts** toolbar.

- **What's New** —Displays the Release Notes documentation.

If you want to remove one or both of these shortcuts, see “Deleting MATLAB Shortcuts from the Toolbar” on page 2-64.

Hiding MATLAB Shortcut Labels on the Toolbar

To hide the shortcut labels on the toolbar, and leave just the shortcut icon:

- 1 Right-click in the **Shortcuts** toolbar.
- 2 From the context menu, select **Show Labels**, to clear the check mark next to the item.

Now, when you move the mouse over a shortcut icon, its label appears as a tooltip.

Displaying Hidden MATLAB Shortcut Labels on the Toolbar

To redisplay a shortcut label that you previously hid:

- 1 Right-click on the **Shortcuts** toolbar.
- 2 From the context menu, check the **Show Labels** item by selecting it. The label reappears on the toolbar.

Deleting MATLAB Shortcuts from the Toolbar

To remove a shortcut:

- 1 Right-click the toolbar shortcut button.
- 2 From the context menu. select **Delete**.
- 3 Click **OK** in the confirmation dialog box.

Alternative Ways to Create MATLAB Shortcuts

In addition to the previously described ways to create shortcuts, you can do any of the following:

- From the Command History window, select MATLAB language statements, right-click, and select **Create Shortcut** from the context menu. By default, MATLAB assigns shortcuts created from the Command History window to the **Toolbar Shortcuts** category. Shortcuts in the **Toolbar Shortcuts** category appear on the **Shortcuts** toolbar.
- Drag statements from a desktop tool, such as the Command History, onto the **Start** button.
- Right-click the **Shortcuts** toolbar, and select **New Shortcut**. Complete the resulting Shortcut Editor dialog box. If you maintain the **Toolbar Shortcuts** category, the shortcut appears on the shortcuts toolbar.

Performing Desktop Actions Using the Keyboard

Keyboard Key Combinations

As an alternative to using the mouse, you can press a combination of keyboard keys to perform some desktop actions. MATLAB supports the use of both mnemonics and keyboard shortcuts. The following topics explain the differences between these two methods and how to use them:

- “What Is a Mnemonic?” on page 2-66
- “Using Mnemonics” on page 2-66
- “What Is a Keyboard Shortcut?” on page 2-67
- “Examples of Mnemonics and a Keyboard Shortcut” on page 2-67
- “Performing Desktop Actions Using Keyboard Shortcuts” on page 2-69

What Is a Mnemonic?

A *mnemonic* is a means of using keystrokes to perform a desktop action. For instance, clicking a button or opening a menu, and then choosing an option. It is called a mnemonic because it frequently uses the first letter of the menu or menu option name. This convention helps you to remember the keystrokes required to use the mnemonic.

Mnemonics appear as underlined letters on menus or buttons. For instance, the **F** in the MATLAB **File** menu appears as shown in the following image.



Using Mnemonics

To open a menu or activate a button using mnemonics, press the **Alt** key and the letter key indicated by the underlined letter in the menu name, menu option name, or button name. The action occurs when you press the letter. You also can use mnemonics to perform an action that would require multiple mouse clicks. For example, opening the print dialog box: **Alt+F, P**. For more information see “Examples of Mnemonics and a Keyboard Shortcut” on page 2-67.

Customized keyboard shortcuts can override mnemonics. For example, if you specify **Alt+F, P** as the keyboard shortcut for the Delete action across the desktop, then pressing **Alt+F, P** no longer opens the Print dialog box. You cannot customize mnemonics.

Platform Differences.

- MATLAB running on the Apple Macintosh platform does not support mnemonics.
- The Windows operating system has a setting to hide the display of mnemonics. To display hidden mnemonics, make the MATLAB desktop the active window, and then press the **Alt** key. For details, see the Windows documentation.

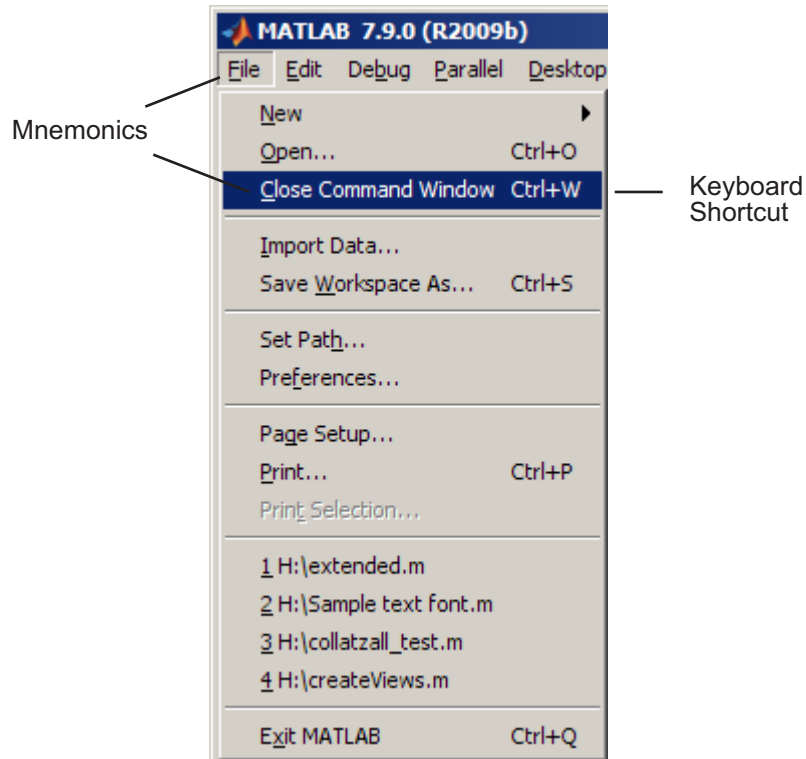
What Is a Keyboard Shortcut?

A *keyboard shortcut* is a means of using keyboard key strokes to perform a desktop action, without opening a desktop menu. For example, the default keyboard shortcut for opening a file is **Ctrl+O**. See “Overview of Keyboard Shortcuts” on page 2-69 for more information.

If you define a keyboard shortcut that uses the same keystrokes as a mnemonic, but performs a different action, then the mnemonic no longer works.

Examples of Mnemonics and a Keyboard Shortcut

The image that follows shows the desktop File menu. Notice the mnemonic and keyboard shortcut for closing the Command Window.



- **Mnemonics**

In the illustration, the underlined F in **File** on the menu bar and the underlined C in the **Close Command Window** option name indicate the mnemonics. You can close the Command Window without using the mouse by pressing **Alt+F, C**.

- **Keyboard Shortcut**

In the illustration, **Ctrl+W**, to the right of the **Close Command Window** menu option indicates the keyboard shortcut. You can close the Command Window without opening the **File** menu by pressing **Ctrl+W**.

Performing Desktop Actions Using Keyboard Shortcuts

In this section...

“Overview of Keyboard Shortcuts” on page 2-69

“Choosing a Set of Keyboard Shortcuts” on page 2-70

“Comparing Sets of Keyboard Shortcuts” on page 2-74

“Displaying Keyboard Shortcuts” on page 2-75

“Customizing Keyboard Shortcuts” on page 2-79

“Evaluating and Resolving Keyboard Shortcut Conflicts” on page 2-85

“Examples of Creating, Modifying, and Deleting Keyboard Shortcuts” on page 2-87

“Deleting a Set of Keyboard Shortcuts” on page 2-90

“Using Keyboard Shortcuts Settings Files Created on Other Systems” on page 2-91

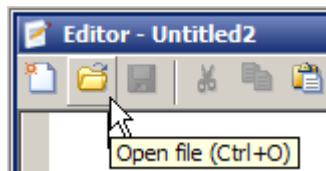
“Keyboard Shortcut Restrictions” on page 2-91

Overview of Keyboard Shortcuts

A *keyboard shortcut* is a means of using keyboard key strokes to perform a desktop action, without opening a desktop menu. If a keyboard shortcut is assigned to an action, it appears next to that action on the menu and as a tooltip for that action on the toolbar. For example, the default keyboard shortcut for opening a file is **Ctrl+O**.



A tooltip for the action also appears on the Editor toolbar.



An action can have multiple keyboard shortcuts. All defined shortcuts work, but only one appears on the desktop menu and as a toolbar tooltip.

You can:

- Choose from a set of shortcuts that install with MATLAB.
- Create customized sets of shortcuts.
- Use a set of shortcuts copied from another system

Choosing a Set of Keyboard Shortcuts

By default, MATLAB uses keyboard shortcut settings that corresponds to the platform on which you are running. To choose different keyboard shortcut settings, follow these steps:

- 1 Open the Keyboard Shortcuts Preferences dialog box by choosing **File > Preferences > Keyboard > Shortcuts**.
- 2 Click the down arrow in the **Active settings** field, and make a selection from the drop-down list, as summarized in this table.

Settings File	Option to Select	Details
Installed with MATLAB	Windows Default Set or Emacs Default Set	For a description of the files that install with MATLAB, see “Installed Settings Files for Keyboard Shortcuts” on page 2-71
You previously added	The file name	No additional information.

Settings File	Option to Select	Details
On your system, but not in the drop-down list	Browse	“Browsing to Keyboard Shortcuts Settings Files” on page 2-71
Created by someone else and uploaded to File Exchange	Search File Exchange for Downloadable Shortcut Sets	“Downloading Keyboard Shortcut Settings Files from File Exchange” on page 2-72.
MATLAB keyboard shortcuts available in Version 7.9 (R2009a) and earlier releases	Search File Exchange for Downloadable Shortcut Sets	“Downloading Keyboard Shortcut Settings Files from File Exchange” on page 2-72.

3 Click **Apply**.

Installed Settings Files for Keyboard Shortcuts

The following table lists the keyboard shortcuts settings files installed with MATLAB.

Operating System	Keyboard Shortcut Settings Files Installed with MATLAB
Windows	<ul style="list-style-type: none"> • Windows Default Set (Default) • Emacs Default Set
UNIX	<ul style="list-style-type: none"> • Emacs Default Set (Default) • Windows Default Set
Macintosh	<ul style="list-style-type: none"> • Macintosh Default Set (Default)

Browsing to Keyboard Shortcuts Settings Files

Browse to use a keyboard shortcuts settings file that is on your system, but not an **Active settings** choice in the Keyboard Shortcuts Preferences dialog box. This situation typically arises when you copy a settings file from another

system to a folder other than the `prefdir` directory. To browse to a settings file and make it your active settings file, follow these steps:

- 1** Choose **File > Preferences > Keyboard > Shortcuts**.
- 2** In the **Active settings** field, click the down arrow, and then select **Browse**.
- 3** In the Open dialog box, navigate to the folder containing the settings file.
- 4** Select the settings file, and then click **Open**.
- 5** In the Keyboard Shortcuts preferences pane, click **OK**.

The settings file you selected in step 4 is now the active settings file for MATLAB.

Future MATLAB sessions will provide this settings file as a choice in the **Active settings** drop-down menu.

Downloading Keyboard Shortcut Settings Files from File Exchange

Download keyboard shortcut settings files from File Exchange when you want to do either of the following:

- Restore the MATLAB default keyboard shortcuts that were in place for the Windows platform in Version 7.9 (R2009a) and earlier releases.
- Find and download keyboard shortcuts that others created and uploaded to File Exchange.

Follow these steps:

- 1** Choose **File > Preferences > Keyboard > Shortcuts**.
- 2** In the **Active settings** field, click the down arrow, and then select **Search File Exchange for Downloadable Shortcut Sets**.

MATLAB opens File Exchange.


- 3** Enter your MathWorks account information, and then click **Submit**.

If you do not have a MathWorks account, create one by clicking **Create an account**, and then completing the form that opens in your Web browser.

- 4 Search File Exchange for the keyboard shortcut set that you want to use.

When you follow the steps presented so far, File Exchange lists all files tagged with `keyboard shortcuts configurable` including:

- MATLAB Desktop R2009a Non-Default Keyboard Shortcut sets
- MATLAB Desktop R2009a Default Keyboard Shortcut sets

For a description, click the file name in the **File Summary** column of File Exchange. Click the Back button  to return to the list of files.


- 5 Click the download button  next to the file you want to download.
- 6 Respond to the Download MATLAB Desktop confirmation dialog boxes as follows:
 - a Click **Download**.
 - b Click **Change Current Folder to Download Location**.

The downloaded .ZIP file appears in the Current Folder Browser. Expand it to preview its contents.

- 7 In the Current Folder browser, right-click the downloaded .ZIP file, and then select **Extract**.

MATLAB creates a subfolder with the same name as the .ZIP file and extracts the files from that .ZIP file into the newly created folder.

- 8 In the Current Folder browser, expand the newly created folder, and then double-click the settings file you want to use.

A keyboard key icon  preceding a file name indicates a valid keyboard shortcut settings file.

- 9 In the Keyboard Shortcuts Preferences dialog box, review the settings, and then click **OK**.

The newly downloaded settings file is now in effect.


Comparing Sets of Keyboard Shortcuts

Compare sets of keyboard shortcuts to:

- Upgrade MATLAB from a version before Version 7.9 (R2009b).
MATLAB 7.9 made keyboard shortcuts consistent across the desktop. Therefore, you might find that shortcuts you used before Version 7.9 are different.
- See how a set of keyboard shortcuts you found on File Exchange differs from your current set of keyboard shortcuts.
- See how a set of keyboard shortcuts differs from the default set.

Steps for Comparing Keyboard Shortcuts

To compare your current set of keyboard shortcuts to another set:

- 1** Choose **File > Preferences > Keyboard > Shortcuts**.
- 2** Click the Actions button .
- 3** From the drop-down menu, choose the set of keyboard shortcuts to which you want to compare the current set.
- 4** The Comparison Tool opens and displays the two keyboard shortcut sets side-by-side.

Reading the Results of Comparing Sets of Keyboard Shortcuts

When you compare keyboard shortcut sets, they appear in the Comparison Tool as follows:

- One set displays on the left side of the tool and the other set displays on the right side of the tool.
- Each column header displays the name of the keyboard shortcut set contained within the column.
- Highlighting identifies rows that differ:
 - Rows that exist in one file, but not the other, appear in green highlighting.

- Rows that appear in both files, but that differ in content appear in pink highlighting.
- When multiple desktop tools support the same keyboard shortcut for a single desktop action, there is a row for each tool. For example, if both the MATLAB desktop and the Editor support the keyboard shortcut **Ctrl+W** for closing a selected window, a column of the Comparison Tool might appear like this:

51	Close	MATLAB Desktop	Ctrl+W	Closes the selected window
52	Close	MATLAB Editor	Ctrl+W	Closes the selected window

- When there are multiple keyboard shortcuts for the same action in a single tool, there is a row for each keyboard shortcut. For example, if there are two different keyboard shortcuts in the Editor for applying a code analyzer autofix, a column of the Comparison Tool might appear like this:

Autofix Message	MATLAB Editor	Alt+Enter	Applies the suggested autofix	11
Autofix Message	MATLAB Editor	Shift+F9	Applies the suggested autofix	12

- On Macintosh platforms, the textual format of keyboard shortcuts is slightly different from other platforms, and also differs from the representation shown on MATLAB desktop menus. These differences are due to the Macintosh platform displaying shortcuts using symbols. For instance, the Macintosh platform uses the symbol **⌘** for a keyboard key. Because the Comparison Tool represents symbols as text strings; it specifies the symbol **⌘** as CMD.

See also “Using Features of the Comparison Tool” on page 7-63.


Displaying Keyboard Shortcuts

The following sections describe the various ways you can display keyboard shortcuts:

- “Listing All Keyboard Shortcuts in a Set” on page 2-76
- “Displaying Keyboard Shortcuts on Menus” on page 2-76
- “Displaying Keyboard Shortcuts in the Preferences Dialog Box” on page 2-77

Listing All Keyboard Shortcuts in a Set

You can copy all the keyboard shortcuts from a keyboard shortcuts set and paste them in a text file or spreadsheet application, such as Microsoft® Excel®. To create a list of keyboard shortcuts for easy browsing and future reference, follow these steps:

- 1** Choose **File > Preferences > Keyboard > Shortcuts**.
- 2** Click the Actions button .
- 3** From the drop-down menu, choose **Copy to Clipboard**.
- 4** Open a spreadsheet application or a text editor.

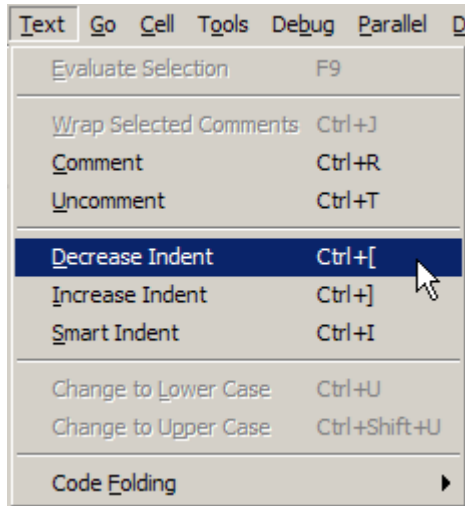
For the best formatting use a spreadsheet application.
- 5** Paste in the data from the clipboard.

In Microsoft Excel, for example, choose **Home > Paste**.

Displaying Keyboard Shortcuts on Menus

Open the menu to see if the keyboard shortcut appears next to the menu option.

For example, suppose you want to determine the keyboard shortcut for decreasing the indent in the Editor. Open the **Text** menu, and then view the keyboard shortcut for **Decrease Indent**. The keyboard shortcut **Ctrl+[** appears to the right of the option.

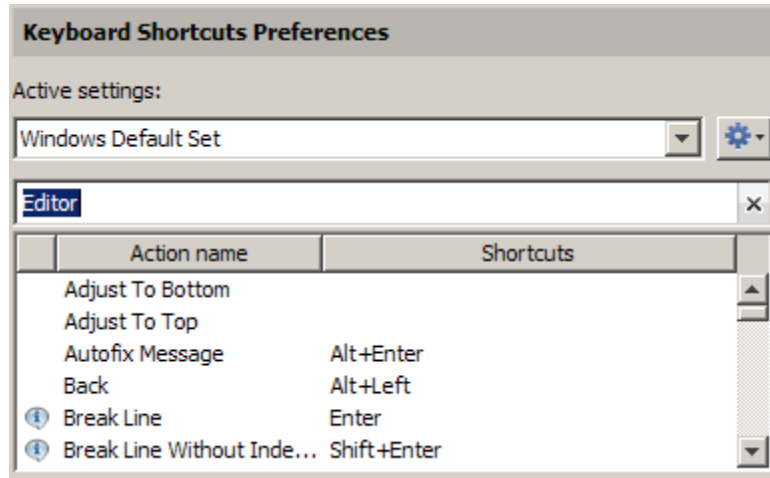


If no keyboard shortcut appears on the menu, one does not currently exist for that action. To create a keyboard shortcut for an action, follow the steps in “Customizing Keyboard Shortcuts” on page 2-79.

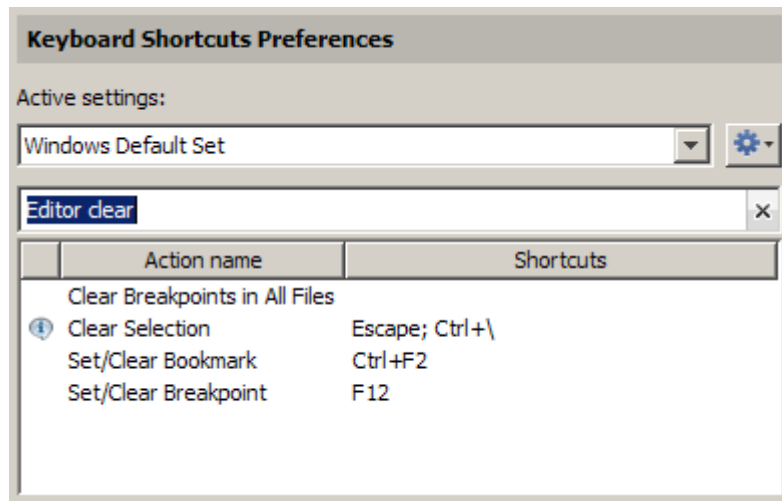
Displaying Keyboard Shortcuts in the Preferences Dialog Box

To identify a keyboard shortcut when there is no menu option for an action, use the **Keyboard Shortcuts Preferences** pane:

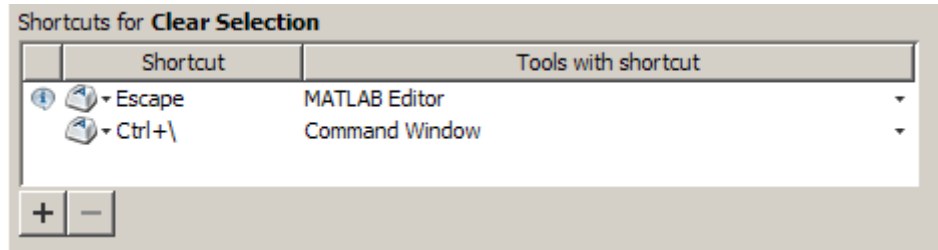
- 1 Choose **File > Preferences > Keyboard > Shortcuts**.
- 2 In the filter field, type the name of the tool for which you want to list the keyboard shortcuts. For example, type **Editor** to see the keyboard shortcuts currently defined for actions you can perform in the Editor.



- 3 Narrow the list of **Action names** that the preferences pane displays by adding a string describing the action. For example, add `clear`, if you want to find the keyboard shortcut for clearing selected text in the Editor. Type a short string to increase the likelihood of the filter returning the action you seek.



- 4 Select the action name of interest. In this example, select **Clear Selection**.
- 5 View the table labeled **Shortcuts for Clear Selection**. It indicates that the **Escape** key is the current keyboard shortcut for the **Clear Selection** action in the Editor.



Customizing Keyboard Shortcuts

To customize or view keyboard shortcuts for MATLAB desktop tools, choose **File > Preferences > Keyboard > Shortcuts**. If you have an active Internet connection, you can watch the Customizable Keyboard Shortcuts video for an overview.

The following sections provide details:

- “Steps for Customizing Keyboard Shortcuts” on page 2-80
- “Filtering Keyboard Shortcut Actions” on page 2-83
- “Specifying Keystrokes for a Keyboard Shortcut” on page 2-84
- “Evaluating and Resolving Keyboard Shortcut Conflicts” on page 2-85
- “Examples of Creating, Modifying, and Deleting Keyboard Shortcuts” on page 2-87
- “Keyboard Key Combinations” on page 2-66
- “Displaying Keyboard Shortcuts” on page 2-75

Consider using File Exchange to share your active settings file with others. For more information, see “Submitting Your Files to the Repository” on page 8-41.

Steps for Customizing Keyboard Shortcuts

1 Choose **File > Preferences > Keyboard > Shortcuts**.

2 In the **Active settings** field, choose the file that contains the set of keyboard shortcuts that you want to customize.

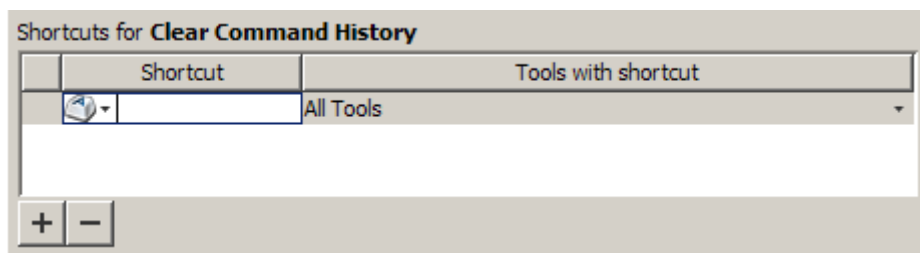
Typically, the first time you modify keyboard shortcuts, you begin with the default settings for your platform. For details, see “Choosing a Set of Keyboard Shortcuts” on page 2-70.

3 Under **Action name**, select the action for which you want to define or modify a keyboard shortcut. An action is the operation for which you want to customize the shortcut, such as **Clear Command History**.

For tips on finding the action you want, see “Filtering Keyboard Shortcut Actions” on page 2-83.

4 Click the Add button .


An editable field opens under the **Shortcut** column.



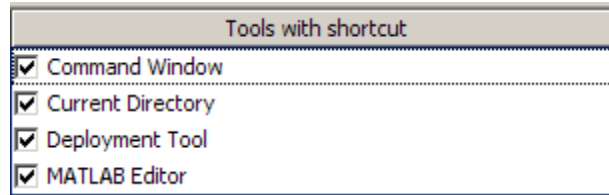
5 Type the shortcut that you want to use for the action you selected in Step 3. Alternatively, you can choose a shortcut from the drop-down menu.



For details, see “Specifying Keystrokes for a Keyboard Shortcut” on page 2-84.

6 Assign the shortcut to the tool or tools with which you want to use it. For example, in the **Tools with shortcut** column:

- a Click the down arrow  for the list of desktop tools to which you can assign a shortcut. Not all actions are available with all desktop tools.

- b** Select a check box to assign the shortcut to a tool. Clear a check box to remove it.



- 7** Evaluate and resolve any conflicts, indicated by the informational  and error  icons.


For more information, see “Evaluating and Resolving Keyboard Shortcut Conflicts” on page 2-85.

- 8** Click **Apply**.

- The keyboard shortcut becomes available immediately.
- If a changed shortcut corresponds to a menu option that previously displayed no keyboard shortcut, MATLAB reflects the new keyboard shortcut on the menu.

Restoring Default Keyboard Shortcut Sets

If you modify keyboard shortcuts, and then decide you do not want to keep the changes, you can restore the default shortcuts. To restore the default state of a keyboard shortcut:

- 1** Click the **Actions** button .
- 2** Select **Undo Modifications to Windows Default Set (modified)** or **Undo Modifications to Emacs Default Set (modified)**, as appropriate for your system.
- 3** Click **OK**.

Note Undoing modifications reverts all keyboard shortcuts changes that you made to the set. You cannot undo modifications on a shortcut-by-shortcut basis.

Saving Keyboard Shortcuts to a Settings File

Save keyboard shortcuts to a settings file to:

- Save changes you make to a default settings file, such as the `Windows default` set, to a new set.

MATLAB preserves changes you make to the default sets across sessions. However, if you undo modifications to a default keyboard shortcut set (as described in “Restoring Default Keyboard Shortcut Sets” on page 2-81) you lose all changes, unless you first save them to a new set.


- Copy the keyboard shortcuts settings file to another system running MATLAB and use it there.
- Overwrite a settings file that you previously saved.

You cannot overwrite the default settings files that install with MATLAB. MATLAB saves modifications that you make to a default set using the name of the default set appended with the text `(modified)`. For instance, `Windows default (modified)`.

- Share a keyboard shortcuts settings file with others.

For information on sharing your active settings file with the MATLAB community, see “Submitting Your Files to the Repository” on page 8-41.

To save a keyboard shortcuts settings file, follow these steps:

- 1** Open the Keyboard Shortcuts Preferences dialog box by choosing **File > Preferences > Keyboard > Shortcuts**.
- 2** Click the **Actions** button , and then select **Save As**.
- 3** In the Save dialog box, navigate to the folder where you want to save the file, specify the file name, and then click **Save**.

MATLAB saves the file as an `.xml` file in the folder that you specified.

Filtering Keyboard Shortcut Actions

Use the filter field to see the list of actions for which you can customize or define a keyboard shortcut:

1 Type all or part of any one of the following:

- An action name, for example, **Delete**.

MATLAB displays only the action names or desktop menus that contain the text you specify.

- The name of a desktop tool or menu, for example, **File** or **Command Window**.

MATLAB displays a list of the action names associated with the tool or menu you specify. In addition, the list includes any action names that contain the name of the tool or menu. For example, if you specify **Command History**, the list of action names includes **Next History Command**, which is a **Command Window** action.

- A keyboard shortcut, for example, **Ctrl+R**

MATLAB displays only the action names that have the shortcut you specify. Be aware of the following:


- You can enter most keyboard shortcuts by either pressing keystrokes or typing the key names.

For example, to enter **Ctrl+S**, use the keystroke (by pressing the **Ctrl** key and the **S** key). Or, type **Ctrl+S** character by character (**C-t-r-l-+-Y**).

- If using keystrokes for a keyboard shortcut does not work, try typing the characters instead. You *must* type some keyboard shortcuts character by character, such as shortcuts including the **Tab**, **Backspace**, or **Delete** keys.
- Type **numpad** to refer to the number pad that is on the far right of some keyboards.
- Type **Up** or **Down** to refer to the **Up arrow** or **Down arrow** keypad keys, respectively.

2 Verify that an **Action name** performs the action you expect:



- a Hover the mouse pointer over the **Action name**. For example, **Remove Next Word**.
- b View the tooltip that appears.

Action name	Shortcuts
Cut	Ctrl+X; Shift+Delete
Delete	Delete
 Delete Previous Character	Backspace; Shift+Backspace
Delete Selected Rows/Columns	Ctrl+Minus
Delete to Selection	
Remove Next Word	Ctrl+Delete

Deletes the next word


Specifying Keystrokes for a Keyboard Shortcut

A *keystroke* can be a single key or the combination of a modifier (**Alt**, **Shift**, or **Ctrl**) and another key. When you create a keyboard shortcut, specify the keystrokes for the shortcut as follows:

- 1 Click the **Add** button .
- 2 Specify the number of keystrokes you want to use for the shortcut:
 - To use the default number of keystrokes, which is one keystroke, skip to step 3.
 - To specify multiple keystrokes, or to specify explicitly one keystroke follow these steps:
 - a Click the down arrow next to the key icon  in the **Shortcuts** field.
 - b Choose **Limit to 1 keystroke**, **Limit to 2 keystrokes**, or **Limit to 3 keystrokes**.

For instance, **Ctrl+F** is one keystroke, **Ctrl+Y**, **Shift+Z** is two keystrokes, and **Ctrl+Y**, **Shift+Z**, **F9** is three keystrokes.



- 3 Specify the keystrokes by doing one of the following:



- Type the keystrokes, by pressing the keys, *not* by typing the key names character by character.
For example, press the **Ctrl** key and the **Y** key. Do not type **C-t-r-l+-Y**.
- Choose a keystroke, such as the **Tab** key, by clicking the down arrow next to the key icon  in the **Shortcuts** field. Then, choose the key name.
The listed keys already have a defined action within dialog boxes. For example, the **Tab** key navigates from one field to the next in dialog boxes.

Evaluating and Resolving Keyboard Shortcut Conflicts


Conflicts arise when two or more different actions have the same shortcut. There is no requirement that you resolve keyboard shortcut conflicts. However, if the same shortcut specifies two different actions, the shortcuts can be confusing to use.

View keyboard shortcut conflicts by choosing **File > Preferences > Keyboard > Shortcuts**.

The Keyboard Shortcuts preferences pane indicates conflicts using informational  and error  icons.

-  —An informational icon indicates that two different actions in two different tools have the same shortcut. For information on resolving these conflicts, see “Actions in Different Tools Have the Same Shortcut — Evaluating Conflicts” on page 2-85.
-  —An error icon indicates that two different actions within the same tool have the same shortcut. For information on resolving these conflicts, see “Actions in the Same Tool Have the Same Shortcut — Evaluating Conflicts” on page 2-86.

Actions in Different Tools Have the Same Shortcut — Evaluating Conflicts


Typically, you want to resolve conflicts indicated by the informational icon  when all the following are true:

- You use both tools frequently.
- You perform both actions frequently.

- You have difficulty remembering the action that the shortcut performs in each tool.

For instance on Microsoft Windows platforms, by default, **Ctrl+Shift+U** undocks a tool from the MATLAB desktop. However if you select text in the Editor, and then press **Ctrl+Shift+U**, it changes the selected text to uppercase. If you frequently use both of these actions, you can specify a different keyboard shortcut for one or both actions.

Actions in the Same Tool Have the Same Shortcut – Evaluating Conflicts

Typically, you want to resolve conflicts indicated by the error icon .

It can be *unnecessary* to resolve these conflicts if one or more of the following are true:

- The situation is temporary.
For instance, you are performing a two-step procedure. In the first step, you assign the keyboard shortcut to an action that results in a conflict. Then, in the second step, you remove the shortcut from the original action.
- The two actions are associated with different modes of the same tool.

By default, when the MATLAB Editor is in cell mode, **Ctrl+Up** and **Ctrl+Down** move the cursor to the Next and Previous cell, respectively. When the Editor is not in cell mode, those keyboard shortcuts scroll up and scroll down, respectively. The shortcuts are in conflict, but the behavior probably is expected, for the given MATLAB Editor mode.

Although not evident from the preferences pane, **Ctrl+C** presents a similar situation on Windows systems. **Ctrl+C** is the keyboard shortcut for interrupting MATLAB execution. However, the default keyboard shortcut for the copy action is also **Ctrl+C**. Therefore, if you:

- Select an item, and then press **Ctrl+C**, it copies the selected item to the clipboard, — regardless of whether MATLAB is busy.
- Do not select an item and press **Ctrl+C**, it interrupts MATLAB execution.

If you change the default keyboard shortcut for the copy action from **Ctrl+C** to another keystroke, then **Ctrl+C** interrupts MATLAB execution, regardless of whether you have selected an item.

Resolving Keyboard Shortcut Conflicts


To resolve a conflict, change or delete shortcuts such that there is a one-to-one correspondence between a shortcut and a frequently used action. For examples, see “Changing a Keyboard Shortcut” on page 2-88 and “Deleting a Keyboard Shortcut” on page 2-89.

Examples of Creating, Modifying, and Deleting Keyboard Shortcuts

- “Creating a New Keyboard Shortcut” on page 2-87
- “Changing a Keyboard Shortcut” on page 2-88
- “Deleting a Keyboard Shortcut” on page 2-89

Creating a New Keyboard Shortcut

By default, no keyboard shortcut is available for adding a Help topic to the list of favorites. If you frequently mark topics as favorites, you can define a keyboard shortcut for this action, as follows:

- 1** Choose **File > Preferences > Keyboard > Shortcuts**.
- 2** In the filter field, type **Help**.
- 3** Scroll through the **Action name** list, and select **Add to Favorites**.
- 4** Click the plus button 

MATLAB adds a row to the table above the plus button.

- 5** In the **Shortcut** field, click the down arrow, and then change **Limit to 1** keystroke to **Limit to 2** keystrokes.
- 6** In the **Shortcut** field, press **Ctrl+S**, and then **Alt+V**.

Notice that the All Possible Conflicts table is empty, which indicates that no other desktop action is currently using this combination of keystrokes.

7 Click **Apply**.

Notice that:

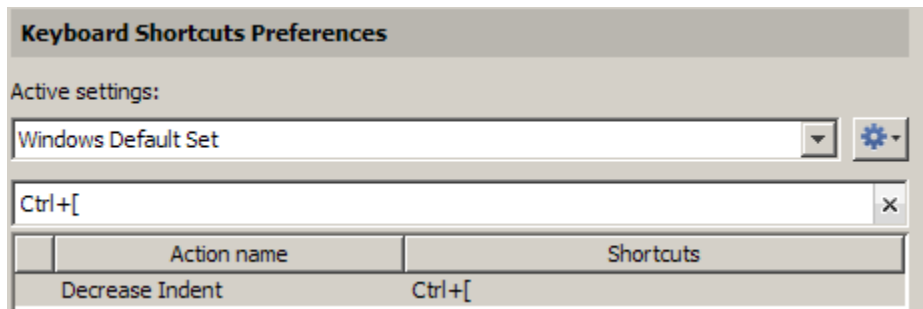
- The Add to Favorites dialog box opens when you press **Ctrl+S**, **Alt+V** in the Help browser.
- **Ctrl+S**, **Alt+V** appears next to **Add to Favorites** when you click the **Favorites** menu in the Help browser.

Changing a Keyboard Shortcut

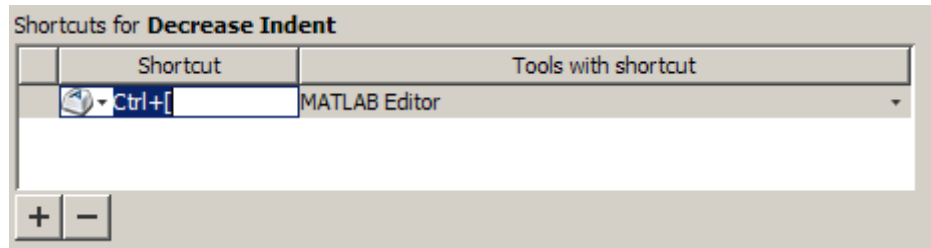
Suppose you frequently adjust indenting in the MATLAB Editor. However, you have difficulty remembering the default keyboard shortcut of **Ctrl+[** for decreasing the indent. So, you decide to change it to something that is easier to remember.

This example changes the keyboard shortcut for **Decrease Indent** in the MATLAB Editor from **Ctrl+[** to **Ctrl+Backspace**:

- 1** Choose **File > Preferences > Keyboard > Shortcuts**.
- 2** Under **Active settings**, choose Windows Default Set.
- 3** In the filter field, press **Ctrl+[**.
- 4** Under **Action name**, select **Decrease Indent**.



- 5** In the table labeled **Shortcuts for Decrease Indent**, under **Shortcut**, click **Ctrl+[**. MATLAB makes the field editable.



- 6** In the **Shortcut** field, press **Ctrl+Backspace** twice.

The first time you press the key combination, it deletes **Ctrl+[**. The second time you press it, **Ctrl+Backspace** appears in the field.

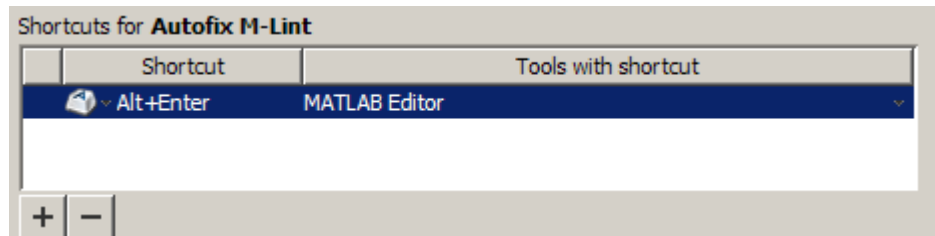
- 7** Click **Apply**.


MATLAB saves your changes to the Windows Default Set (modified) settings.

Deleting a Keyboard Shortcut

Suppose you find yourself frequently pressing the wrong keyboard shortcut. For example, on Windows, you press **Alt+Enter** (to apply a code analyzer autofix) instead of **Ctrl+Enter** (to evaluate the current cell in the MATLAB Editor). To avoid accidentally applying an autofix, delete the **Alt+Enter** shortcut by following these steps:

- 1** Choose **File > Preferences > Keyboard > Shortcuts**.
- 2** Under **Active settings**, choose Windows Default Set or Windows Default Set (modified).
- 3** In the filter field, press **Alt+Enter**.
- 4** Under **Action name**, select the row containing **Autofix Message**.
- 5** In the next table, under **Shortcut**, select the row containing **Alt+Enter**.



6 Click the remove button .

7 Click **Apply**.

If it does not exist, MATLAB creates a `Windows Default Set (modified)` keyboard shortcut set. This set consists of the `Windows Default Set` of keyboard shortcuts, less the shortcut for **Alt+Enter**. If the `Windows Default Set (modified)` settings file exists, then MATLAB deletes the **Alt+Enter** keyboard shortcut from that set of keyboard shortcuts.

See also “Deleting a Set of Keyboard Shortcuts” on page 2-90.


Deleting a Set of Keyboard Shortcuts

If you previously saved or copied a set of keyboard shortcuts to your system and you no longer want it, delete it as follows:

1 Choose **File > Preferences > Keyboard > Shortcuts**.

2 Under **Active settings**, choose the set of keyboard shortcuts that you want to delete.

You cannot delete default keyboard shortcut sets, such as `Windows Default Set`.

3 Click the Actions button  and choose `Delete filename`, where *filename* is the name of a keyboard shortcut set you previously saved or copied to your system.

For information on deleting a single keyboard shortcut from a set that you want to keep, see “Deleting a Keyboard Shortcut” on page 2-89.

Using Keyboard Shortcuts Settings Files Created on Other Systems

If you find a keyboard shortcuts settings file that is useful to you, or if you want to use one you created on a different system, make it the active settings file as follows:

- 1 Copy the settings file to a folder on your system, such as:

```
I:\my_matlab_files\active_settings_files\new_settings.xml
```

- 2 Choose **File > Preferences > Keyboard > Shortcuts**.
- 3 In the **Active settings** field, click the down arrow, and then click **Browse**.
- 4 In the Open dialog box, navigate to the folder where you copied the settings file.
- 5 Select the settings file, and then click **Open**.
- 6 In the Keyboard Shortcuts preferences pane, click **Apply**. The settings file you specified is now the active settings file for MATLAB.

Consider using File Exchange to share your active settings file with others, or to find a file that is useful to you. For more information on File Exchange, see Chapter 8, “File Exchange — Finding and Getting Files Created by Other Users”.

Keyboard Shortcut Restrictions

These sections describe the tools, portions of tools, and actions for which you cannot change keyboard shortcuts:

- “Tools for Which You Cannot Customize Keyboard Shortcuts” on page 2-91
- “Actions for Which You Cannot Customize Keyboard Shortcuts” on page 2-92

Tools for Which You Cannot Customize Keyboard Shortcuts

You cannot change the keyboard shortcuts associated with the following tools or portions of tools:

- **Figure windows**—For example, you cannot modify the keyboard shortcut, **Ctrl+S**, for saving a MATLAB `.fig` file.
- **Toolboxes**—For example, you cannot modify keyboard shortcuts in the SimBiology[®] desktop.
- **Incremental search**—You can modify the keyboard shortcuts for initiating a forward or backward incremental search. However, you cannot change the keyboard shortcuts that you use within incremental search mode, such as **Ctrl+Shift+S** to search forward.
- **Dialog boxes**—For example, you cannot create a keyboard shortcut for the **OK** button.

Actions for Which You Cannot Customize Keyboard Shortcuts

The following table describes some frequently used actions for which you cannot customize keyboard shortcuts.

Action	Keyboard Shortcut
Cancel the current action.	<p>Esc (escape)</p> <p>For example, if you select the Edit menu, the menu items display. Pressing Esc retracts the menu items.</p> <p>In the Function Browser, pressing Esc up to three times has the following effects:</p> <ol style="list-style-type: none"> 1 Dismisses the search history 2 Clears the search field 3 Closes the Function Browser
Interrupt MATLAB execution on all supported platforms.	Ctrl+C
Interrupt MATLAB execution on Windows and UNIX systems.	Ctrl+Cancel
Interrupt MATLAB execution on Macintosh systems.	Cmd+. (period)

Action	Keyboard Shortcut
Open context menu on Windows and UNIX systems.	Ctrl+Shift+F10
Close the desktop and consequently shut down the MATLAB program. Outside the desktop, close the active window (except on Macintosh platforms).	Alt+F4
Accessibility affordances	Tab for navigating through fields in dialog boxes, for example.
Make an open tool the active tool	<p>These shortcuts appear on the desktop Windows menu:</p> <ul style="list-style-type: none"> • Command Window: Ctrl+0 • Command History: Ctrl+1 • Current Folder: Ctrl+2 • Editor: Ctrl+Shift+0 • Figures: Ctrl+Shift+1 • Figure Palette: Ctrl+7 • Comparison Tool: Ctrl+Shift+4 • File Exchange: Ctrl+6 • Help: Ctrl+4 • Plot Browser: Ctrl+8 • Profiler: Ctrl+5 • Variable Editor: Ctrl+Shift+3 • Web Browser: Ctrl+Shift+2 • Workspace: Ctrl+3

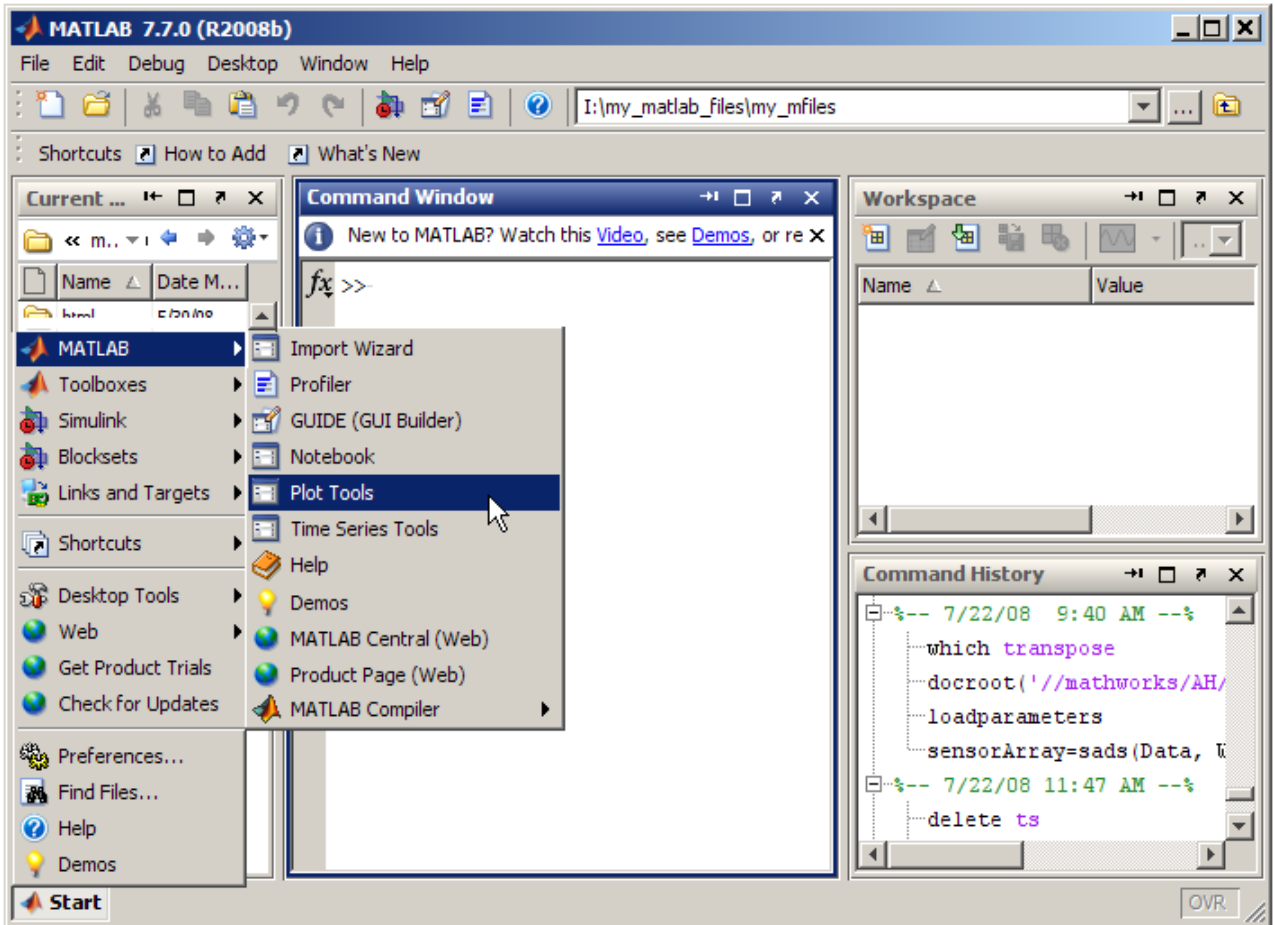
Accessing Tools with the Start Button

In this section...
“Viewing Products and Tools with the Start Button” on page 2-94
“Adding Your Own Toolboxes to the Start Button” on page 2-96

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

Viewing Products and Tools with the Start Button

- 1 Click the **Start** button to view a menu of product categories and desktop tools installed on your system. As an alternative, press **Alt+S** (except on Apple Macintosh platforms). In the following illustration, the **Start** button shows MATLAB selected.








- From the menu and submenu items, select an item to open. The icons help you quickly locate the item you want—see the icon descriptions in the following table.

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

Identifying Start Button Menu Icons

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

Icon	Description of Action When Opened
	Documentation for that product opens in the Help browser.
	A list of demos for the product appears in the Help browser Demos pane.
	Selected tool opens.
	Block library opens.
	Document opens in your system Web browser.

Adding Your Own Toolboxes to the Start Button

If you author a collection of MATLAB program files or Simulink blocks, you can provide access to its features via the **Start** button. A collection of MATLAB program files is called a *toolbox*. A collection of Simulink blocks is called a *blockset*. (For background information about the **Start** button, see “Accessing Tools with the Start Button” on page 2-94.)

MATLAB determines the information to display on the **Start** button using `info.xml` files that are in folders on the search path. These XML files also contain information to place entries for toolbox and blockset documentation in the **Contents** pane of the Help browser. To add your toolbox to the Start button:

- 1 Make your current folder the folder containing your toolbox files.
- 2 Copy the template file `info_template.xml` to one you name `info.xml` in your current folder. If you have set up HTML files for your toolbox, you have already created an `info.xml` file for it. See “More About Adding Items to the Start Button” on page 2-97 for details.
- 3 Add a `<list>` `</list>` element as the last element within `<productinfo>`. Put the `<list>` after the `<help_location>` element, if any.

Within the `<list>`, add a `<listitem>` for each item you add to the **Start** menu for the toolbox. For example:

```
<list>
  <listitem>
    <!-- The label provides the text for this menu item -->
    <label>MyToolbox Documentation</label>
    <!-- This callback is a command to open your documentation -->
    <callback>web ./helpfiles/mytoolbox_product_page.html -helpbrowser</callback>
    <!-- Menu item icon (a toolbox icon from the help browser ) -->
    <icon>$toolbox/matlab/icons/bookicon.gif</icon>
  </listitem>
  <listitem>
    ...
  </listitem>
  ...
</list>
```

Each `<callback>` element contains a complete MATLAB command specifying the action triggered by selecting that menu item. `<icon>` elements identify **Start** menu item icons. For a table of standard icons that you can use, see “More About the helptoc.xml File” on page 5-32.

- 4 Include the `info.xml` file in your toolbox folder, along with your toolbox program files and documentation files.
- 5 Add the toolbox folder to the search path. Refresh the **Start** button, if a **Toolboxes** item does not appear.
- 6 If you are providing the toolbox to others, instruct them to perform the previous step themselves after they start MATLAB .

If you also include HTML documentation for your toolbox for display in the Help browser, put the Help browser and **Start** button XML code into a single `info.xml` file.

More About Adding Items to the Start Button

The template XML file called `info_template.xml` includes code to add menu items for a toolbox to the **Start** menu. This directory is `matlabroot/help/techdoc/matlab_env/examples/templates`. `<icon>`

elements in that file specify icon image files to use in the **Start** menu. You can use an example icon image file, `sampleicon.gif`, contained in the `/examples/templates` folder. To display a custom icon, substitute your own image file name for `sampleicon.gif` in your `info.xml` file. Put your icon file in the same folder that `info.xml` occupies.

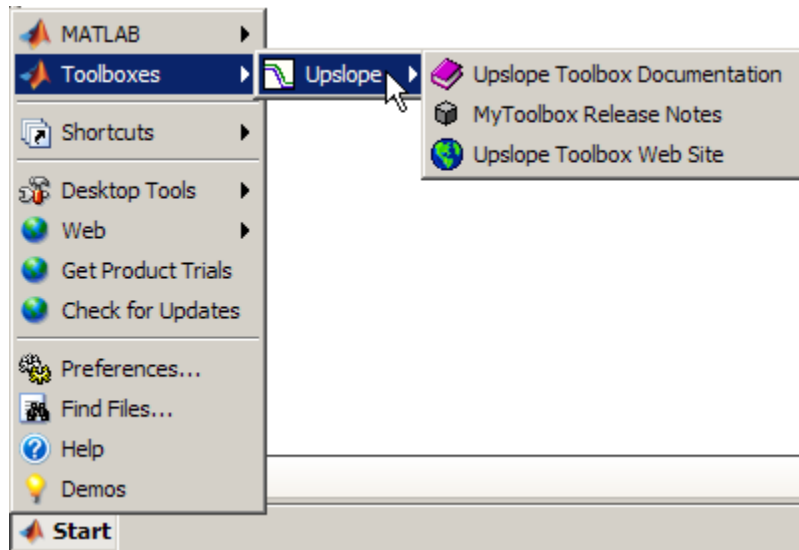
For instructions on copying and using the sample `info_template.xml` file, see “Identifying a Help Folder: the `info.xml` File” on page 5-24. This file contains a `<list>` element that you can modify to customize what appears on the **Start** button for your toolbox. You can remove `<list>` elements or include additional ones.

When you have created a `info.xml` file in your toolbox folder containing `<list>` elements for your **Start** menu items (or have copied the Upslope Toolbox example files, which include a complete `info.xml` file):

- 1** Add the folder containing `info.xml` to the search path. The folder cannot be the current folder when you add it to the path.

See “Adding Folders to the Search Path” on page 7-75.

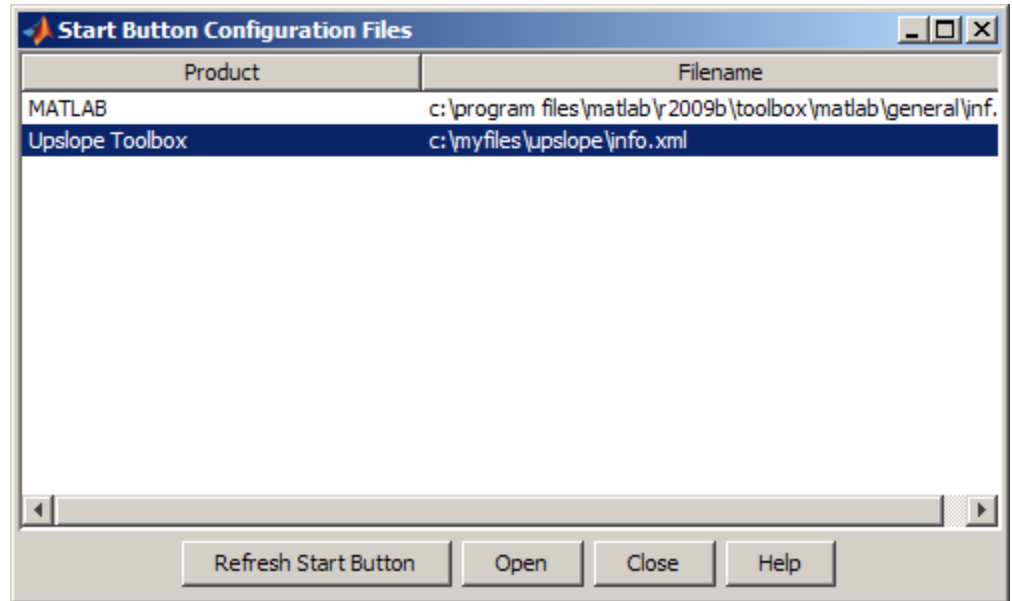
- 2** Click the **Start** button and then click the Toolboxes item. If you set up everything properly, you see an entry your toolbox, similar to the following illustration.



If you do not see your toolbox on the **Start** button, you might need to refresh the **Start** button so it can locate your `info.xml` file. Try the following:

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

The Start Button Configuration Files dialog box opens.



- b** Click **Refresh Start Button**.
 - c** Click **Close** to close the dialog box.
- 3** Ensure there are no errors. If, when interpreting your `info.xml` file, MATLAB detects an invalid construct, it displays an error message in the Command Window. For more information, see “Addressing Validation Errors for `info.xml` Files” on page 5-61 and “Addressing Validation Errors for `info.xml` Files” on page 5-61,

For more information about setting up an `info.xml` file, see “Adding HTML Help Files to the Help Browser” on page 5-17.

Using Web Browsers in MATLAB

In this section...
“About Web Browsers in MATLAB” on page 2-101
“Displaying Pages in Web Browsers” on page 2-103
“Web Preferences” on page 2-104

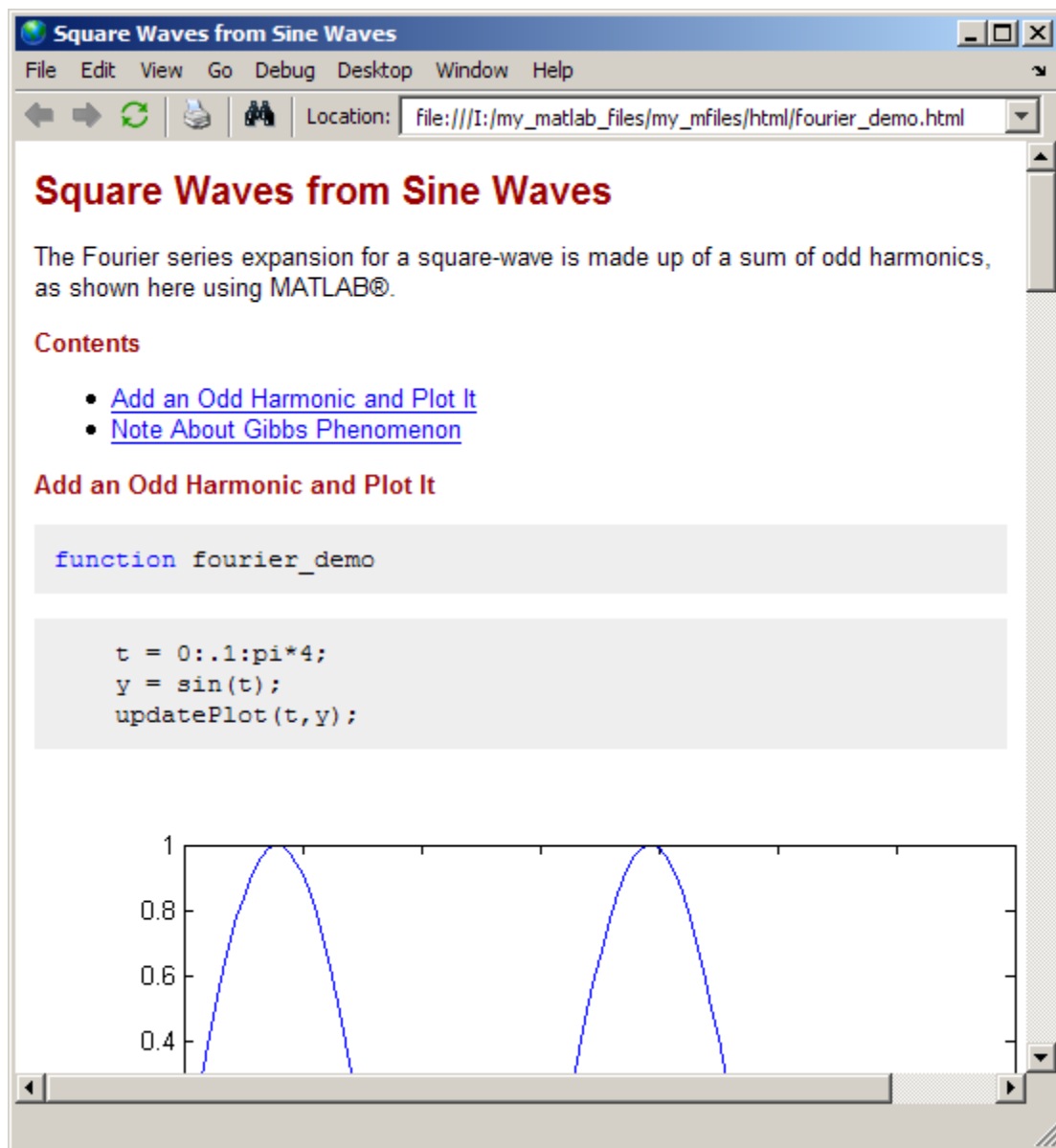
About Web Browsers in MATLAB

From MATLAB, Web sites and documents can display in any of the following browsers:

- MATLAB Web browser
- Help browser
- Your system Web browser, such as Mozilla® Firefox®

MATLAB uses the different browsers to display different types of information:

- Web sites display in your system browser.
- Documentation and demo pages display in the Help browser.
- Other HTML files display in the MATLAB Web Browser. For example, after publishing a MATLAB program file to HTML, the HTML file displays in the MATLAB Web Browser:



Square Waves from Sine Waves

The Fourier series expansion for a square-wave is made up of a sum of odd harmonics, as shown here using MATLAB®.

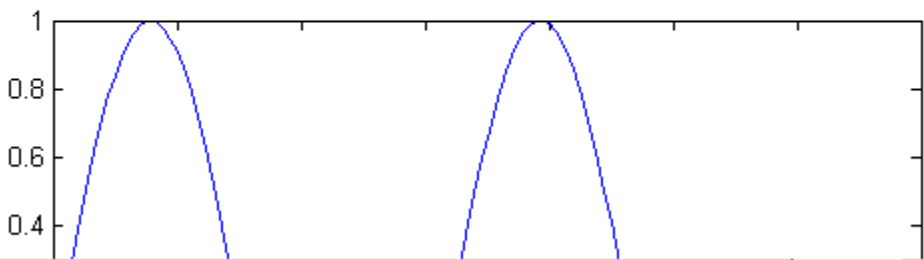
Contents

- [Add an Odd Harmonic and Plot It](#)
- [Note About Gibbs Phenomenon](#)

Add an Odd Harmonic and Plot It

```
function fourier_demo
```

```
t = 0:.1:pi*4;  
y = sin(t);  
updatePlot(t,y);
```



MATLAB Web and Help Browsers

Because the MATLAB Web and Help browsers are desktop tools, you can perform desktop operations on them, such as docking the tools in the desktop. For more information, see Chapter 2, “Desktop”.

The MATLAB Web and Help browsers may not support all the features that a particular Web site or HTML page uses. For example, the MATLAB Web Browser does not display `.bmp` (bitmap) image files. Instead use `.gif` or `.jpeg` formats for image files in HTML pages.

System Browser

The system browser that MATLAB uses depends on your platform:

- On Microsoft Windows and Apple Macintosh platforms, MATLAB uses the default browser for your operating system.
- On UNIX platforms, MATLAB uses the Mozilla Firefox browser. You can specify a different system browser for MATLAB using Web preferences.

Displaying Pages in Web Browsers

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

To display a Web page or any file type in the MATLAB Web Browser:

- 1** Select **Desktop > Web Browser**.

An empty MATLAB Web browser window opens.

- 2** Type a URL or full path to a filename in the **Location** field.

To open a link to another page in a separate MATLAB Web browser window, click the middle mouse button, if you have one.

To open any type of document in any type of Web browser MATLAB supports, use the `web` function. Specify a URL or file to display, and the type of browser to use.

Web Preferences

Use Web preferences to specify characteristics related to accessing the Internet:

- “Specifying Proxy Server Settings” on page 2-104
- “Specifying the System Browser for UNIX Platforms” on page 2-106
- “Specifying Fonts for the MATLAB Web Browser” on page 2-107

Specifying Proxy Server Settings

If your network uses a firewall or another method of protection that restricts Internet access, provide information about your proxy server to MATLAB.

To specify the proxy server settings:

- 1** Select **File > Preferences > Web**.
- 2** Select the **Use a proxy server to connect to the Internet** check box:
- 3** Specify values for **Proxy host** and **Proxy port**. Examples of acceptable formats for the host are: 172.16.10.8 and ourproxy. For the port, enter only an integer, such as 22. If you do not know the values for your proxy server, ask your system or network administrator for the information.

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

Note MATLAB stores the password without encryption in your `matlab.prf` file.

- 4** Ensure that your settings work by clicking the **Test connection** button.

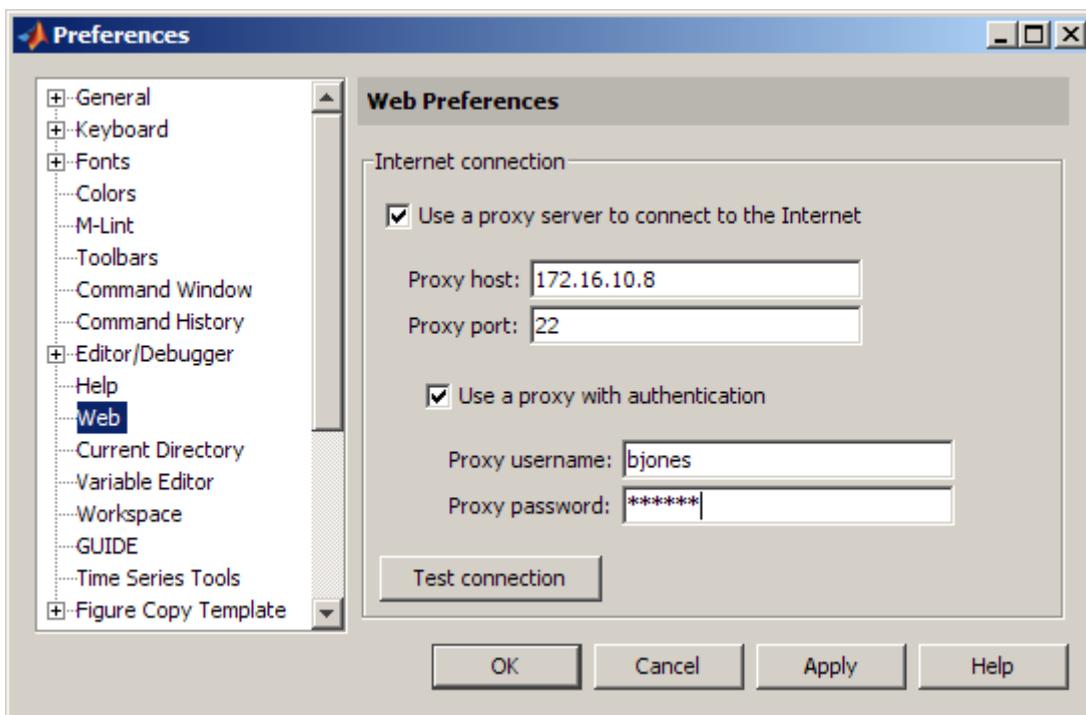
MATLAB attempts to connect to `http://www.mathworks.com`:

- `ser.xml`;

If MATLAB can access the Internet, **Success!** appears next to the button.

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

5 Click **OK** to accept the changes.



After specifying the preference, you can access the Internet from MATLAB through your proxy server.

Limitations of Specifying Proxy Server Settings.

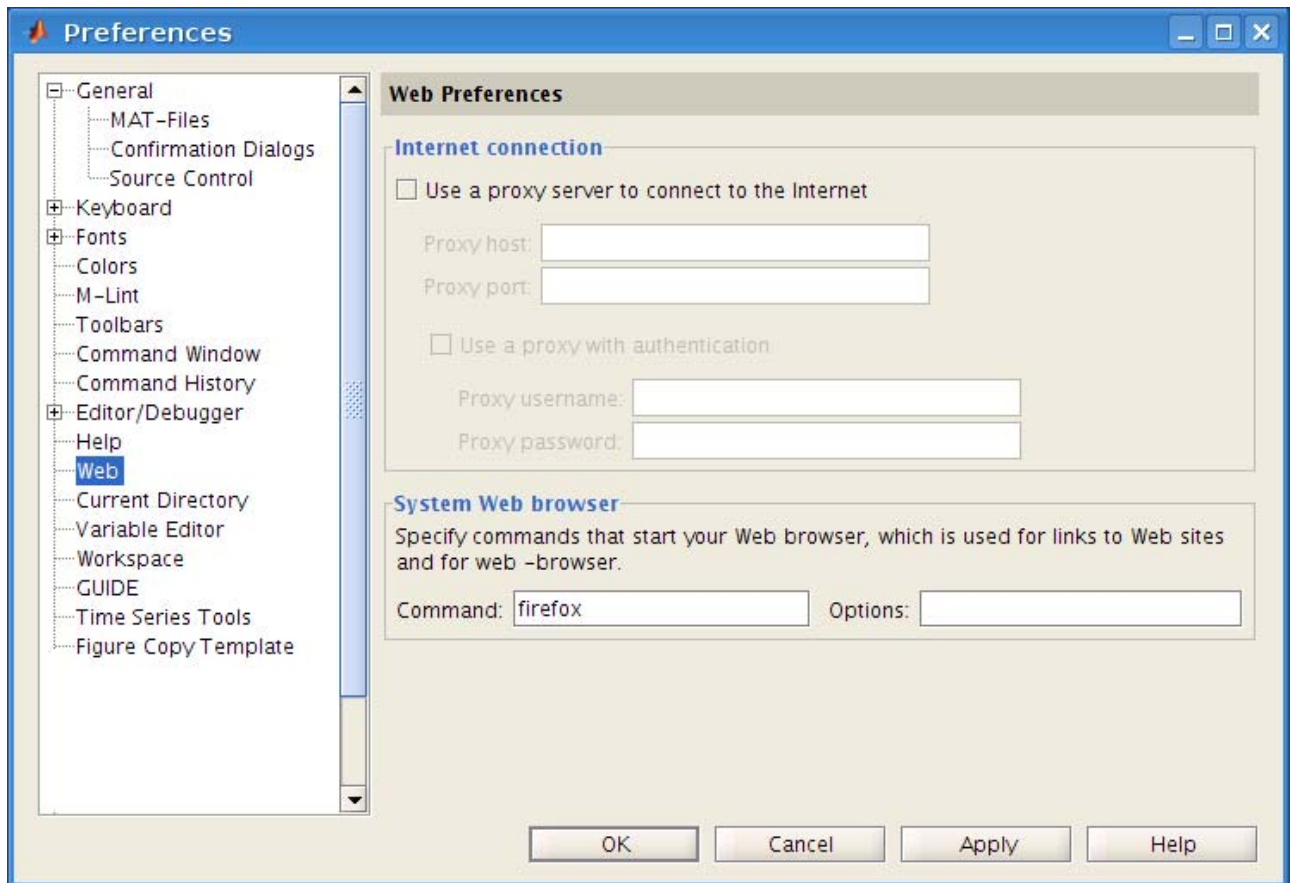
- MATLAB supports non-authenticated, basic, digest, and NTLM proxy authentication types.

- You cannot specify the proxy server settings using a script.
- There is no automated way to provide the proxy server settings your system browser uses to MATLAB.

Specifying the System Browser for UNIX Platforms

For background information, see “System Browser” on page 2-103. To specify the system browser:

- 1 Select **File > Preferences > Web**. The Preferences dialog box opens to the Web pane.



Note The **System Web browser** preference is for UNIX platforms (excluding Macintosh) and does not appear in the preferences pane for other platforms.

- 2** Under **System Web browser**, in the **Command** field, specify the system command to open the browser, for example, `opera`, which opens the Opera Web browser.
- 3** Add options for opening your system browser in the **Options** field. For example, `geometry 1064x860` specifies the size of the window for Opera.
- 4** Click **OK**.

Specifying Fonts for the MATLAB Web Browser

To modify the MATLAB Web Browser font, select

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

Other Features for Managing the Desktop

In this section...
“Using Menus and Context Menus” on page 2-108
“Using Toolbar Features” on page 2-110
“Viewing Status in the Status Bar” on page 2-111
“Sizing, Arranging, and Sorting Columns in Desktop Tools” on page 2-111
“Selecting Multiple Items” on page 2-113
“Cut, Copy, Paste, and Move” on page 2-114
“Printing and Page Setup Options for Desktop Tools” on page 2-115
“Accessing MathWorks on the Web” on page 2-119

Using Menus and Context Menus

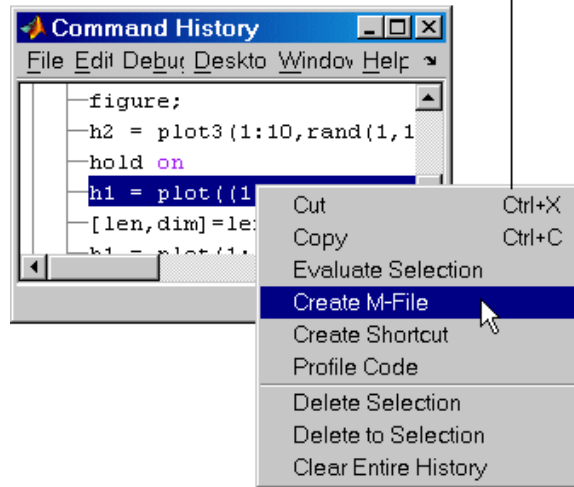
Understanding Merged Menus

When you use a tool in the desktop, its menu appears at the top of the desktop. When you work in a different tool in the desktop, you still use the menu at the top of the desktop. However, 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. The context menu for the selection or tool appears, presenting the available actions. For example, the following is the context menu for a selection in the Command History window.

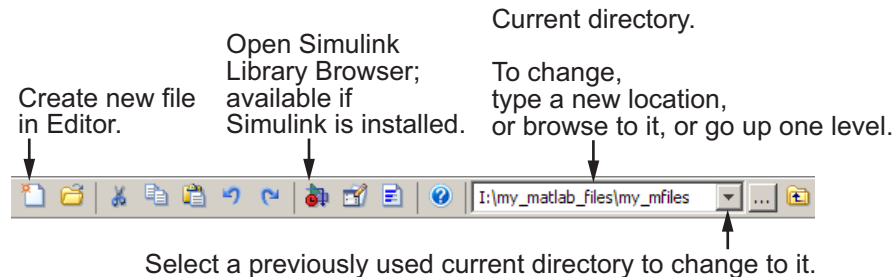
Access context (pop-up) menus by right-clicking a selection or any area in a tool.



If a context menu does not appear, try right-clicking in a different part of the tool. When a context menu item is gray, the item does not apply to the current selection or area.

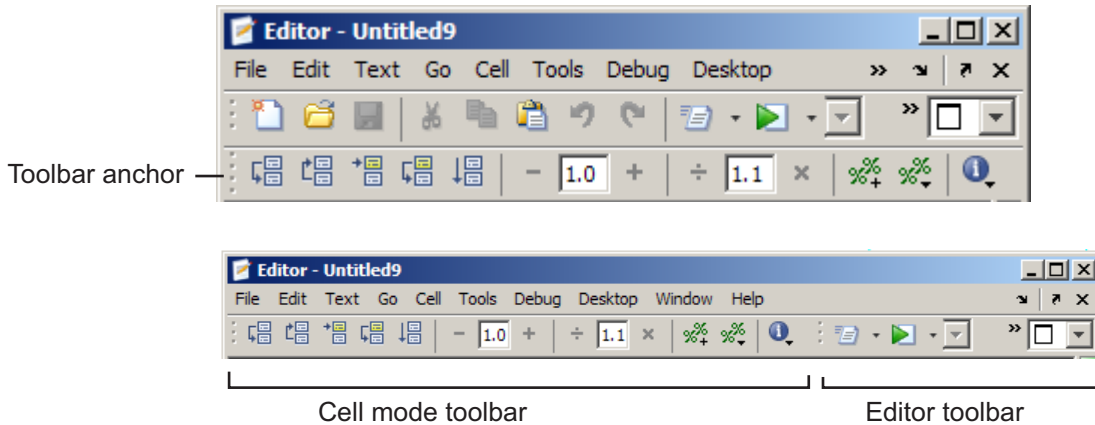
Using Toolbar Features

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



The following are the major toolbar features:

- **Tooltips** — Position the pointer over a button for a couple seconds and a tooltip appears describing the item.
- **Customizing** — You can customize the toolbar to show or remove controls, and to rearrange the controls. Use **File > Preferences > Toolbars**. For details, click **Help** in the resulting dialog box.
- **Toolbars in Tools** — Some tools have their own toolbars, located within the tool window. For example, the Current Folder browser has its own toolbar. When you undock one of these tools, the undocked tool includes the toolbar.
- **Hiding Toolbars** — To hide a toolbar, or to show it again after previously hiding it, select **Desktop > Toolbars**, and select the toolbar of interest. As an alternative, right-click a toolbar or menu bar and select a toolbar from the context menu to hide or show it. In a figure window, use the **View** menu to select the toolbar of interest.
- **Repositioning Toolbars** — If a tool has more than one toolbar, you can change the position of the toolbars. For example, in the Editor, the default is for the Editor toolbar to be above the Cell Mode toolbar. To move a toolbar, grab the toolbar anchor (at the left end) and drag the toolbar to a different location. The following images show the Editor toolbar before and after moving the Cell Mode toolbar to the left of the Editor toolbar.



Viewing and Changing the Current Folder in the Desktop Toolbar

The current folder field in the desktop toolbar shows the current working folder in MATLAB. Use this field to change the current folder. For example, to use a previous current folder, click the down arrow in the field and select a path from the history.

Viewing Status in the Status Bar

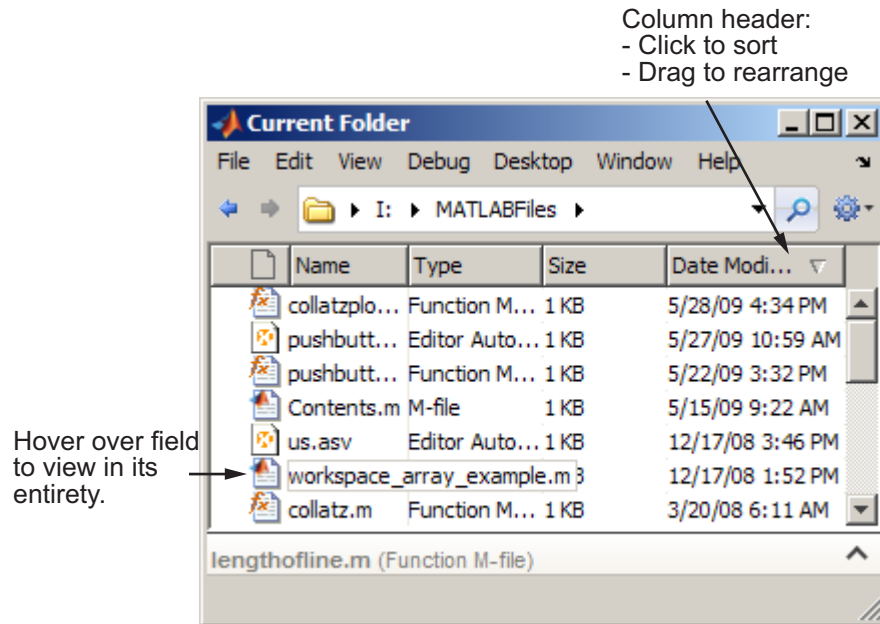
Along the bottom of the desktop is the status bar. It provides status information, such as when MATLAB is busy executing statements or when the Profiler is on.

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

Sizing, Arranging, and Sorting Columns in Desktop Tools

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

To...	Do This...
Change the column width	Drag the separator bar between two column headings.
View all the information in a column that is too narrow to show it all	Position the pointer over an item to view the full value for that item. It displays like a tooltip.
Rearrange the columns	Drag a column header to a different position.
Sort the information by a particular column	<p>Click the column header. For example, in the Current Folder browser, click the Date Modified date to sort the items in date order.</p> <p>In some columns, you also can reverse the sort order by clicking the column header again. A small gray arrow in the header indicates the current sort order. For example, a down arrow in the Date Modified column header indicates a descending sort order. The newest files are at the top of the list.</p>



Selecting Multiple Items

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

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

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

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

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

See also, “Performing Desktop Actions Using the Keyboard” on page 2-66

Cut, Copy, Paste, and Move

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

Use **Paste** to move items copied to the clipboard from other applications. The **Paste to Workspace** item in the **Edit** menu opens the selection on the clipboard in the Import Wizard. You can use this wizard to copy data from another application, such as the Microsoft Excel application, into MATLAB. For details, see “Tips for Using the Import Wizard” in the MATLAB Data Import and Export documentation.

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

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

See also the `clipboard` function.

Drag and Drop

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

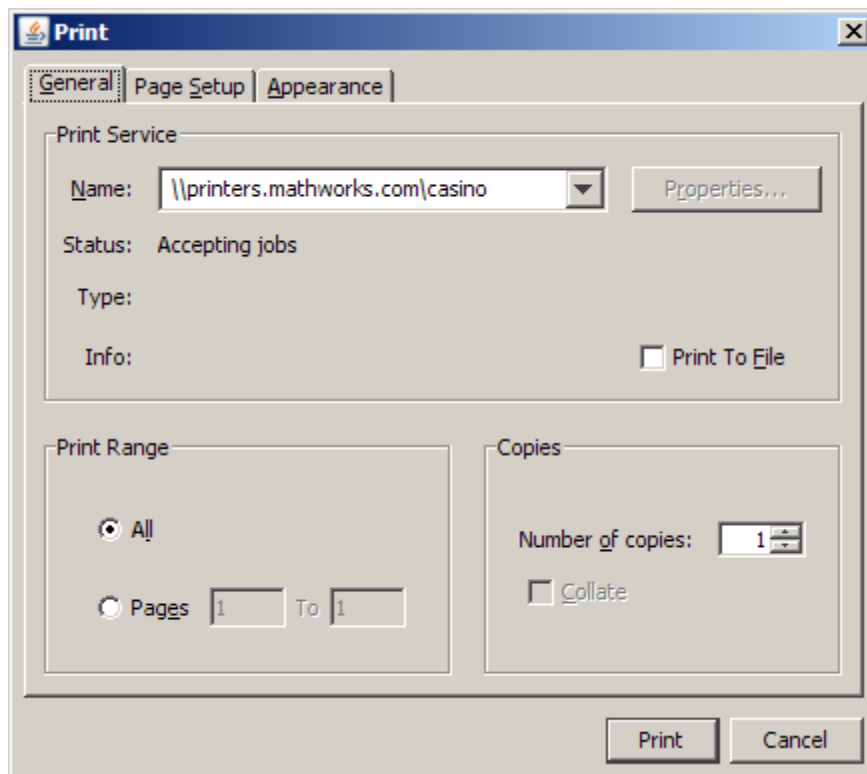
Another example is to open a file in the Editor by dragging the file name from the Current Folder browser to the Editor. If you drag editable text (for example, text in the Editor), it cuts the text rather than copies it. Use **Ctrl** and drag to copy rather than cut editable text.

On Windows platforms, you can drag items from external applications into MATLAB. For example, dragging text from a document created using the Microsoft Word application into the Editor cuts and pastes it into the open file. Dragging a MATLAB code file from Windows Explorer tool to the Command Window runs the file. Similarly, you can drag selections from desktop tools to other applications. For example, you can drag text from the Editor to the Word application.

Printing and Page Setup Options for Desktop Tools

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

To print, select **File > Print** from the tool. A Print dialog box opens. The **Properties** button in the Print dialog box is enabled for the Web browser, the Help browser, and the Profiler. However, it is disabled for the other desktop tools.



To specify standard page setup options for your platform when you print from the Command History, Workspace browser, and Variable Editor, select **File > Page Setup**. A standard page setup dialog box for your platform opens.

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

- “Specifying Page Setup Options” on page 2-117
- “Layout Options for Page Setup” on page 2-117
- “Header Options for Page Setup” on page 2-118
- “Fonts Options for Page Setup” on page 2-118

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 print, using the options you specified in Page Setup.

Layout Options for Page Setup

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

- **Print header** — Print the header specified in the **Header** pane.
- **Print line numbers** — Print line numbers.
- **Wrap lines** — Wrap any lines that are longer than the printed page width.
- **Syntax highlighting** — For keywords and comments that are highlighted in the Command Window, specify how they are to appear in print. Options are black and white text (that is, no highlighting), colored text (for use with a color printer), or styled text. For styled text, keywords appear in bold, comments appear in italics, and all other text appears in the normal style. Only keywords and comments you input in the Command Window are highlighted; output is not highlighted.

Header Options for Page Setup

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

- **Page number** — Format for the page number, for example # of n
- **Border** — Border style for the header, for example, Shaded box
- **Layout** — Layout style for the header. For example, Standard one line includes the date, time, and page number all on one line

Fonts Options for Page Setup

Specify the font to use 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 the element.

For example, if you access this dialog box while using the **Command Window**, you can select **Use Command Window font** for **Body** text. The printed text matches the **Command Window** font.

- 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.

Tip To change the font that a desktop tool uses, select **File > Preferences > Fonts > Custom**.

Accessing MathWorks on the Web

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

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

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

- **MathWorks Web Site** — Home page of the MathWorks Web site (<http://www.mathworks.com>).
- **Products & Services** — MathWorks Products and Services page (<http://www.mathworks.com/products/>) with information about the full family of products.
- **Support** — MathWorks Support page (<http://www.mathworks.com/support>) where you can look for solutions to problems, or report new problems.
- **Training** — List of courses for learning to use MathWorks products (<http://www.mathworks.com/services/training/courses/>).
- **MathWorks Account** — Login page for MathWorks Account (<http://www.mathworks.com/accesslogin/>). If you are registered, your main account page displays. Otherwise, you are directed to a page where you register online. Registration allows you to view your product registration and license information and helps you stay up to date on the latest developments for MATLAB.
- **MATLAB Central** — The user community for MATLAB (<http://www.mathworks.com/matlabcentral/>) for . It includes contests for MATLAB users and a screen saver with the logo for MATLAB.
- **MATLAB File Exchange** — Code library of files contributed by MathWorks customers and employees, available for free download and use with MathWorks products. You can also access the repository using the File Exchange desktop tool. For more information, see Chapter 8, “File Exchange — Finding and Getting Files Created by Other Users”.

- **MATLAB Newsgroup Access** — Provides access to the Usenet newsgroup for MATLAB and related products, `comp.soft-sys.matlab`, where you can post and answer questions, as well as view the archives.
- **MATLAB Newsletters** — Access to online versions of News and Notes and MATLAB Digest. News and Notes is published twice a year and contains feature articles, technical notes, and product information for users of MATLAB. MATLAB Digest, an electronic bulletin consisting of technical notes, solutions, and timely announcements to the user community, is issued more frequently. See <http://www.mathworks.com/company/newsletters>.

Managing Your Licenses

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

To access the licensing feature:

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

Note Some options require an Internet connection. If your Internet connection requires a proxy server, use MATLAB Web preferences to specify the server host and port. See “Specifying Proxy Server Settings” on page 2-104 for more information.

Option	Description
Activate Software	Starts the activation application, which walks you through the activation process. Answer the questions on each dialog box, select the license you want to activate, and click Activate .
Deactivate Software	<p>Displays a list of all your MathWorks licenses on this computer, with their current status. When you select a license and click Deactivate Selected License, MATLAB deactivates all releases on this computer associated with the license, and updates the licensing information at the MathWorks Web site. You will not be able to use MathWorks software with that license on this computer.</p> <p>If you are not connected to the Internet, MATLAB deactivates the licences on your computer but cannot update the corresponding license information stored at the MathWorks Web site. In this scenario, MATLAB returns a <i>deactivation string</i>. To complete deactivation, save a copy</p>

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

Check for Updates

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

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

- 1** Select **Help > Check for Updates**. The Check for Updates dialog box displays.
- 2** From the **Select View** list, choose to view the latest version numbers for all MathWorks products installed on your system, or all MathWorks products. The latest versions are displayed.
- 3** Click any column heading to sort or reverse the sort order by that column.
- 4** Use the What's New column to access the release notes for a product. Release notes document new features and changes, bug reports, and compatibility considerations.
- 5** To upgrade to the most recent version, click **Download Products at MathWorks.com**, which links to the Downloads area of the MathWorks Web site. If you do not want to upgrade at this time, click **Close**.

Specifying Options for MATLAB Using Preferences

In this section...

“Setting Preferences for MATLAB” on page 2-124


“Summary of Preferences” on page 2-125

“Where MATLAB Stores Preferences” on page 2-126

“Preferences Folder and Files MATLAB Uses When Multiple MATLAB Releases Are Installed” on page 2-127

Setting Preferences for MATLAB

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

- 1** Select **File > Preferences**. Alternatively, click the Preferences button  on the desktop toolbar; if the button is not on the toolbar, you can add it—for information, see “Setting Toolbars Preferences for Desktop Tools” on page 2-156.
- 2** From the left pane of the Preferences dialog box, choose a tool or product and click the plus sign (+) to display more preferences for that item. From the expanded list, select the entry you want. The right pane shows the preferences for that item.
- 3** Change settings. Click **Apply** or **OK** to set the preferences. Preferences take effect immediately. They remain persistent across sessions of MATLAB.

Function Alternative

Open the Preferences dialog box using the preferences function.

Summary of Preferences

Preference	What You Can Specify
General Preferences	Toolbox path caching, figure window printing, delete function behavior, MAT-file save formats, confirmation dialogs, and source control.
Keyboard	Tab completion, function hints, and delimiter matching for the Command Window and Editor. Keyboard shortcuts for desktop tools.
Fonts	Font type, style, and size for desktop tools. Customize for any tool.
Colors	Colors for text, background, syntax highlighting, hyperlinks in desktop tools. Colors for code analyzer, variable and function highlighting, and cell display in the Editor
Code Analyzer	Show or hide M-Lint messages in the Editor and in the Code Analyzer Report.
Toolbars	Remove, add, and rearrange controls on toolbars for desktop tools.
Command Window	Numeric format and display, accessibility, and tab size.
Command History	Display, filtering, and saving.
Editor/Debugger	Editor type, startup options, display, tab size and indenting, language (including syntax highlighting colors for all files other than MATLAB files), code folding, and autosave.
Help	Product filter, help on selection window, and PDF reader for Linux platforms.
Web	Internet proxy server settings.
Current Folder	Number of entries in history and refresh options.
Variable Editor	Numeric format, use of Enter key, and decimal separator.

Preference	What You Can Specify
Workspace	Statistical calculation options.
GUIDE	Display options for GUI-building tool.
Time Series Tools	Property Editor dialog and x-axes warning dialog. For details, click the Help button in the Preferences dialog box.
Figure Copy Template	Application, text, line, uicontrols, axis, format, background color, and size.
Other products	Preferences for other installed MathWorks products.

Where MATLAB Stores Preferences

MATLAB and other MathWorks products store their preferences in the file `matlab.prf`. This file loads when you start MATLAB. The folder containing this file is called the preferences folder. The preference folder also contains other related files.

The Path to and File Name for the Preferences Folder

To see the full path for the folder where `matlab.prf` and related files are located, type `prefdir` in the MATLAB Command Window.

On Apple Macintosh platforms, the folder can be in a hidden folder, for example, `myname/.matlab/R2009b`. If so, to access the hidden folder:

- 1 In the AppleMac OS® Finder tool, select **Go > Go to Folder**.
- 2 In the resulting dialog box, type the path returned by `prefdir`, and then press **Enter**.

The name of the preferences folder, matches the name of the release. For instance, for MATLAB R2010b, the name of the preferences folder is `R2010b`.

Effects of Changing Preferences

When you change preferences using the MATLAB Desktop (**File > Preferences**), it updates `matlab.prf`. When you close MATLAB, it saves those changes to `matlab.prf`.

Effects of Installation and Deinstallation on the Preferences Folder

Installing MATLAB has no effect on the preferences folder. That is, MATLAB creates, checks, copies, and writes to the preferences folder when you start up MATLAB, not when you install it. When you deinstall MATLAB, there is an option in the uninstaller to remove the preferences folder. However, this option is not selected by default.

Preferences Folder and Files MATLAB Uses When Multiple MATLAB Releases Are Installed

The files in the preferences folder that MATLAB uses depends on the version of MATLAB you are starting up. How and if MATLAB migrates (reuses) preferences files from one version to the next also depends on the version.

Process MATLAB Uses to Create and Migrate the Preferences Folder and its Files

When you start it up, MATLAB looks for a preferences folder name that matches the release starting up, and then does one of the following:

- If MATLAB finds a preferences folder name matching the release starting up, it uses that folder and the files within it.

If that folder is empty, MATLAB recreates the default files for the release starting up.

- If MATLAB does not find a preferences folder name matching the release starting up, it creates one. Then, MATLAB checks to see if the release of MATLAB that immediately precedes the one you are starting up is installed.

- If that previous release is not installed, MATLAB recreates the folder and default files for the version starting up.

For example, if you start up R2010b and R2010a is not installed, then MATLAB recreates the default files for the R2010b preferences folder. This is true even if R2009b or earlier is installed.

- If that previous release is installed, MATLAB migrates the files from the preferences folder corresponding to that previous release to the preferences folder for the release starting up.

For example, if you start up R2010b and R2010a is installed, then MATLAB migrates the files from R2010a preferences folder to the R2010b preferences folder.

Controlling the Preferences Files MATLAB Uses

This table describes how to control which versions of preferences files MATLAB uses.

To Use:	Do This:
Default preference files for a given release of MATLAB	Make sure the preferences folder for that release exists, but is empty before starting up that MATLAB version.
All the preference files from the release of MATLAB immediately preceding the release you plan to start up.	Ensure that the preferences folder exists for that preceding release. If so, delete the entire preferences folder for the release of MATLAB you plan to start up.
The release-specific default for just a particular file in the preferences folder	Delete just that file from the preferences folder for the release of MATLAB you plan to start up. One file to consider keeping is <code>history.m</code> . For more information about that file, see “Viewing Statements in the Command History Window” on page 3-68.

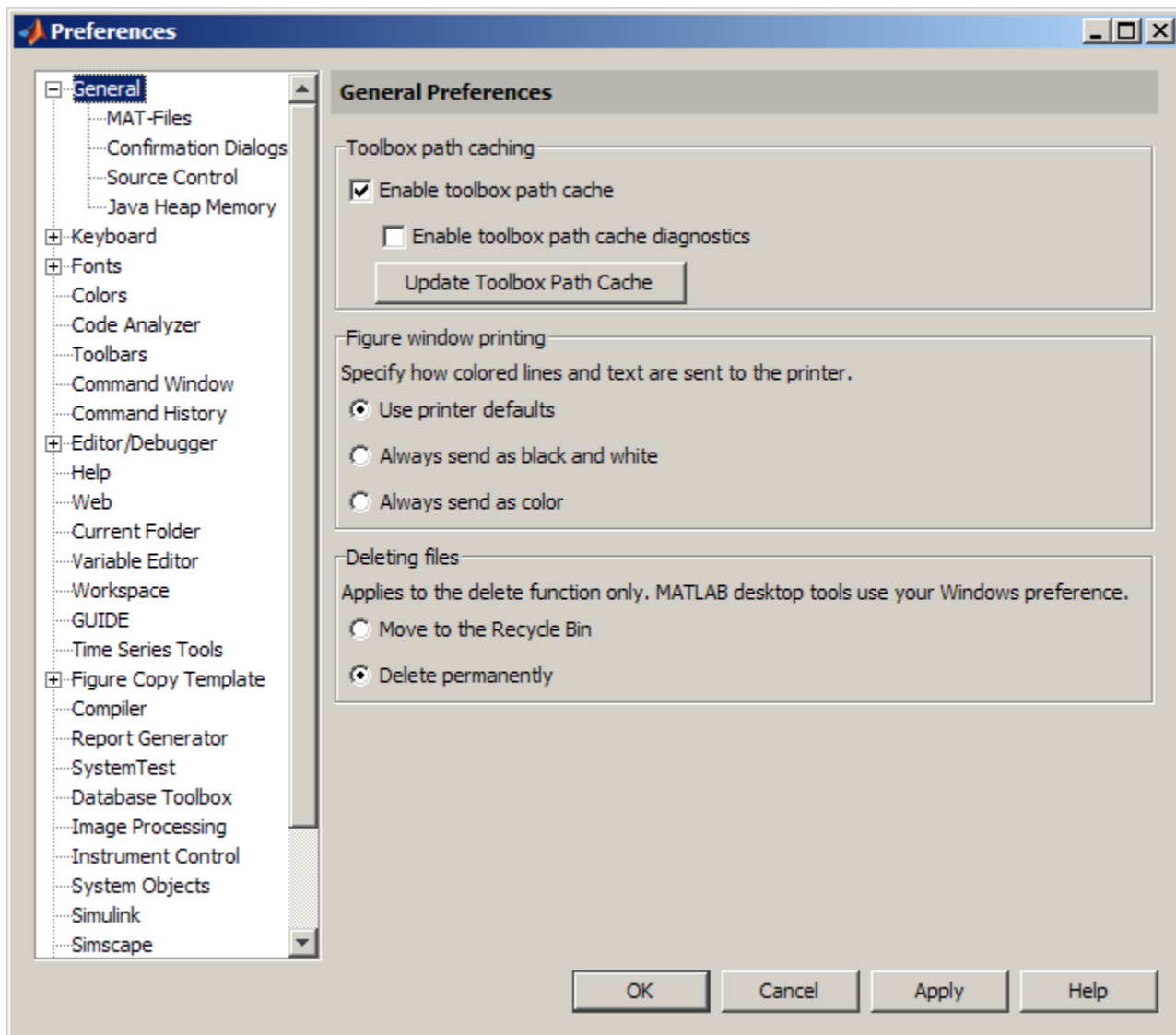
Setting General Preferences for the MATLAB Application

In this section...
“General Preferences” on page 2-129
“MAT-Files Preferences” on page 2-131
“Confirmation Dialogs Preferences” on page 2-132
“Source Control Preferences” on page 2-136
“Java Heap Memory Preferences” on page 2-136

General Preferences

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

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



Toolbox Path Caching

See “Toolbox Path Caching in the MATLAB Program” on page 1-19.

Figure Window Printing

See “Printing and Exporting” in MATLAB Graphics documentation.

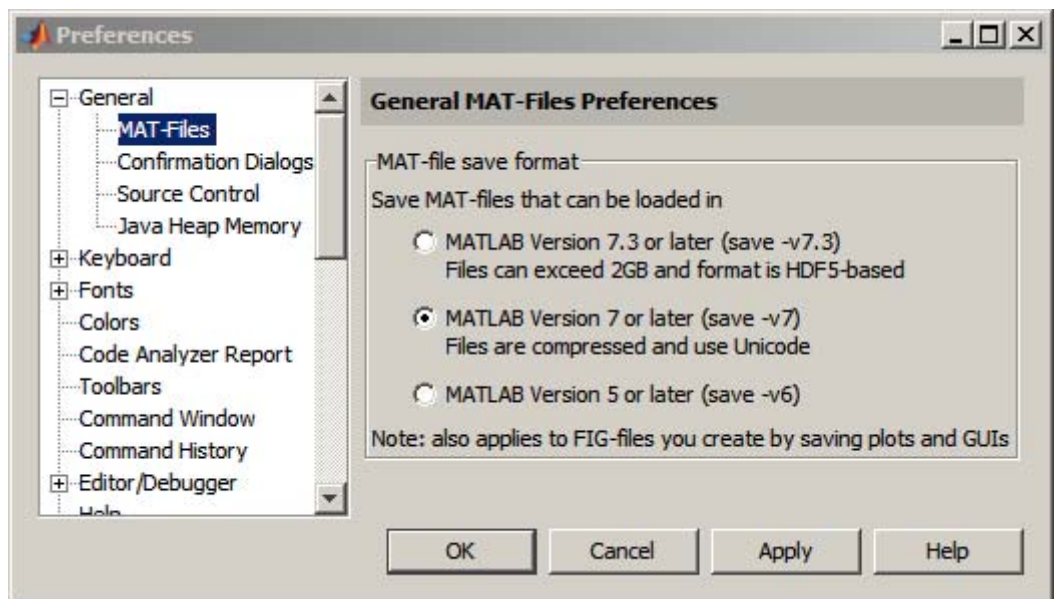
Deleting Files

See:

- “Deleting Files and Folders Using the Current Folder Browser” on page 7-42
- “Deleting Files and Folders Using Functions” on page 7-44

MAT-Files Preferences

The **MAT-file save format** sets the default version for creating MAT-files and FIG-files. The setting applies to the **save** function and **Save** menu items such as **File > Save Workspace As**.



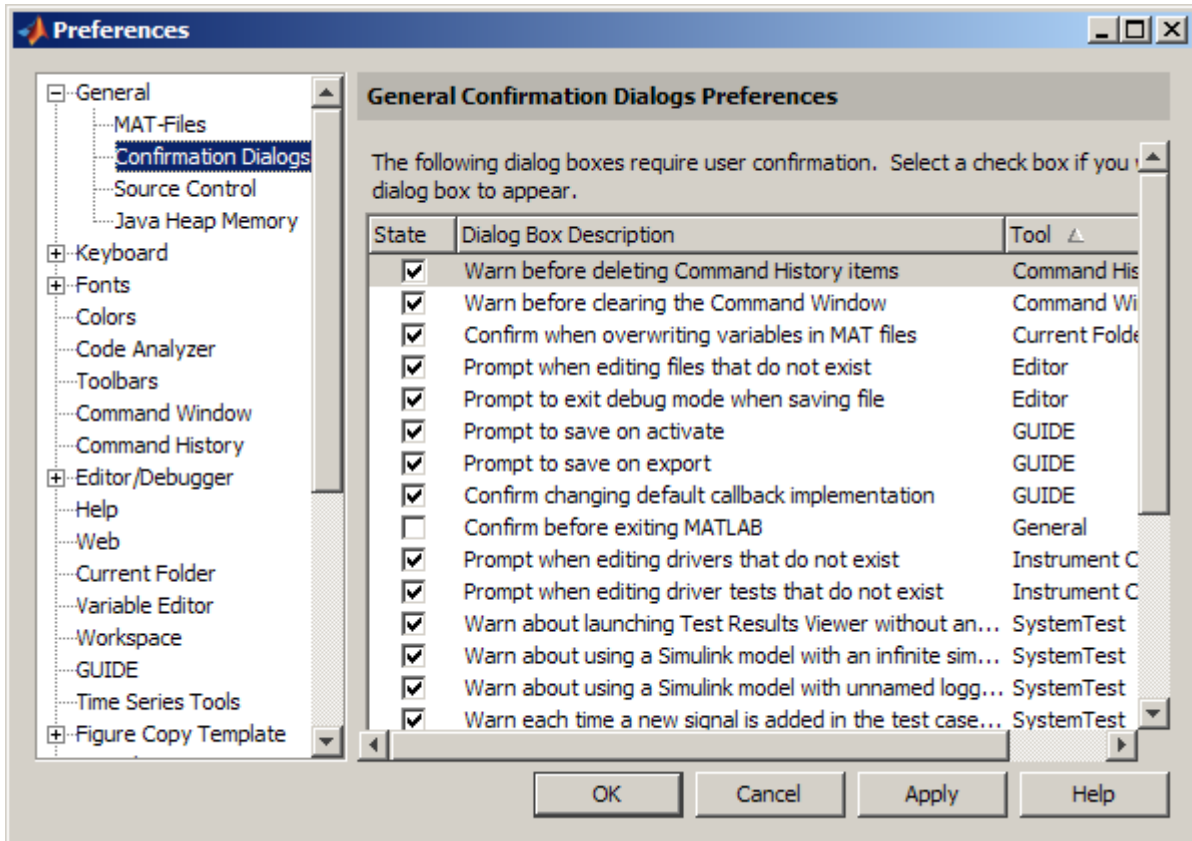
Options are:

- **MATLAB Version 7.3 or later (save -v7.3)** — On 64-bit systems, this option allows you to save data items larger than 2 GB. As with Version 7, files are compressed and use Unicode® character encoding.
- **MATLAB Version 7 or later (save -v7)** — MATLAB compresses data in Version 7 MAT-files to reduce the required storage space, and automatically decompresses the data during load operations. Unicode character encoding allows other MATLAB users to access the data, regardless of the default character encoding scheme used by their systems. This is the default on new installations of MATLAB software, or upgrades from MATLAB versions earlier than 7.3.
- **MATLAB Version 5 or later (save -v6)** — Specify this option to save MAT-files for use with versions prior to MATLAB Version 7, or to create uncompressed files.

For more information, see “MAT-File Versions” in the MATLAB Data Import and Export documentation.

Confirmation Dialogs Preferences

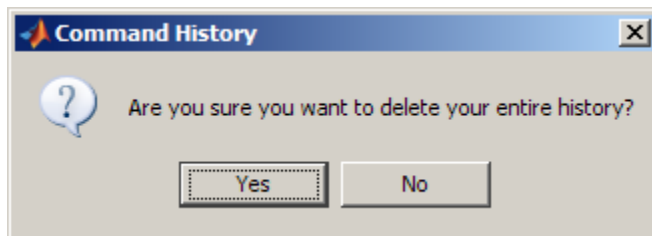
These preferences instruct MATLAB to display or not display specific confirmation dialog boxes.



When the check box for a confirmation dialog is selected and you perform the action it refers to, the confirmation dialog box appears. If you clear that check box, the dialog box does not appear when you perform the action.

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

For example, select the check box **Warn before deleting Command History items**. Then select **Edit > Clear Command History**. MATLAB displays the following confirmation dialog box.



If a confirmation dialog box includes a **Do not show this prompt again** check box and you click **OK**, the confirmation dialog box will not appear the next time you perform the action. In addition, the check box in the **Confirmations Dialogs** preferences pane is cleared.

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

Confirmation Dialogs Check Box Item	Description of the Confirmation Dialog Box	For More Information
Warn before deleting Command History items	Appears when you delete entries from the Command History window.	“Deleting Entries from the Command History Window” on page 3-76
Warn before clearing the Command Window	Appears when you clear the Command Window content using menu items. Does not appear when you use the <code>clc</code> function.	“Clearing the Command Window” on page 3-50
Confirm when overwriting variables in MAT-files	Appears when you save variables by dragging them from the Workspace browser onto a MAT-file in the Current Folder browser.	“Creating and Updating MAT-Files with the Current Folder Browser” on page 7-37
Prompt when editing files that do not exist	Appears when you type <code>edit filename</code> , if <code>filename</code> does not exist in the current folder or on the search path.	“Function Alternative for Creating New Files” on page 9-8

Confirmation Dialogs Check Box Item	Description of the Confirmation Dialog Box	For More Information
Prompt to exit debug mode when saving file	Appears when you try to save a modified file while in debug mode.	“Ending Debugging” on page 9-158
Prompt to save on activate	Appears when you have unsaved changes to a figure and program file, and then activate the GUI, by clicking the Run button, for example.	“GUIDE Preferences” in the MATLAB Creating Graphical User Interfaces documentation
Prompt to save on export	Appears when you have unsaved changes to a figure and program file, and then select File > Export .	“GUIDE Preferences” in the MATLAB Creating Graphical User Interfaces documentation
Confirm changing default callback implementation	Appears after you have modified a callback signature in GUIDE.	“Changing Callbacks Assigned by GUIDE” in the MATLAB Creating Graphical User Interfaces documentation
Confirm before exiting MATLAB	Appears when you quit MATLAB.	Quitting MATLAB
Warn about missing search databases	Appears if you have help files in the Help browser for products not produced by MathWorks and the search database for those files has not been updated for the version of MATLAB you are running.	Contact the provider of the help files to obtain the correct version of the search database. Without the most current version, you can use the help files in the Help browser, but the Help browser search will not include those files in search results.
Confirm when deleting variables	Appears when you delete variables from the workspace using menu items. Does not appear with the <code>clear</code> function.	“Deleting Workspace Variables” on page 6-9

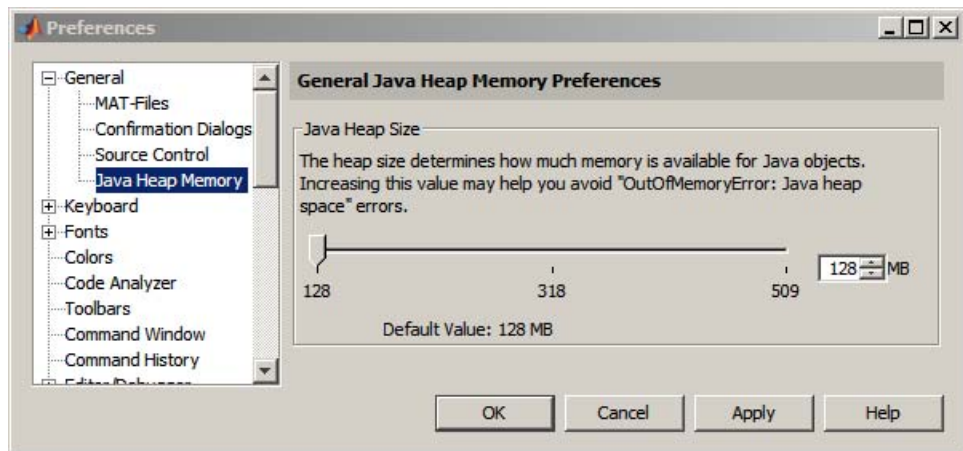
Source Control Preferences

For information, see Chapter 13, “Source Control Interface”.

Java Heap Memory Preferences

The **Java Heap Size** allows you to adjust the amount of memory that MATLAB software allocates for Java objects.

Note The default heap size is sufficient for most cases.



To adjust the Java heap size:

- 1 Select **File > Preferences > General > Java Heap Memory**.
- 2 Select a heap size value using the slider or spin box.

Note Increasing the Java heap size decreases the amount of memory available for storing data in arrays.

3 Select **OK** to store the value in the MATLAB preferences file and close the window. If you select **Apply**, MATLAB stores the value, but does not close the window.

4 Restart MATLAB.

If the requested amount of memory is not available upon restart, MATLAB resets the value in the preferences file to the default, and displays an error dialog box. To readjust the value, repeat the previous steps.

If increasing the heap size does not eliminate memory errors, check your Java code for memory leaks. Eliminate references to objects that are no longer useful. For more information, see the Java SE Troubleshooting guide at <http://java.sun.com/javase/6/webnotes/trouble/>.

Customizing the Desktop Using Preferences

In this section...
“Setting Keyboard Preferences for Desktop Tools” on page 2-138
“Setting Fonts Preferences for Desktop Tools” on page 2-141
“Setting Colors Preferences” on page 2-150
“Setting Color Preferences for Programming Tools” on page 2-154
“Setting Toolbars Preferences for Desktop Tools” on page 2-156

Setting Keyboard Preferences for Desktop Tools

Select **File > Preferences > Keyboard** to set the following preferences for the Command Window and Editor/Debugger:

- Tab completion
- Function hints
- Delimiter matching

See also “Customizing Keyboard Shortcuts” on page 2-79.

Setting Tab Completion Preferences

Enable in Command Window. Select the check box to use tab completion when typing functions in the Command Window—for more information about the feature, see “Completing Statements in the Command Window — Tab Completion” on page 3-24. Clear the check box if you do not want to use the tab completion feature. With the tab completion preference cleared, when you press the **Tab** key, MATLAB moves the cursor to the next tab stop rather than completing a function. See also the preference for “Tab size” on page 3-64.

Enable in Editor/Debugger. Select the check box to use tab completion when typing functions in the Editor—for more information about the feature, see “Completing Statements in the Command Window — Tab Completion” on page 3-24. Clear the check box if you do not want to use the tab completion feature. With the tab completion preference cleared, when you press the **Tab** key, MATLAB moves the cursor to the next tab stop rather than completing a function. For related information, select **File > Preferences > Editor/Debugger > Tab**, and click **Help**.

Tab key narrows completions. Select this check box to narrow the list of possible completions shown by typing another character and pressing **Tab**. For details, see “Narrowing Completions Shown” on page 3-27.

View Command Window tab key preferences. Click the link to set preferences for the **Tab** key size in the Command Window, which MATLAB uses when the tab completion preference is not enabled.

View Editor/Debugger tab key preferences. Click the link to set preferences for the **Tab** key size and indenting preferences in the Editor/Debugger.

Setting Function Hints Preferences

To show function hints in the Command Window and Editor, select the function hints check boxes. If you do not want to use function hints, clear the check boxes. Function hints are a reminder of the syntax for a function that you use while entering a statement. The hints appear in a temporary pop-up window when you enter the opening parenthesis after a function name. For more information, see “Viewing Function Syntax Hints While Entering a Statement” on page 3-33.

Setting Delimiter Matching Preferences

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

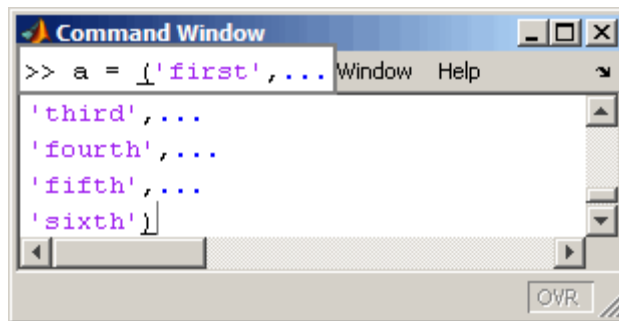
With these preferences selected, MATLAB alerts you to matched and unmatched delimiters based on the MATLAB language syntax rules. For example, when you type a parenthesis or another delimiter, MATLAB highlights the matched parenthesis or delimiter in the pair.

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

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

```
len(n1) = sum(sqrt(dot(temp',temp')));
```

If the matching delimiter is not visible on the screen, a pop-up window appears and shows the line containing the matching delimiter. In the Editor, the line number is included. Click in the pop-up window to go to that line.



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

- **Balance** — The corresponding delimiter is highlighted briefly.
- **Underline** — Both delimiters in the pair are underlined briefly.
- **Highlight** — Both delimiters in the pair are highlighted briefly.

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

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

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

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

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

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

Setting Fonts Preferences for Desktop Tools

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

- The operating system's standard location
Ask your system administrator where this is on your system.
- The `/jre/lib/fonts` folder where Java software is installed on your system.

See the following sections for details on setting fonts preferences:

- “Desktop Fonts Preferences” on page 2-142
- “Custom Fonts Preferences” on page 2-146
- “Changing the Font — Example” on page 2-147
- “Antialiasing for Desktop Fonts on Linux and UNIX Platforms” on page 2-149
- “Making Fonts Available to MATLAB Tools on Windows Platforms” on page 2-150

Desktop Fonts Preferences

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

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

Select **File > Preferences > Fonts** to set fonts for desktop tools. You can specify:

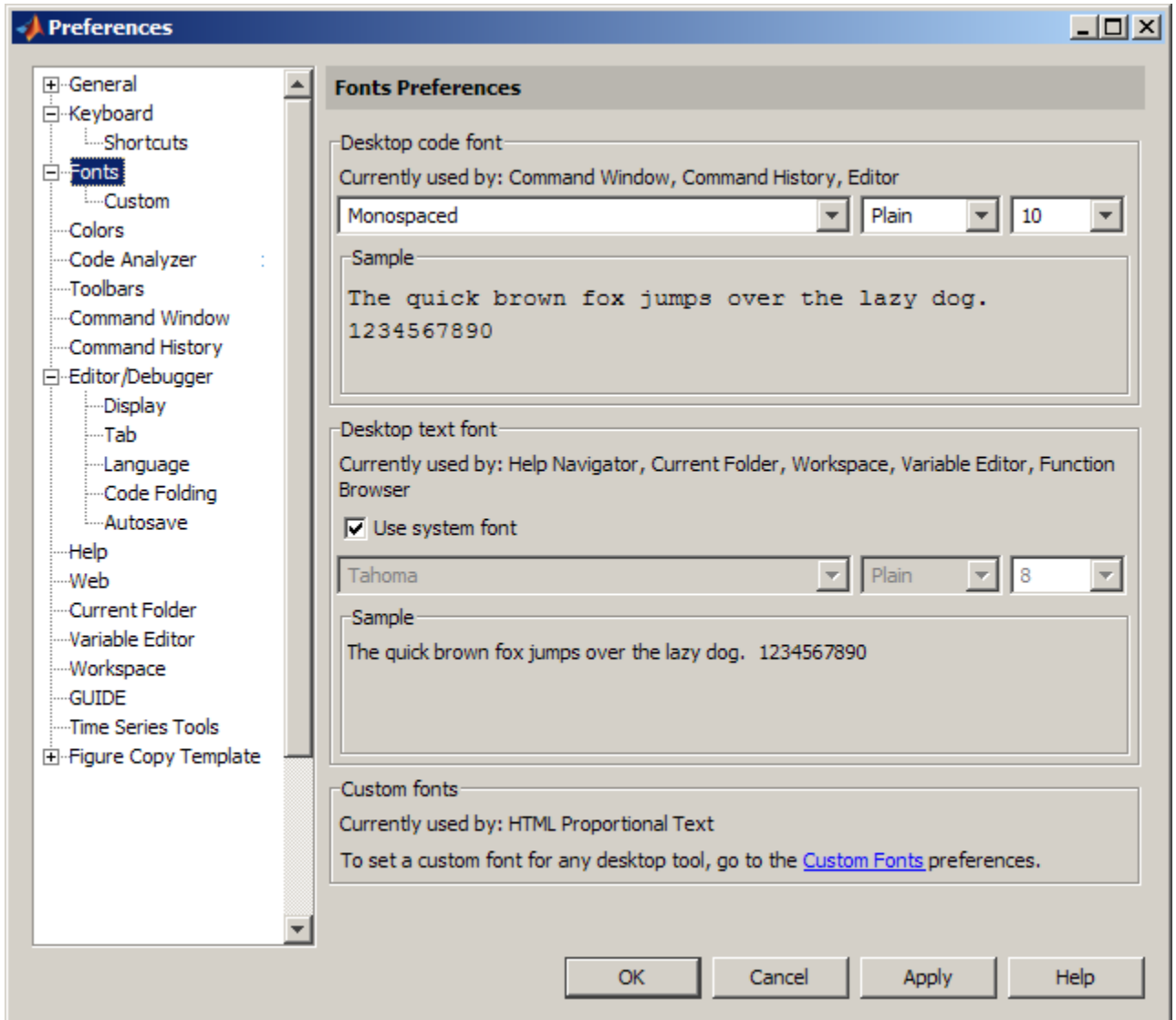
- A font for all the tools that primarily display code, such as the Command Window
- A font for all the tools that display text, such as the Current Folder
- Custom fonts, including a font for all the tools that use HTML Proportional text, such as the Help display pane and the MATLAB Web browser

If you want, you can separately specify the font for each desktop tool.

Select the font characteristics from the lists shown. For font size, you can type or select a size. You can type a size not shown as a choice in the drop-down menu.

You can set some font options differently for printing — see “Printing and Page Setup Options for Desktop Tools” on page 2-115.

For information about making additional fonts available to MATLAB, see “Making Fonts Available to MATLAB Tools on Windows Platforms” on page 2-150.



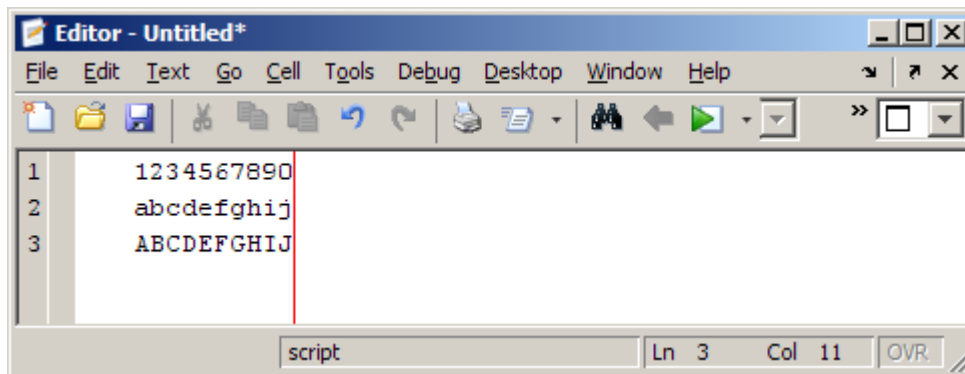
Desktop Code Font and Desktop Text Font. You specify separate font characteristics for tools that primarily display code (**Desktop code font**), such as the Command Window, and tools that primarily display text (**Desktop text font**), such as the Current Folder browser. (For other tools, such as the Help display pane and the MATLAB Web Browser you use custom fonts—see “Custom Fonts Preferences” on page 2-146.)

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

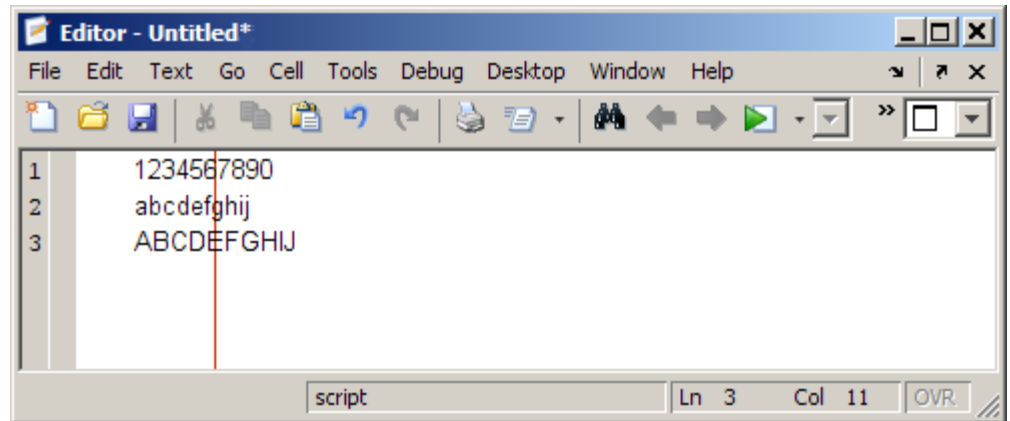
Typically, you specify a proportional font for tools that display little or no code. You use the desktop text font preference to set just one preference that applies to all tools that display little or no code. If you want to use the system font as the desktop text font, select **Use system font**.

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

With a monospaced font, all characters are the same width. Here, the font is 10 pt. Monospace. Note the 10th character in each line aligns with the Editor's right-hand text limit, which is set to column 10.



With a proportional font, characters are different widths. Here, the font is 10 pt. SanSerif. Each line contains 10 characters, but the length of each line differs. The Editor's right-hand text limit, at column 10, is not relevant.



When you change a font characteristic for **Desktop code font**, the characteristic takes effect for all tools that use the desktop code font. The same is true when you change a font characteristic for **Desktop text font**.

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

Factory Default Font Settings

The following table lists the factory default code and text font settings, and the tools that use those font settings. If you previously changed settings, and now want to revert to the default values, update the settings using the values in the table.

Font Type	Factory Default Characteristics and Sample	Tools Using Font Type by Factory Default
Desktop code font	Monospaced, Plain, 10 point Sample text font	<ul style="list-style-type: none"> • Command History • Command Window • Editor (which also applies to the Shortcuts Editor)
Desktop text font	Your system's current font.	<ul style="list-style-type: none"> • Current Folder browser (which also applies to the Path browser) • Function Browser in the Command Window and Editor • Help Navigator • Workspace browser • Variable Editor

See Also

“Specifying Options for MATLAB Using Preferences” on page 2-124

Custom Fonts Preferences

Use custom font preferences to specify the font for HTML Proportional Text, and to override font settings for individual desktop tools. Desktop tools otherwise use the settings that the **Fonts** pane specifies. The **Fonts** pane is described in “Desktop Fonts Preferences” on page 2-142.

HTML Proportional Text is the default font for the following tools and portions of tools:

- The MATLAB Help display pane
- The small window that opens for the help on selection feature.
- The MATLAB Web browser— which displays the HTML output generated from publishing
- The Profiler

- The Function Browser function help
- Extended Code Analyzer messages

To specify custom fonts preferences:

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

The **Fonts Custom Preferences** pane appears.

2 Select the tool for which you want to specify custom fonts from the **Desktop tools** list. The type of font the tool currently uses appears under **Font to Use**.

3 For **Font to Use**, select one of the following:

- **Desktop code**, which you can customize using the **Fonts** pane.
- **Desktop text**, which you can customize using the **Fonts** pane.
- **Custom**, and then specify the font characteristics.

4 Click **OK**.

Note If you change the font style (for example, to bold or italic) for **HTML Proportional Text**, it has no effect. If you change the font size, it affects both noncode and code text for tools using the HTML Proportional Text font.

Changing the Font – Example

This example:

- Changes the settings for the desktop code font from the factory default settings. (See Factory Default Font Settings on page 145.)
- Changes the Command History font preference so that it uses the desktop text font instead of the code font.
- Specifies a custom font for the Current Folder browser.

1 Select **File > Preferences > Fonts**.

- 2** Under **Desktop code font**, select Times New Roman, Plain, 14 point.
- 3** Under **Desktop text font**, select **Use system font**.
- 4** Click **Apply**.
- 5** Make the Command History window use the desktop text font:
 - a** Click the **Custom Fonts** link.
 - b** From **Desktop tools**, select Command History.
 - c** Select the **Desktop text** radio button.
 - d** Click **Apply**.
- 6** Apply a custom font to the Current Folder browser:
 - a** From **Desktop tools**, select Current Folder.
 - b** Select the **Custom** radio button.
 - c** Select Arial Narrow and Plain, and then type 11 in the size field.
 - d** Click **OK**.

The following table details the results of the changes.

Tool	Font Type	Font Characteristics
Command Window	Desktop code	Times New Roman® font, Plain, 14 point
Command History	Desktop text	Same as your current system font, which appears in the dimmed fields below the Use system font check box.
Editor	Desktop code	Times New Roman font, Plain, 14 point
Help Navigator	Custom	SanSerif, Plain, 8 point
HTML Proportional Text	Custom	SansSerif, Plain, 10 point

Tool	Font Type	Font Characteristics
Current Folder browser	Custom	Monotype Corporation Arial® Narrow font, Plain, 11 point
Workspace browser	Desktop text	Same as your current system font, which appears in the dimmed fields below the Use system font check box.
Variable Editor	Desktop text	Same as your current system font, which appears in the dimmed fields below the Use system font check box.
Function Browser	Desktop text	Same as your current system font, which appears in the dimmed fields below the Use system font check box.

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

See Also. For information about how MATLAB stores preferences, and to get help for other preferences, see “Specifying Options for MATLAB Using Preferences” on page 2-124.

Antialiasing for Desktop Fonts on Linux and UNIX Platforms

To give the desktop a smoother appearance on Linux³ and UNIX⁴ platforms, select the antialiasing preference on the **Preference > Fonts** pane. The preference applies to all fonts.

3. Linux is a registered trademark of Linus Torvalds.

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

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

Making Fonts Available to MATLAB Tools on Windows Platforms

Under the following circumstances, consider updating fonts on your Windows platform as described:

- If a new compatible font is available to MATLAB

A compatible font on Windows platforms for desktop components (such as the Command Window), figure windows, and uicontrols is one compatible with TrueType® and Microsoft OpenType® fonts. A compatible font for graphics objects, such as `xlabel`, `ylabel`, `title`, and `text` is a bitmapped font.

Use the Windows Control Panel to install the font. Then, restart MATLAB so that it can use the font.

- If you open a file created by someone else, and you see boxes or meaningless symbols instead of text.

When you see boxes or meaningless symbols instead of text, it is probably because you are using different language fonts from the file creator. This situation can occur, for example, if you open a file created by someone whose native language is Japanese and your native language is English. The Japanese user is probably using fonts for East Asian languages and you are not.

In the Windows Control Panel, find the region and language options, and then install the supplemental files for East Asian languages.

For more information, refer to the Windows help.

Setting Colors Preferences

- “Setting Colors Used in Desktop Tools” on page 2-151

- “Desktop Tool Colors” on page 2-151
- “MATLAB Syntax Highlighting Colors” on page 2-152
- “Other Colors” on page 2-154
- “See Also” on page 2-154

Setting Colors Used in Desktop Tools

Desktop color preferences specify the colors used in MATLAB desktop tools and the colors that convey syntax highlighting. Select **File > Preferences > Colors** to set color preferences for desktop tools. For instance:

- To set colors for text and the background:
 - 1** Clear **Use system colors**.
 - 2** Select the colors you want to use from the **Text** and **Background** color palettes.
- To set the color of hyperlinks in the Command Window, select a color from the **Hyperlinks** color palette.

You can set some color options differently for printing — see “Printing and Page Setup Options for Desktop Tools” on page 2-115.

Desktop Tool Colors

Use **Desktop tool colors** to change the color of the text and background in the desktop tools. The colors also apply to the Import Wizard. The colors do not apply to the HTML display pane nor to the Web Browser.

Select the check box **Use system colors** if you want the desktop to use the same text and background colors that your platform (for example, Microsoft Windows) uses for other applications.

To specify different text and background colors, follow these steps:

- 1** Clear the **Use system colors** check box.
- 2** Click the arrow next to the **Text** color and choose a new color from the palette shown.

When you choose a color, the **Sample** area in the dialog box updates to show you how it will look.

- 3** Click the arrow next to the **Background** color and choose a new color.

If you use a gray background color, a selection in an inactive window will not be visible.

- 4** Click **Apply** or **OK** to see the changes in the desktop tools.

Click **Restore Default Colors** to return to the default settings for desktop tool colors, as well as for syntax highlighting colors.

Gray Background Color. For some UNIX⁵ platforms, there is a gray background color for desktop tools, such as the Editor. This occurs when the preference for **Desktop tool colors** is set to **Use system colors**, and the system's window manager uses gray as the background color default. To change the color, clear the check box for **Use system colors** and then select a new **Background** color from the palette.

MATLAB Syntax Highlighting Colors

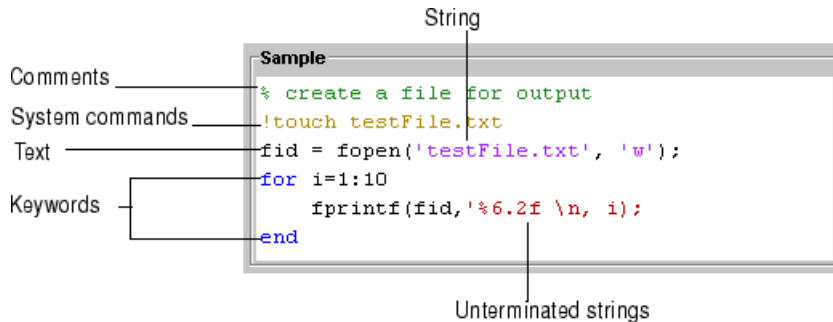
In the Command Window, Command History, Editor, and Shortcuts callback area, MATLAB conveys syntax information via different colors. This helps you to easily identify elements, such as `if/else` statements. This feature is known as syntax highlighting.

In the Command Window, only the input you type is highlighted; the output from running MATLAB functions is not highlighted.

Note Use the Editor/Debugger language preferences to set syntax highlighting colors for files you create for the TLC, C, C++, or Sun Microsystems Java languages, or for HTML, and XML. For details, see “Setting Language Preferences” on page 9-20.

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

When you choose a color under the **MATLAB syntax highlighting colors** area, the **Sample** area indicates how it will look in your file.



The default colors for each element are as follows:

- **Keywords** — Flow control functions, such as `for` and `if`, as well as the continuation ellipsis (`...`), are blue.
- **Comments** — All lines beginning with a `%`, designating the lines as comments in MATLAB, are 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 maroon. When you complete the string with the closing quotation mark (`'`), it becomes purple. For functions you enter using command syntax instead of function syntax, the arguments are highlighted as strings. This is to alert you that in command notation, variables are passed as literal strings rather than as their values. For more information, see “MATLAB Command Syntax” in the MATLAB Programming Fundamentals documentation.
- **Unterminated strings** — A single quote without a matching single quote, and whatever follows the quote, are maroon. This might alert you to a possible error.
- **System commands** — Commands such as `!` (shell escape) are gold.
- **Errors** — Error text that appears after you run code, including any hyperlinks, is red.

Click **Restore Default Colors** to return to the default settings for syntax highlighting colors and desktop tool colors.

Note MATLAB syntax highlighting is enabled by default. If you find it is disabled, follow these steps to reenable it:

- 1** Select **File > Preferences > Editor/Debugger**, and then click **Language**.
 - 2** In the **Language** drop-down menu, select MATLAB.
 - 3** In the **Syntax highlighting** area, select **Enable syntax highlighting**.
 - 4** Click **Apply**.
-

Other Colors

Specify the color for **Hyperlinks**, which applies to links in the Command Window. If you use a dark background color for the Command Window, use a light or other contrasting color for hyperlinks so that you can see them.

See Also

- For information about other preferences and how MATLAB stores preferences, see “Specifying Options for MATLAB Using Preferences” on page 2-124.
- For information about setting color preferences for the code analyzer, variable and syntax highlighting and code cells, see “Setting Color Preferences for Programming Tools” on page 2-154

Setting Color Preferences for Programming Tools

While the Colors preferences enable you to set colors for multiple desktop tools, the Programming Tools color preferences apply to the Editor only. These preferences help you to find and adjust various coding problems, conditions, and structures within your code.

Select **File > Preferences > Colors > Programming Tools** to specify these preferences for MATLAB code files:

- Code analyzer colors
- Variable and function highlighting colors
- Cell display options

Code Analyzer Colors

The code analyzer helps you to identify potential problems and refine your MATLAB code. By default, the Editor indicates:

- Code for which there are warnings, by underlining that code with an orange wavy line and placing an orange line in the message bar.
- Code for which there are errors, by underlining that code with a red wavy line and placing a red line in the message bar.

For information on changing the default color for errors, see “Setting Colors Preferences” on page 2-150.

- Code that MATLAB can fix automatically, by highlighting that code in tan.

For more information, see “Setting Code Analyzer Preferences” on page 9-124 and “Automatically Analyzing Code in the Editor” on page 9-108.

Variable and Function Highlighting Colors

The variable and function highlighting feature helps you to quickly find instances of variables and functions throughout a MATLAB file. This highlighting also helps you to determine how your code is setting, using, and reusing variables and functions, which can help you to avoid variable scoping problems at run time. By default, the Editor indicates:

- A function, subfunction, or local variable, by highlighting all instances of the same function or variable in sky blue when you place the cursor in the name. A line appears in the message bar for each highlighted function or local variable.
- Nonlocal variables, by displaying their names in teal blue.

For more information, see “Navigating an Open File in the Editor” on page 9-71, “Finding and Replacing Functions or Variables in the Current File” on page 9-78, and “Determining Scope and Usage of Functions and Variables” on page 9-135

Cell Display Options

Code cells enable you to evaluate and adjust subsections of your file when running subsections of code to perform iterations. Code cells also enable you to publish polished documents from your code. By default, the Editor indicates:

- The code cell that currently contains the cursor, by highlighting that cell in yellow.
- Cell divisions, by inserting gray lines between each cell in the code. These lines do not appear in the published file or in the printed file.

For more information see “Evaluating Subsections of Files Using Code Cells” on page 9-175 and Chapter 11, “Publishing MATLAB Code”, respectively.

Setting Toolbars Preferences for Desktop Tools

You can customize some toolbars in the MATLAB application using Toolbars preferences. You can add and remove buttons and other controls, as well as change their position on the toolbar.

To customize a toolbar, follow these steps:





- 1** Select **File > Preferences > Toolbars**. You also can access Toolbars Preferences by right-clicking a toolbar, and then selecting **Customize** from the context menu.
- 2** From the **Toolbar** drop-down menu, select the toolbar that you want to customize:
 - **MATLAB** — the toolbar in the MATLAB desktop
 - **Editor** — the toolbar in the MATLAB Editor
 - **Editor Cell Mode** — a specialized toolbar in the Editor. For more information, see “Evaluating Subsections of Files Using Code Cells” on page 9-175.

- **Workspace**, the toolbar in the Workspace browser
- **Current Folder**, the toolbar in the Current Folder browser

The controls for the selected toolbar appear in the **Layout** and **Controls** sections of the Toolbars Preferences pane.

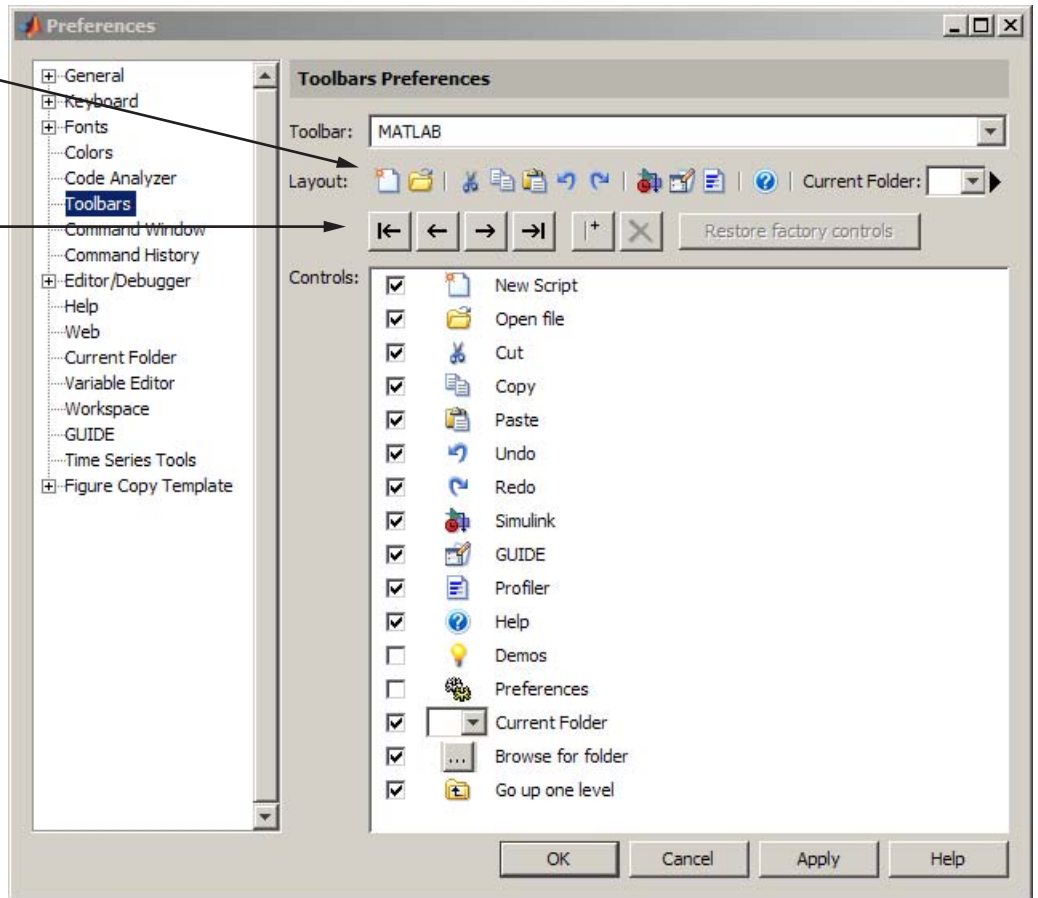
- 3** In the **Controls** list, select or clear the check box for controls that you want to display or remove from the toolbar, respectively.

To show the controls that appeared on the selected toolbar and in the same order as when MATLAB was first installed, click **Restore Factory Controls**.

- 4** In the **Layout** area, rearrange the order of the controls and separator bars on the selected toolbar, by doing any of the following:
 - Drag the icon for a control or separator bar to another position.
 - Select the icon for a control or separator bar, and then click one of the move buttons. For example, select the Demos icon , and then the **Move to the End** button . The Demos icon moves to the right end.
 - Add a separator bar after a selected control by clicking **Add Separator** button: .
 - Remove a control or separator bar, by selecting its icon, and then clicking the **Remove** button .
- 5** Click **Apply** or **OK**. The toolbars in the desktop and Editor update to reflect the changes you made.

Icons representing toolbar controls

Layout buttons



For information about hiding, showing, and moving toolbars, see “Using Toolbar Features” on page 2-110.

Accessibility

In this section...
“Software Accessibility Support” on page 2-159
“Documentation Accessibility Support” on page 2-160
“Assistive Technologies” on page 2-161
“Installation Notes for Accessibility Support” on page 2-162
“Troubleshooting” on page 2-165

Software Accessibility Support

MathWorks products includes a number of modifications to make them more accessible to all users. Software accessibility support for blind and visually impaired users includes:

- Support for screen readers and screen magnifiers, as described in “Assistive Technologies” on page 2-161
- Command-line alternatives for most graphical user interface (GUI) options
- Keyboard access to GUI components
- A clear indication of the current cursor focus
- Information available to assistive technologies about user interface elements, including the identity, operation, and state of the element
- Nonreliance on color coding as the sole means of conveying information about working with a GUI
- Noninterference with user-selected contrast and color selections and other individual display attributes, as well as noninterference for other operating system-level accessibility features
- Consistent meaning for bitmapped images used in GUIs
- HTML documentation that is accessible to screen readers

Keyboard access to the user interface includes support for “sticky keys,” which allow you to press key combinations (such as **Ctrl+C**) sequentially rather than simultaneously.

Except for scopes and real-time data acquisition, MathWorks software does not use flashing or blinking text, objects, or other elements having a flash or blink frequency greater than 2 Hz and lower than 55 Hz.

MathWorks believes that its products do not rely on auditory cues as the sole means of conveying information about working with a GUI. However, if you do encounter any issues in this regard, please report them to the MathWorks Technical Support group.

http://www.mathworks.com/contact_TS.html

Documentation Accessibility Support

Documentation is available in HTML format for all MathWorks products in this release.

Accessing the Documentation

To access the documentation with a screen reader, go to the documentation area on the MathWorks Web site at

<http://www.mathworks.com/access/helpdesk/help/helpdesk.html>

Navigating the Documentation

Note that the first page that opens lists the products. To get the documentation for a specific product, click the link for that product.

The table of contents is in a separate frame. You can use a document's table of contents to navigate through the sections of that document.

Because you will be using a general Web browser, you will not be able to use the search feature included in the MATLAB Help browser. You will have access to an index for the specific document you are using. The cross-product index of the MATLAB Help browser is not available when you are using a general Web browser.

Products

The documentation for all products is in HTML and can be read with a screen reader. However, for most products, most equations and most graphics are not accessible.

The following product documentation has been modified (as described below) to enhance its accessibility for people using a screen reader such as the JAWS® application software from Freedom Scientific BLV Group:

- MATLAB — Many sections, but not the function reference pages. Command line help on functions is accessible, however.
- Spreadsheet Link™ EX
- Optimization Toolbox™
- Signal Processing Toolbox™
- Statistics Toolbox™

Documentation Modifications

Modifications to the documentation include the following:

- Describing illustrations in text (either directly or via links)
- Providing text to describe the content of tables (as necessary)
- Restructuring information in tables to be easily understood when a screen reader is used
- Providing text links in addition to any image mapped links

Equations

Equations that are integrated in paragraphs are generally explained in words. However, most complex equations that are represented as graphics are not currently explained with alternative text.

Assistive Technologies

Note To take advantage of accessibility support features, you must use MathWorks products on a Microsoft Windows platform.

Tested Assistive Technologies

MathWorks has tested the following assistive technologies:

- The JAWS screen reader application software 5.0, 6.0, and 7.0 for Windows platforms from Freedom Scientific
- Built-in accessibility aids from Microsoft, including the Magnifier and “sticky keys”

Use of Other Assistive Technologies

Although MathWorks has not tested other assistive technologies, such as other screen readers or ZoomText® Xtra screen magnifier from Ai Squared, MathWorks believes that most of the accessibility support built into its products should work with most assistive technologies that are generally similar to the ones tested.

If you use other assistive technologies than the ones tested, MathWorks is very interested in hearing from you about your experiences.

Installation Notes for Accessibility Support

Note If you are not using a screen reader such as the JAWS application software, you can skip this section.

This section describes the installation process for setting up your MATLAB environment to work effectively with the JAWS software.

Use the regular MATLAB installation script to install the products for which you are licensed. The installation script has been modified to improve its accessibility for all users.

Note Java Access Bridge 2.0 software from Sun Microsystems is installed automatically when you install MATLAB.

After you complete the product installation, there are some additional steps you need to perform to ensure the JAWS software works effectively with MathWorks products.

Setting Up JAWS Software

Make sure that the JAWS application software is installed on your machine. If it is, there is probably a shortcut to it on the Windows desktop.

Setting up JAWS software involves these tasks:

- 1 Add the Java Access Bridge to your Windows path (for networked installations only).
- 2 Create the `accessibility.properties` file.

These tasks are described in more detail below.

(For Networked Installations Only) Add Java Access Bridge Software to Your Path. If you are running MATLAB in a networked installation environment (that is, if the MATLAB Installer was not run on your machine), you need to take the following steps to add Java Access Bridge to your Windows path.

Note This procedure assumes the **Start** button in your Windows preferences is set to Classic mode. To set Classic mode, from the **Start** button, select **Settings**. Next select **Task Bar and Menu**. Then select the **Start Menu** tab and make sure the **Classic Start Menu** option is enabled. Click **OK** and you are done.

- 1 From the **Start** button, select **Settings**, next select **Control Panel**. Scroll down and click the **System** icon to display the System Properties dialog box.
- 2 In the **System Properties** dialog box, select the **Advanced** tab.
- 3 Click **Environment Variables**.
- 4 Under **System variables**, select the Path option.

- 5 Click the **Edit** button.
- 6 To the start of the Path environment variable, add the directory that contains `matlab.exe`; for example:

```
C:\matlab\bin\win32;
```

Be sure to include that semicolon between the end of this directory name and the text that was already there.

- 7 Click **OK** three times.
- 8 If the JAWS software is already running, exit and restart.

Note The JAWS software must be started with these path changes in effect to work properly with MATLAB.

Create the accessibility.properties File.

- 1 Create a text file that contains the following two lines:

```
screen_magnifier_present=true  
assistive_technologies=com.sun.java.accessibility.AccessBridge
```

- 2 Use the filename `accessibility.properties`.
- 3 Move the `accessibility.properties` file into

```
matlabroot\sys\java\jre\win32\jre1.5.0_07\lib\
```

Pronunciation Dictionary for the JAWS Software. As a convenience, MathWorks provides a pronunciation dictionary for the JAWS application software. This dictionary is in a file called `MATLAB.jdf`.

During installation, the file is copied to your system under the root directory for MATLAB at `sys\Jaws\matlab.jdf`.

To use the dictionary, you must copy it to the `\SETTINGS\ENU` folder located beneath the root installation directory for the JAWS software.

You need to restart the JAWS software and MATLAB for the settings to take effect.

Testing

After you install the JAWS software and set up your environment as described above, you should test to ensure the JAWS software is working properly:

- 1 Start the JAWS software.
- 2 Start MATLAB.

The JAWS software should start talking to you as you select menu items and work with the user interface for MATLAB in other ways.

Troubleshooting

This section identifies workarounds for some possible issues you may encounter related to accessibility support in MathWorks products.

JAWS Software Does Not Detect When Installation of the MATLAB Software Has Started

When you select `setup.exe`, the Windows copying dialog box opens and you are informed. After the files have been copied, the installation splash screen opens, and then the installer starts. However, the JAWS software does not inform you that the installer has begun: the installer either starts up below other windows or applications or it is minimized. Since the installer is not an active item, nothing is read.

Therefore, check the Windows applications bar for the installer. After you go to the installer, you can use the JAWS software to perform the installation.

JAWS Software Stops Speaking

When many desktop components are open, the JAWS software sometimes stops speaking for MATLAB.

If this happens, close most of the desktop components, exit MATLAB, and restart.

Command Output Not Read

In the MATLAB Command Window, the JAWS software does not automatically read the results of commands.

This problem is likely to be caused by the way you have keyboard shortcuts defined in MATLAB. Keyboard shortcuts activate a certain command in MATLAB when you press a key or combination of keys. By default, MATLAB assigns the **Up Arrow** and **Down Arrow** keys to the `Previous History` and `Next History` commands in the Command Window. However, JAWS needs these two keys to be assigned to the `Cursor Up` and `Cursor Down` commands to be able to automatically read the results of commands.

You can check the shortcut setting for the **Up Arrow** key simply by pressing this key while in the Command Window. If the cursor moves up on the screen, then your settings are correct and this is not the cause of the problem. If the **Up Arrow** key displays your most recent command, or performs any action other than moving the cursor up, then follow the steps below to correct the problem.

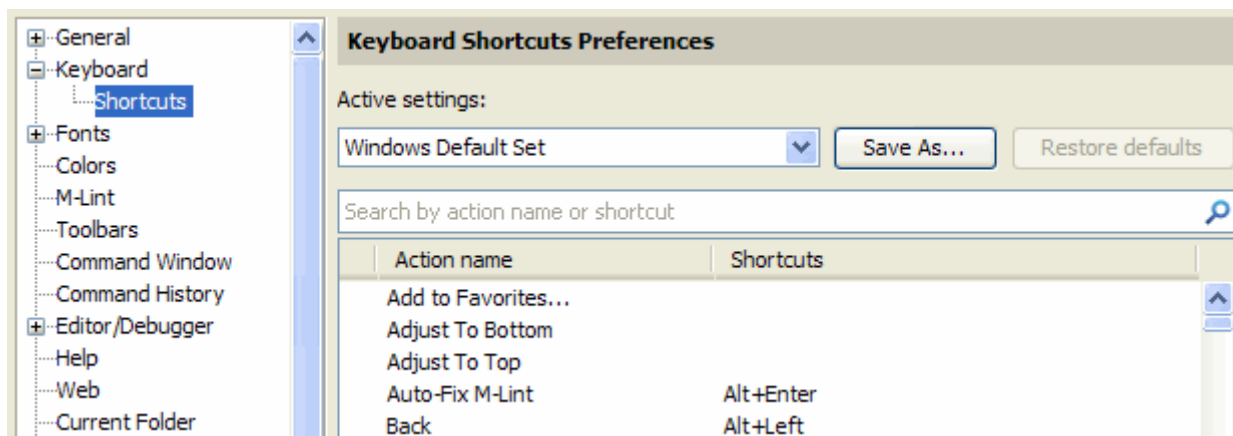
Note Reassigning the arrow key shortcuts means that you will no longer be able to use these keys to activate the `Previous History` and `Next History` commands. You can, however, assign any two unused keys to these actions. See “Reassigning Shortcuts for Previous History and Next History” on page 2-169 in the documentation below.

For more information on either of the procedures described below, see “Performing Desktop Actions Using Keyboard Shortcuts” on page 2-69 in the “Desktop Tools and Development Environment” documentation.

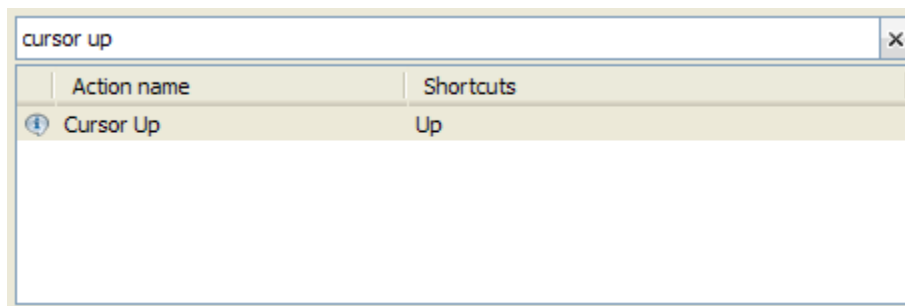
Assigning Shortcuts for Cursor Up and Cursor Down. Follow these steps to make the **Up Arrow** key a shortcut to `Cursor Up` and **Down Arrow** a shortcut to `Cursor Down`. You only need to execute this procedure once. When you close the dialog box with the **OK** button or the **Apply** and then **Cancel** buttons, MATLAB saves your settings and restores them in your next MATLAB session.

- 1 Begin by opening the MATLAB **Preferences** dialog box and clicking **Keyboard** and then **Shortcuts**. This displays the **Keyboard Shortcuts Preferences** dialog box.

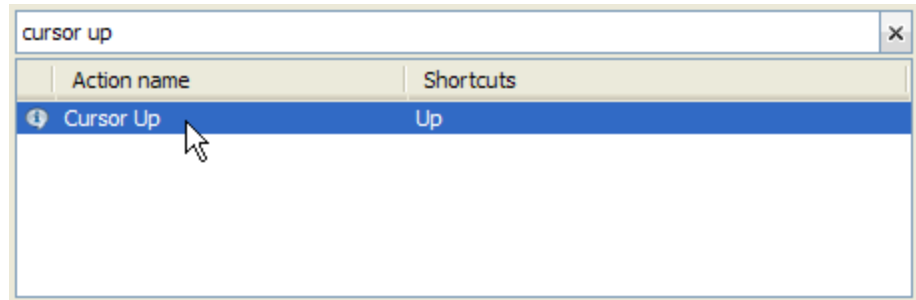
Near the top, the **Action name** and **Shortcuts** columns list all available actions and the shortcut keys currently assigned to them. Above **Action name**, locate the filter field that reads "Search by action name or shortcut" in greyed-out text:



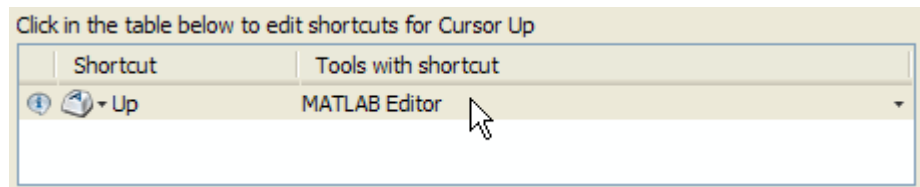
- 2 Enter the words "cursor up" in this field. The **Action name** and **Shortcuts** columns should now show only the one selected action (Cursor Up) and any shortcut keys that have been assigned to that action (Up, in this case):



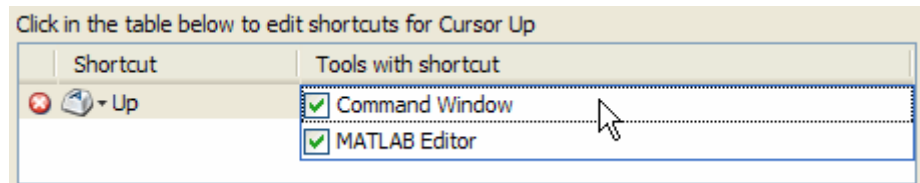
Click the line that shows this action-to-shortcut key association:



- 3 The next panel down shows that the Up (i.e., **Up Arrow**) shortcut is already assigned to the selected action (Cursor Up). However, as shown in the **Tools with shortcut** column, this association applies only in the MATLAB Editor, and not in the Command Window. Click the text under **Tools with shortcut** that reads MATLAB Editor:



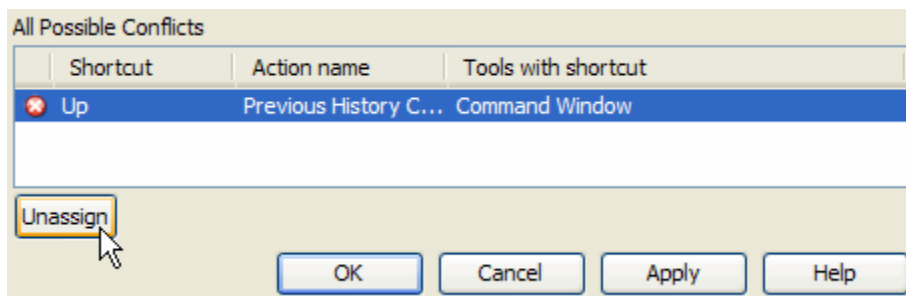
- 4 This displays a checkbox for each application in which you can associate the **Up Arrow** key with Cursor Up. Click the line that reads Command Window to put a check in the checkbox for that selection:



Move the cursor away, and the **Tools with shortcut** column now reads **All Tools**.

- 5 The panel below that, labeled **All Possible Conflicts**, shows that the shortcut you have just established conflicts with an existing shortcut in the Command Window. You now have two actions, Cursor Up and Previous

History Command, assigned to the same key (**Up Arrow**). To resolve this conflict, select the line that shows Previous History Command, and then click the **Unassign** button:

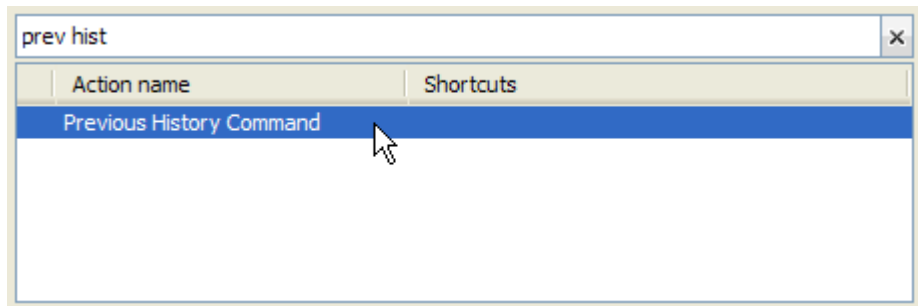



Click **Apply** to make this setting active.

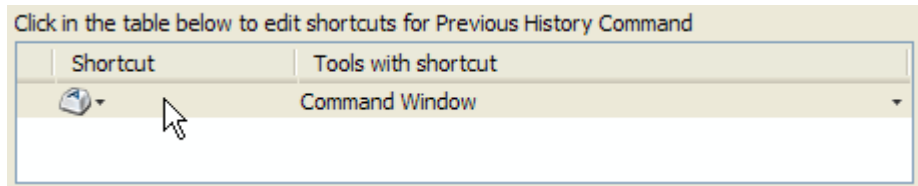
- Repeat steps 2 through 5 to assign the **Down Arrow** key to the Cursor Down command in the Command Window. This also removes the existing shortcut between **Down Arrow** and Next History command.

Reassigning Shortcuts for Previous History and Next History. At this point, you have made assignments for the Cursor Up and Cursor Down desktop actions. The previously defined shortcuts for the Previous History and Next History commands have been removed. If you would like to retain these latter two shortcuts, you need to assign new keys to them. To do this, follow the steps below:

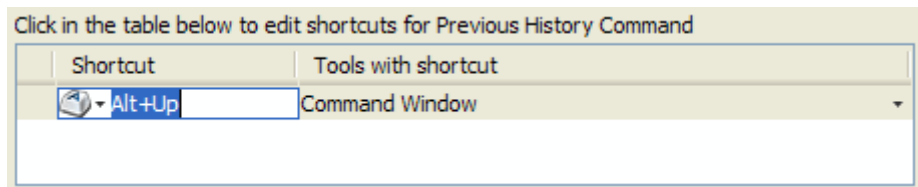
- Type “previous history” in the box above **Action name**. Note that just the abbreviated “prev hist” will suffice. Next, click the line below **Action name** that reads Previous History Command:



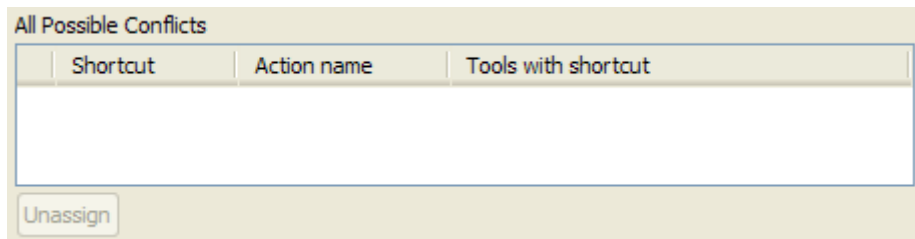
- 2 In the next panel down, click the line that displays the  icon. This opens a key entry field at this location:



- 3 Press the key or key combination you want to use as the shortcut for the Previous History action. For example, if you want **Alt+Up Arrow** to be the shortcut for Previous History, hold down the **Alt** key and press the **Up Arrow** key:



- 4 View the **All Possible Conflicts** panel to see if there are any conflicts with this assignment. If there are no conflicts, click **Apply** to make the setting active. If you do have conflicts, then either choose a different key to use as the shortcut, or see the documentation on “Evaluating and Resolving Keyboard Shortcut Conflicts” on page 2-85 in the “Desktop Tools and Development Environment” documentation.



- 5 Repeat steps 1 through 4 to assign a shortcut key to the Next History command in the Command Window.

Some GUI Menus Are Treated as Check Boxes

For some GUIs (for example, the figure window), menus are treated by the JAWS software as though they are check boxes, whether or not they actually are.

You can choose a menu item for such GUIs by using accelerator keys (e.g., **Ctrl+N** to select **New Figure**), if one is associated with a menu item. You can also use mnemonics for menu navigation (e.g., **Alt+E**).

Note that check boxes that you encounter by tabbing through the elements of a GUI are handled properly.

Text Ignored in Some GUIs

For some dialog boxes, the JAWS software reads the dialog box title and any buttons, but ignores any text in the dialog box.

Also, in parts of some GUIs, such as some text-entry fields, the JAWS software ignores the label of the field. However, the JAWS software will read any text in the text box.

Macintosh Platform – Differences

GUI Conventions in the Documentation and Macintosh Platforms

MATLAB on the Apple Macintosh platform sometimes uses conventions that are standard for the Macintosh platform, but might be different from what the MATLAB documentation states. The documentation typically presents conventions for Microsoft Windows platforms. The intended action for the Macintosh platform is typically obvious. For example, the documentation might instruct you to do the following, which is the convention on Windows platforms:

- 1 Select **File > Save**.
- 2 Select **Yes**, **No**, or **Cancel** from the Save dialog box.

However, on Macintosh platforms, the Save dialog box presents the options **Don't Save** and **Save**.

Pointer Device Instructions and Macintosh Platforms

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

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

On Macintosh platforms, you cannot use a file browser GUI to navigate directly to a file or folder within the MATLAB root folder. (The MATLAB root folder, also called *matlabroot*, is the folder where MATLAB is installed). When you use the Macintosh Finder or a file browser GUI in MATLAB and select `Applications/R2009b_MATLAB`, no contents appear. On Macintosh platforms, the MATLAB root folder is `R2009b_MATLAB.app`, and the Macintosh

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

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

- In the Macintosh Finder, right-click (or **Ctrl**+click) `MATLAB_R2008b`, and from the context menu, select **Show Package Contents**.
- In a MATLAB GUI where you cannot access the contents of `MATLAB_R2008b`, follow these steps:
 - 1** Press **Command+Shift+G** to open the Go To Folder dialog box.
 - 2** In the Go To Folder dialog box, enter the full path to `matlabroot`, for example, `/Applications/MATLAB_R2008b.app/`.
 - 3** Press **OK**.The MATLAB GUI displays the contents of the MATLAB root folder.
- Use an alternative to the GUI. For example, instead of using **File > Open** to open a file in the Editor, use the `edit` function.
- In the Command Window, change the current folder to `matlabroot` by running `cd(matlabroot)`, and then open the GUI. Some GUIs then display the contents of `matlabroot`.

Running Functions — Command Window and History

The Command Window is where you run (execute) MATLAB language statements, while the Command History is a log of the statements you have run.

- “Using the Command Window” on page 3-2
- “Running Functions and Programs, and Entering Variables” on page 3-5
- “Entering Statements in the Command Window” on page 3-17
- “Assistance While Entering Statements” on page 3-23
- “Controlling Output in the Command Window” on page 3-47
- “Finding Text in the Command Window” on page 3-52
- “Preferences for the Command Window” on page 3-60
- “Using the Command History Window” on page 3-66
- “Preferences for Command History” on page 3-78

Using the Command Window

In this section...

“About the Command Window” on page 3-2

“Opening the Command Window” on page 3-2

“Using the Command Window Prompt” on page 3-3

“Changing How the Command Window Looks” on page 3-4

About the Command Window

The Command Window is one of the main tools you use to enter data, run MATLAB code, and display results. If you have an active Internet connection, you can watch the Working in the Development Environment video demo for an overview of the major functionality.

Opening the Command Window

When the Command Window is not open, access it by selecting **Command Window** from the **Desktop** menu. Alternatively, open the Command Window with the `commandwindow` function.

If you prefer a simple command-line interface without the other MATLAB desktop tools visible, select one of the following:

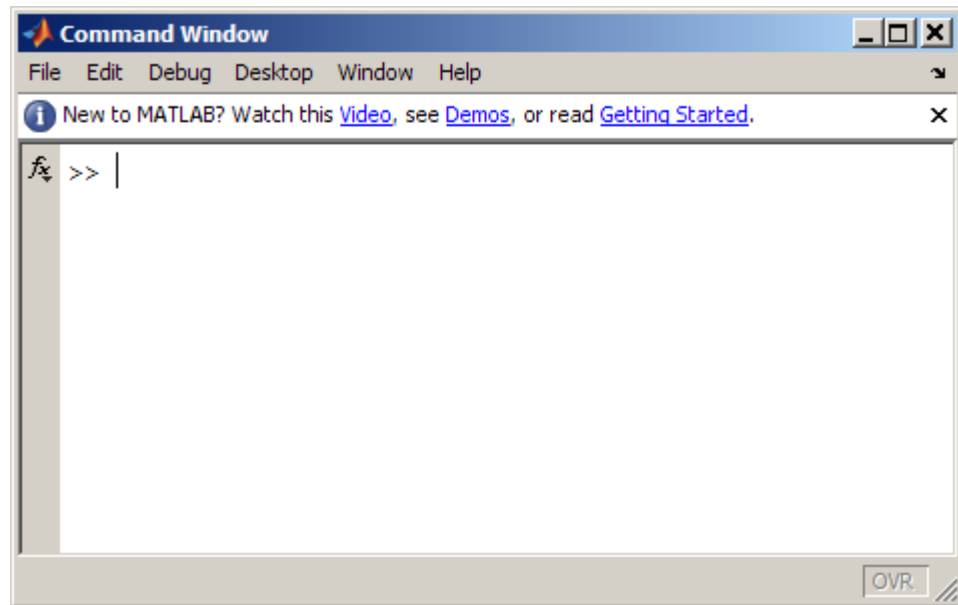
- **Desktop > Desktop Layout > Command Window Only**

The desktop contains only the Command Window, as shown in the image that appears after this list.

- **Desktop > Desktop Layout > All but Command Window Minimized**

All tools open and are minimized in the desktop, except the Command Window, which is maximized. The Editor also remains maximized if it was open and contained a document at the time you chose the **All but Command Window Minimized** menu item.

For more information, see “Opening and Arranging Desktop Tools” on page 2-5.



To restore the desktop to the default arrangement, select **Desktop > Desktop Layout > Default**.

Using the Command Window Prompt

Enter statements at the Command Window prompt. The prompt indicates that MATLAB is ready to accept input from you. This prompt is also known as the command line.

The prompt can be any one of the following:

- `>>` — Indicates that the Command Window is in normal mode.
- `EDU>>` — Indicates that the Command Window is in normal mode, in MATLAB Student Version.
- `K>>` — Indicates that MATLAB is in debug mode.

Type `dbquit` to return to normal mode. For more information, see Chapter 9, “Editing and Debugging MATLAB Code”

Changing How the Command Window Looks

There are various ways you can change how the Command Window looks. The following table provides links to sections that describe how.

Action	See This Section
Move or resize the Command Window.	“Opening and Arranging Desktop Tools” on page 2-5
Change fonts in the Command Window.	“Setting Fonts Preferences for Desktop Tools” on page 2-141
Show and hide buttons, wrap lines, and similar details.	“Preferences for the Command Window” on page 3-60

Running Functions and Programs, and Entering Variables

In this section...

“Running Statements at the Command-Line Prompt” on page 3-5

“Stopping Execution” on page 3-8

“Running External Programs” on page 3-8

“Evaluating or Opening a Selection” on page 3-11

“Displaying Hyperlinks in the Command Window” on page 3-12

Running Statements at the Command-Line Prompt

The following sections describe how to run statements at the prompt and examine errors:

- “Entering Variables and Running Functions” on page 3-5
- “Running MATLAB Program Files Not Provided by MathWorks” on page 3-7
- “Examining Errors” on page 3-7
- “Order of Processing” on page 3-8

Entering Variables and Running Functions

At the prompt, enter data and run functions. For example, to create A, a 3-by-3 matrix, type

```
A = [1 2 3; 4 5 6; 7 8 10]
```

When you press the **Enter** or **Return** key after typing the line, the MATLAB software responds with:

```
A =  
  
     1     2     3  
     4     5     6  
     7     8    10
```

To run a function, it must be in the current folder or in a folder on the search path. By default, functions included with MATLAB are on the search path. Therefore, you do not need to do anything special to run functions provided with MathWorks products.

Type the function including all arguments and press **Enter** or **Return**. MATLAB displays the result. For example, type

```
magic(2)
```

and MATLAB returns

```
ans =  
    1     3  
    4     2
```

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

To find the name of a function that MathWorks provides, use the function browser—see “Finding Functions Using the Function Browser” on page 3-40.

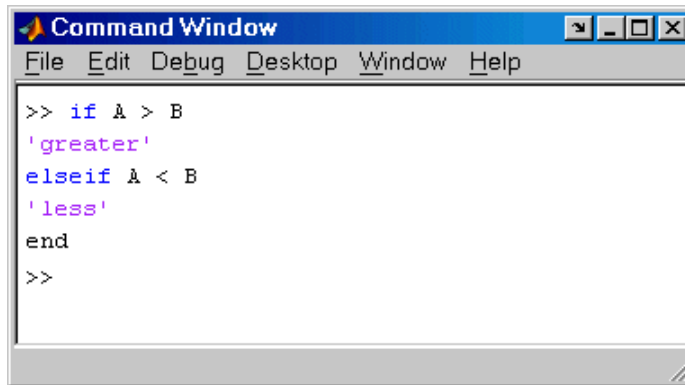
What Is a Statement?. All the information you type before pressing **Enter** or **Return** is known as a statement. Statements can include:

- Variable assignments: For example, `a = 3`
- Functions and their arguments: functions that can accept input arguments and return output arguments, for example, `magic`.
- Commands: functions provided with MATLAB or toolboxes that do not accept input arguments, for example, `clc`, which clears the Command Window.
- Scripts: MATLAB program files you write that do not take input arguments or return output arguments, for example, `myfile.m`.

Some functions support a form that does not require an input argument, thereby operating as commands. For convenience, the term function is used to refer to both functions and commands.

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

following example, you press **Enter** at the end of each line, but the prompt does not appear until you complete the set of statements with `end`.

A screenshot of the MATLAB Command Window. The window title is "Command Window" and it has a menu bar with "File", "Edit", "Debug", "Desktop", "Window", and "Help". The command prompt shows the following code:

```
>> if A > B
'greater'
elseif A < B
'less'
end
>>
```

Running MATLAB Program Files Not Provided by MathWorks

You can run files that contain code in the MATLAB language that you created or that other users created, as follows:

- 1 Ensure that the file is in the current folder or on the search path.

Otherwise, MATLAB returns an `Undefined function or variable` error or a `File not found` error. For more information, see “Determining if MATLAB Can Access a File” on page 7-68.

- 2 Type the name of the file in the Command Window.
- 3 Complete the statement by adding arguments.
- 4 Press **Enter** or **Return**.

For a MATLAB script, you can also use the `run` function and specify the full pathname to a script.

Examining Errors

If an error message appears when you run a file, do one of the following:

- Click the underlined portion of the error message.

- Position the cursor within the file name and press **Ctrl+Enter**.

The offending file opens in the Editor, scrolled to the line containing the error.

Order of Processing

In MATLAB, you can only run one process at a time. If MATLAB is busy running one function, any further statements you issue are buffered in a queue. The next statement runs when the previous statement finishes.

Stopping Execution

To stop execution of whatever is currently running, press **Ctrl+C** or **Ctrl+Break**. On Apple Macintosh platforms, you can also use **Command+.** (the Command key and the period key) to stop the program. For certain operations, stopping the program might generate errors in the Command Window.

For files that run a long time, or that call built-ins or MEX-files that run a long time, **Ctrl+C** does not always effectively stop execution. Typically, this happens on Microsoft Windows platforms rather than UNIX⁶ platforms. If you experience this problem, help MATLAB break execution by including a `drawnow`, `pause`, or `getframe` function in your file, for example, within a large loop. Note that **Ctrl+C** might be less responsive if you start MATLAB with the `-nodesktop` option.

Running External Programs

The exclamation point character (!) sometimes called bang, is a *shell escape* and indicates that the rest of the input line is a command to the operating system. Use it to invoke utilities or call other executable programs without quitting MATLAB. On UNIX platforms, for example, the following code invokes the vi editor for a file named `yearlystats.m`:

```
!vi yearlystats.m
```

After the external program completes or you quit the program, the operating system returns control to MATLAB. Add `&` to the end of the line, such as

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

```
!dir &
```

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

```
!excel.exe &
```

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

Restrictions maintained within the operating system determine the maximum length of the argument list you can provide as input to the bang (!) command. If you are running the Microsoft Windows XP operating system, for example, the length of the argument list input to the bang command cannot exceed 8189 characters.

See the reference pages for the `unix`, `dos`, and `system` functions for details about running external programs that return results and status.

Note To execute operating system commands with specific environment variables, include all commands to the operating system within the `system` call. Separate the commands using `&` (ampersand) for DOS, and `;` (semicolon) for UNIX platforms. This applies to the MATLAB `!` (bang), `dos`, `unix`, and `system` functions. Another approach is to set environment variables before starting MATLAB.

On Macintosh platforms, you cannot run AppleScript (from Apple) directly from MATLAB. However, you can run the Apple Mac OS X `osascript` function from the MATLAB `unix` or `!` (bang) function to run AppleScript from MATLAB.

Running UNIX Programs That Are Off the System Path

You can run a UNIX program from MATLAB when the folder containing that file is not on the UNIX system path that is visible to MATLAB. To determine the system path that is visible to MATLAB, type the following in the Command Window:

```
getenv('PATH')
```

You can make modifications to the system path that persist for the current MATLAB session or across subsequent MATLAB sessions, as described in the sections that follow.

Modify the System Path for the Current MATLAB Session. Do one of the following:

- Change the current folder in MATLAB to the folder that contains the program you want to run.
- Issue these commands using the Command Window:

```
path1 = getenv('PATH')
path1 = [path1 ' :/usr/local/bin']
setenv('PATH', path1)
!echo $PATH
```

If you restart MATLAB, the folder is no longer on the system path visible to MATLAB.

Modify the System Path Across MATLAB Sessions Within the Current Shell Session. To add a folder to the system path from the shell:

- 1 Stop MATLAB.
- 2 Depending on the shell you are using, type one of the following at the system command prompt, where *myfolder* is the folder that contains the program you want to run:

- Type this if you are using `bash` or a related shell:

```
export PATH="$PATH:myfolder"
```

- Type this if you are using `tcsh` or a related shell:

```
setenv PATH "${PATH}:myfolder"
```

- 3 Start MATLAB.
- 4 In the MATLAB Command Window, type:

```
!echo $PATH
```

If you restart MATLAB within the current shell session, the folder remains on the system path visible to MATLAB. However, if you restart the shell session, and then restart MATLAB, the folder is no longer on the system path visible to MATLAB.

Modify the System Path Across All MATLAB Sessions. To make adjustments that persist across shell and MATLAB sessions, add the following commands to the MATLAB startup file as described in “Specifying Startup Options in the MATLABStartup File” on page 1-15:

```
path1 = getenv('PATH')
path1 = [path1 ':/usr/local/bin']
setenv('PATH', path1)
!echo $PATH
```

Evaluating or Opening a Selection

Make a selection in the Command Window and press **Enter** or **Return**. The selection is appended to whatever is at the prompt, and MATLAB executes it.

Similarly, you can select a statement from any MATLAB desktop tool, right-click, and select **Evaluate Selection** from the context menu. Alternatively, for some tools, press **Enter** or **Return**. For example, you can scroll up in the Command Window, select a statement you entered previously, and then press **Enter** to run it. If you try to evaluate a selection while MATLAB is busy, for example, running a file, execution waits until the current operation is done.

You can open a function, file, variable, or Simulink model from the Command Window. Select the name in the Command Window, and then right-click and select **Open Selection** from the context window. This runs the open function for the item you selected so that it opens in the appropriate tool:

- Text files open in the Editor.
- Figure files (.fig) open in a figure window.
- Variables open in the Variable Editor.
- Models open in Simulink software.

See the open reference page for details about what action occurs if there are name conflicts. If no action exists to work with the selected item, **Open selection** calls `edit`.

Function Alternatives

- To open a file in the Editor, use `open` or `edit`.
- To display a file in the Command Window, use `type`.

Displaying Hyperlinks in the Command Window

You can use MATLAB functions to create hyperlinks in the Command Window. The created hyperlink can:

- Open an HTML page in a MATLAB Web browser using an `href` string
- Transfer files via the file transfer protocol (FTP)
- Run one or more MATLAB functions by prefixing them with `matlab:` (called *matlabcolon* syntax)

Creating Hyperlinks to Web Pages

When creating a hyperlink to a Web page, append a full hypertext string on a single line as input to the `disp` or `fprintf` command. For example, the following command:

```
disp('<a href = "http://www.mathworks.com">The MathWorks Web Site</a>')
```

displays the following hyperlink in the Command Window:

```
The MathWorks Web Site
```

When you click this hyperlink, a MATLAB Web browser opens and displays the requested page.

Transferring Files Using FTP

To create a link to an FTP site, enter the site address as input to the `disp` command as follows:

```
disp('<a href = "ftp://ftp.mathworks.com">The MathWorks FTP Site</a>')
```


This command displays the following as a link in the Command Window:

```
The MathWorks FTP Site
```

When you click the link, a MATLAB browser opens and displays the requested FTP site.

Running MATLAB Functions from Hyperlinks

The special keyword `matlab:` lets you embed commands in other functions. Most commonly, the functions that contain it display hyperlinks, which execute the commands when you click the hyperlink text. Functions that support `matlab:` syntax include `disp`, `error`, `fprintf`, `help`, and `warning`.

Use `matlab:` syntax to create a hyperlink in the Command Window that runs one or more functions. For example, you can use `disp` to display an executable hyperlink as follows:

```
disp(' <a href="matlab:a=3; b=4;c=hypot(a,b)">Hypotenuse</a>')
```

displays in the Command Window as

```
Hypotenuse
```

Clicking the hyperlink executes the three commands following `matlab:`, resulting in

```
c =  
    5
```

Executing the link creates or redefines the variables `a`, `b`, and `c` in the base workspace.

The argument to `disp` is an `<a href>` HTML hyperlink. Include the full hypertext string, from `' <a href= to '` within a single line, that is, do not continue a long string on a new line. No spaces are allowed after the opening `<` and before the closing `>`. A single space is required between `a` and `href`.

You cannot directly execute `matlab:` syntax. That is, if you type

```
matlab:a=3; b=4;c=hypot(a,b)
```

you receive an error, because MATLAB interprets the colon as an array operator in an illegal context:

```
??? matlab:a=3; b=4;c=hypot(a,b)
```

```
|  
Error: The expression to the left of the equals sign  
is not a valid target for an assignment.
```

You do not need to use `matlab:` to display a live hyperlink to the Web. For example, if you want to link to an external Web page, you can use `disp`, as follows:

```
disp('<a href="http://en.wikipedia.org/wiki/Hypotenuse">Hypotenuse</a>')
```

The result in the Command Window looks the same as the previous example, but instead opens a page at en.wikipedia.org:

```
Hypotenuse
```

Using `matlab:`, you can:

- “Run a Single Function” on page 3-14
- “Run Multiple Functions” on page 3-14
- “Provide Command Options” on page 3-15
- “Include Special Characters” on page 3-15

Run a Single Function. Use `matlab:` to run a specified statement when you click a hyperlink in the Command Window. For example, run this command:

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

It displays this link in the Command Window:

```
Generate magic square
```

When you click the link, MATLAB runs `magic(4)`.

Run Multiple Functions. You can run multiple functions with a single link. For example, run this command:

```
disp('a href="matlab: x=0:1:8;y=sin(x);plot(x,y)">Plot x,y</a>')
```

It displays this link in the Command Window:

[Plot x,y](#)

When you click the link, MATLAB runs this code:

```
x = 0:1:8;
y = sin(x);
plot(x,y)
```

Redefine `x` in the base workspace:

```
x = -2*pi:pi/16:2*pi;
```

Click the hyperlink, `Plot x,y` again and it changes the current value of `x` back to `0:1:8`. The code that `matlab:` runs when you click the `Plot x,y` defines `x` in the base workspace.

Provide Command Options. Use multiple `matlab:` statements in a file to present options, such as

```
disp('<a href = "matlab:state = 0">Disable feature</a>')
disp('<a href = "matlab:state = 1">Enable feature</a>')
```

The Command Window displays the links that follow. Depending on which link you click, MATLAB sets `state` to 0 or 1.

[Disable feature](#)
[Enable feature](#)

Include Special Characters. MATLAB correctly interprets most strings that include special characters, such as a greater than symbol (`>`). For example, the following statement includes a greater than symbol (`>`).

```
disp('<a href="matlab:str = ''Value > 0''">Positive</a>')
```

and generates the following hyperlink.

[Positive](#)

Some symbols might not be interpreted correctly and you might need to use the ASCII value for the symbol. For example, an alternative way to run the previous statement is to use ASCII 62 instead of the greater than symbol:

```
disp('<a href="matlab:str=['Value ' char(62) ' 0']">Positive</a>')
```

Entering Statements in the Command Window

In this section...

“Case and Space Sensitivity” on page 3-17

“Cut, Copy, Paste, and Undo Features” on page 3-18

“Entering Multiple Lines Without Running Them” on page 3-18

“Entering Multiple Functions in a Line” on page 3-20

“Entering Multiple-Line (Long) Statements Using Line Continuation” on page 3-20

“Recalling Previous Lines in the Command Window” on page 3-21

“Navigating Above the Command Line” on page 3-22

“See Also” on page 3-22

Case and Space Sensitivity

Briefly, MATLAB code is sensitive to casing, but insensitive to blank spaces. For details, see the following sections:

- “Upper and Lowercasing for Variables, Files, and Functions” on page 3-17
- “Spaces in Expressions” on page 3-18

Upper and Lowercasing for Variables, Files, and Functions

In MATLAB code, use an exact match with regard to case for variables, files, and functions. For example, if you have a variable `a`, you cannot refer to that variable as `A`. It is a best practice to use lowercase only when naming functions. This is especially useful when you use both Microsoft Windows and UNIX⁷ platforms because their file systems behave differently with regard to case.

Note that if you use the `help` function, the help displays function names in all uppercase, for example, `PLOT`, solely to distinguish a function name from the rest of the text. Some functions for interfacing to Sun Microsystems Java

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

software do use mixed case and the command-line help and the documentation accurately reflect that.

Spaces in Expressions

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

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

Cut, Copy, Paste, and Undo Features

To edit text in the Command Window, choose **Cut**, **Copy**, **Paste**, **Undo** and **Redo** from the **Edit** menu.

Undo applies to some of the actions listed in **Edit** menu. You can perform an undo operation multiple times in succession. **Redo** reverses an **Undo**.

Press the **Esc** (escape) key to clear everything you have entered on the current line.

If you press **Enter**, you cannot edit a line after entering it, even though you have not completed the flow. In that event, use **Ctrl+C** to end the flow, and then enter the statements again.

Entering Multiple Lines Without Running Them

To enter multiple statements on multiple lines before running any of the statements:

- 1 Type a statement on a line, and then use the keyboard shortcut for **Break Line Without Code Execution** (which is **Shift+Enter** by default).

The cursor moves down to the next line, which does not show a prompt.

- 2 Type the next statement.

- 3 Repeat steps 1 and 2 until you have typed all the statements you want.

4 Edit the statements, if needed.

Then, to run all of the lines, press **Enter** or **Return**.

For example, if you enter the following:

```
>> a=1 % Press Shift+Enter to advance without executing this statement.
b=2   % Press Shift+Enter to advance without execution. You can edit this or the above line.
c=a+b % Press Enter to execute all three statements.
```

MATLAB executes all three lines and returns the following:

```
a =
    1
b =
    2
c =
    3
>>
```

When you enter a paired keyword statement on multiple lines, such as **for** and **end**, you do not need to use **Shift+Enter**. You can use the typical process of pressing **Enter** after each line in the set to advance to the next line. MATLAB executes the keyword statement after you complete it on the last line. For example:

```
>> for r=1:5 % Press Enter. MATLAB advances a line where you continue the paired keyword statement.
a=pi*r^2    % Press Enter. MATLAB advances a line where you continue the paired keyword statement.
end         % Press Enter to execute the paired keyword statement.
```

MATLAB executes all three lines and returns the following:

```
a =
    3.141592653589793
a =
   12.566370614359172
a =
   28.274333882308138
```

See also “Performing Desktop Actions Using Keyboard Shortcuts” on page 2-69.

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 suppresses the output for the command preceding the semicolon. For example, type three functions on one line to build a table of logarithms, and then press the **Enter** or **Return** key:

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

MATLAB runs the functions in order, from left-to-right.

Entering Multiple-Line (Long) Statements Using Line Continuation

To enter a multiple-line statement:

- 1** At the end of the line, indicate that the statement continues on the next line by entering three periods (. . .), also called dots, stops, or an ellipsis.

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.

- 2** Press the **Enter** or **Return** key.
- 3** Continue typing the statement on the next line.
- 4** Repeat steps 1 through 3 until you complete the statement.
- 5** Press **Enter** or **Return** when you complete the statement.

For items in single quotation marks, such as strings, you must complete the string in the line on which it was started. For example, completing a string as shown here

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


results in

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

MATLAB produces an error when you do not complete the string, as shown here:

```
headers = ['Author Last Name, Author First Name, ...  
Author Middle Initial']  
  
??? headers = ['Author Last Name, Author First Name, ...  
Error: Missing variable or function.
```

Recalling Previous Lines in the Command Window

Assuming you have not changed the default keyboard shortcuts for the arrow, tab, and control keys, you can recall, edit, and reuse functions you typed earlier by using these keys. For example, suppose you mistakenly enter

```
rho = (1+ sqrt(5))/2
```

Because you misspelled `sqrt`, MATLAB responds with

```
Undefined function or variable 'sqrt'.
```

Instead of retyping the entire line, press the up arrow \uparrow key. The previously typed line is redisplayed. Use the left arrow key to move the cursor, add the missing `r`, and press **Enter** or **Return** to run the line. Repeated use of the up arrow key recalls earlier lines, from the current and previous sessions. Using the up arrow key, you can recall any line maintained in the Command History window.

Similarly, specify the first few characters of a line you entered previously and press the up arrow key to recall the previous line. For example, type the letters `pl0` and then press the up arrow key. This displays the last line that started with `pl0`, as in the most recent `plot` function. Press the up arrow key again to display the next most recent line that began with `pl0`, and so on. Then press **Enter** or **Return** to run the line. This feature is case sensitive.

If the keys do not behave as documented here, check the actions currently assigned to them, as described in “Displaying Keyboard Shortcuts” on page 2-75.

Navigating Above the Command Line

To look at or copy information in the Command Window that is above the command-line prompt (`>>`), use the mouse and scroll bar, key combinations such as **Ctrl+Home**, and search features. By default, the up and down arrow keys recall statements, and so by default, you cannot use these keys to move the cursor when it is above the command line.

To use the up and down arrow keys to move the cursor when it is above the command line, customize the keyboard shortcuts for the **Cursor Up** and **Cursor Down** actions in the Keyboard Shortcuts preferences. See also “Customizing Keyboard Shortcuts” on page 2-79.

See Also

- “Assistance While Entering Statements” on page 3-23
- “Finding Text in the Command Window” on page 3-52

Assistance While Entering Statements

In this section...

“Highlighting Syntax to Help Ensure Correct Entries” on page 3-23

“Matching Delimiters (Parentheses)” on page 3-24

“Completing Statements in the Command Window — Tab Completion”
on page 3-24

“Viewing Function Syntax Hints While Entering a Statement” on page 3-33

“Getting Help for a Function Shown in the Command Window or Editor”
on page 3-38

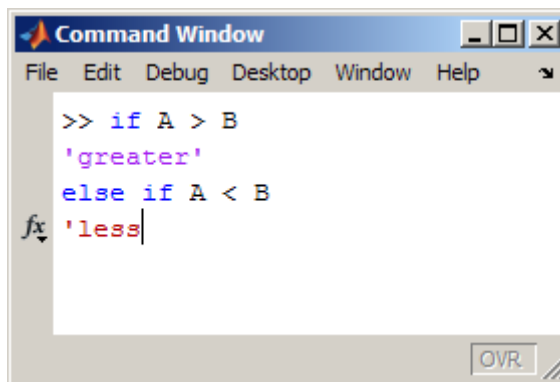
“Finding Functions Using the Function Browser” on page 3-40

“See Also” on page 3-46

Highlighting Syntax to Help Ensure Correct Entries

To help you find elements, such as matching `if/else` statements, some entries appear in different colors in the Command Window. This is known as syntax highlighting. By default:

- Keywords are blue.
- Strings are purple.
- Unterminated strings are maroon.



The screenshot shows a window titled "Command Window" with a menu bar (File, Edit, Debug, Desktop, Window, Help) and a text area containing the following code:

```
>> if A > B
    'greater'
else if A < B
fx 'less|
```

The code is syntax-highlighted: the keywords `if` and `else if` are blue, the strings `'greater'` and `'less'` are purple, and the unterminated string `'less'` is maroon. A cursor is positioned at the end of the unterminated string. A button labeled "OVR" is visible in the bottom right corner of the window.

Except for errors, output in the Command Window does *not* appear with syntax highlighting.

For information on changing the syntax highlighting colors, see “Setting Colors Preferences” on page 2-150.

Matching Delimiters (Parentheses)

You can instruct the MATLAB software to notify you about matched and unmatched delimiters. For example, when you type a parenthesis, bracket, or brace, MATLAB highlights the matched delimiter in the pair. To use the delimiter matching feature, select **File > Preferences > Keyboard > Delimiter Matching**.

For more information, see “Setting Delimiter Matching Preferences” on page 2-139.

Completing Statements in the Command Window — Tab Completion

MATLAB helps you complete the names of known items as you type them in the Command Window so that you can avoid spelling mistakes and can avoid looking up the information in other tools. To use tab completion:

- 1 Ensure that the tab completion preference for the Command Window is selected.

For details, see “Setting Keyboard Preferences for Desktop Tools” on page 2-138.

- 2 Type the first few characters of the item name, and then press the **Tab** key.

These are the items for which MATLAB can complete the names:

- Functions or models on the search path or in the current folder
- MATLAB objects
- File names and folders, including object-oriented programming package and class folders

- Variables, including structures, in the current workspace
- Handle Graphics property names

These examples demonstrate how to use tab completion in the Command Window:

- “Basic Example — Unique Completion” on page 3-25
- “Multiple Possible Completions” on page 3-26
- “Tab Completion for Folders and File Names” on page 3-28
- “Tab Completion for Class Folders and File Names” on page 3-29
- “Tab Completion for Structures” on page 3-31
- “Tab Completion for Handle Graphics Properties” on page 3-32
- “Tab Completion for MATLAB Objects” on page 3-32

Basic Example — Unique Completion

This example illustrates a basic use for tab completion. After creating a variable, `costs_march`, type

```
costs
```

and then press **Tab**. MATLAB automatically completes the name of the variable, displaying:

```
costs_march
```

Then complete the statement, adding any arguments, operators, or options, and press **Return** or **Enter** to run it. In this example, if you press **Enter**, MATLAB displays the contents of `costs_march`. If MATLAB does not complete the name `costs_march` but instead moves the cursor to the right, you do not have the preference set for tab completion. If MATLAB displays `No Completions Found`, `costs_march` does not exist in the current workspace.

You can use tab completion anywhere in the line, not just at the beginning. For example, if you type

```
a = cost
```

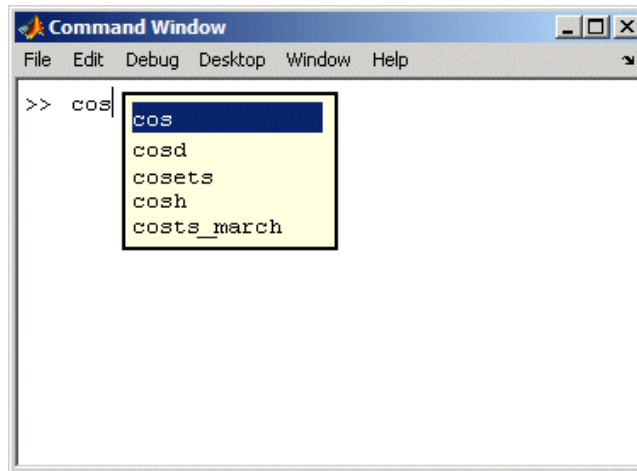
and press **Tab**, MATLAB completes `costs_march`. You can also select `co` or position the cursor after `co` and press **Tab** to complete `costs_march`.

Multiple Possible Completions

If there is more than one name that starts with the characters you typed, then when you press the **Tab** key, MATLAB displays a list of all names that start with those characters. For example, type

```
cos
```

and press **Tab**. MATLAB displays the following.



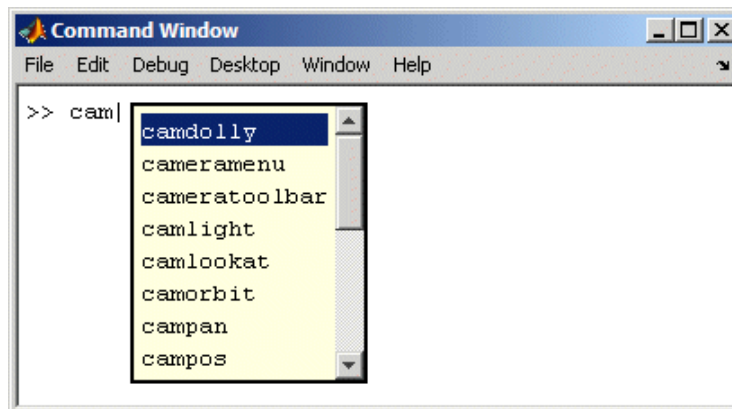
The resulting list of possible completions includes the variable name you created, `costs_march`, and functions that begin with `cos`, including `cosets` from the Communications Toolbox™ software, if it is installed on the system and on the search path in MATLAB. MATLAB completes variable names in the currently selected workspace, and the names of functions and models on the search path or in the current folder.

Continue typing to make your entry unique. For example, type the next character, such as `t` in the example. MATLAB selects the first item in the list that matches what you typed, in this case, `costs_march`. Press **Enter** (or

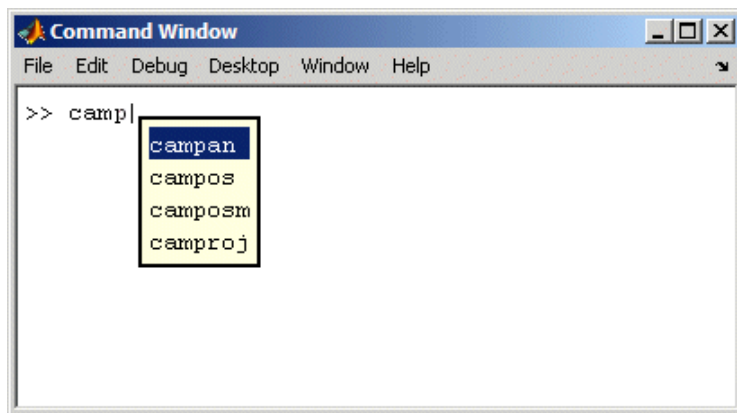
Return) or **Tab** to select that item, which completes the name at the prompt. In the example, MATLAB displays `costs_march` at the prompt. Add any arguments, and press **Enter** again to run the statement.

You can navigate the list of possible completions using up and down arrow keys, and **Page Up** and **Page Down** keys. You can clear the list without selecting anything by pressing **Esc** (escape key). Note that the list of possible completions might include items that are not valid commands, such as private functions.

Narrowing Completions Shown. You can narrow the list of completions shown by typing a character and then pressing **Tab** if the Command Window preference **Tab key narrows completions** is selected. This is particularly useful for large lists. For example, type `cam` and press **Tab** to see the possible completions. There is a scroll bar with the list because there are too many completions to be seen at once.



Type `p` and press **Tab** again. MATLAB narrows the list, showing only all possible `camp` completions.



Continue narrowing the list in the same way. For the above example, type **o** and press **Tab** to further narrow the list. Press **Enter** or **Return** to select an item, which completes the name at the prompt.

Tab Completion for Folders and File Names

Tab completion works for folders and file names in MATLAB functions.

For example, type

```
edit d:/
```

and press **Tab**.

MATLAB displays the list of folders and files in `d`, from which you can choose one. Continue by typing

```
my_M
```

and press **Tab**.

MATLAB displays

```
edit d:/my_MATLAB_files/
```


where `my_MATLAB_files` is the only folder on your d drive whose name begins with `my_M`. Continue using tab completion to display and complete folder names or file names until you finish the `edit` statement.

Tab completion for folders and file names is not supported for functions you write.

Tab Completion for Class Folders and File Names

Tab completion for class folders (including @-folders), package folders, and file names works the same as for standard folders.

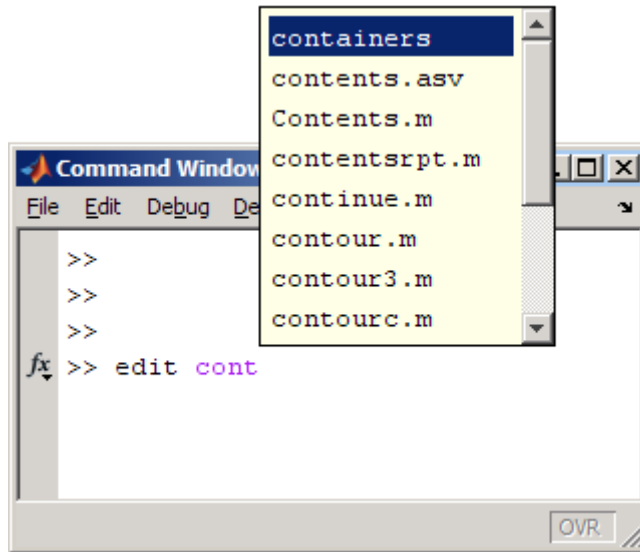
For example, consider the `containers` package in the `matlabroot` folder, which is the folder in which MATLAB is installed. The `containers` package contains a `Map @-` folder and a `Map.m` file. The folder structure appears as follows:

```
+containers\@Map\Map.m
```

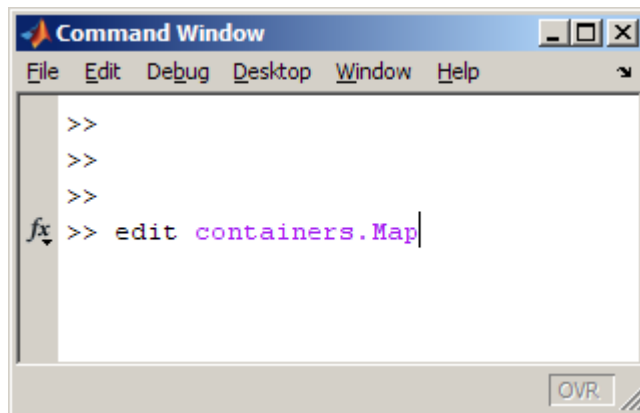
If you type

```
edit cont
```

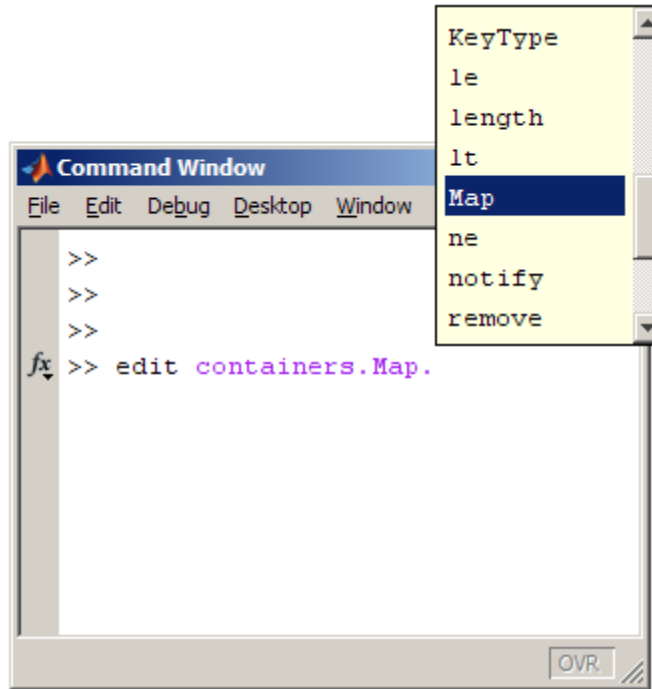
and press **Tab**, MATLAB adds characters to `cont` until MATLAB finds a character that is not common to any names on the MATLAB path. MATLAB displays a list from which to choose, such as the following



If you select **containers**, add a dot, and then press **Tab**, MATLAB displays the following.



If you add a dot, press **Tab**, and scroll down a bit, MATLAB displays the following.



Tab Completion for Structures

For structures in the current workspace, after the period separator, press **Tab**. For example, type

```
mystruct.
```

and then press **Tab** to display all fields of `mystruct`. If you type a structure and include the start of a unique field after the period, pressing **Tab** completes that structure and field entry.

For example, type

```
mystruct.n
```

and press **Tab**, which completes the entry `mystruct.name`, where `mystruct` contains no other fields that begin with `n`.

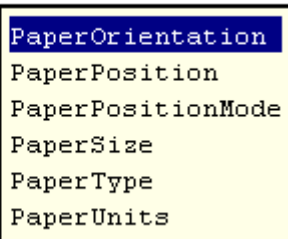
Tab Completion for Handle Graphics Properties

Complete the names of Handle Graphics properties using tab completion, as in this example. Here, `f` is a figure. Type

```
set(f, 'pap
```

and press **Tab**. MATLAB displays

```
set(f, 'paper|
```



Select a property from the list. For example, type

```
u
```

and press **Enter**. MATLAB completes the property, including the closing quote:

```
set(f, 'paperunits'
```

Continue adding to the statement, as in this example:

```
set(f, 'paperunits', 'c
```

and press **Tab**. MATLAB automatically completes the property

```
set(f, 'paperUnits', 'centimeters'
```

because `centimeters` is the only possible completion.

Tab Completion for MATLAB Objects

You can use tab completion with MATLAB objects to select from available properties and methods.

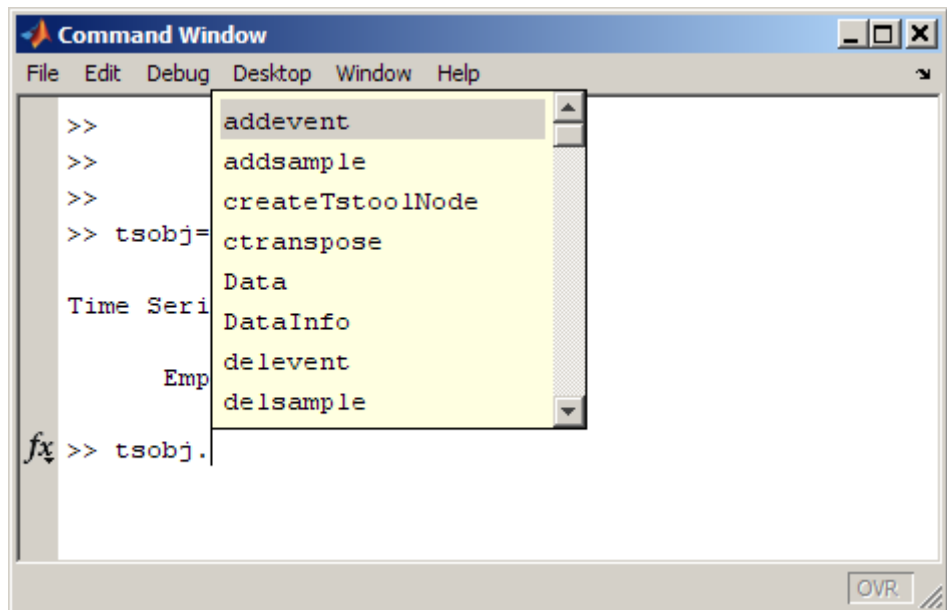
For example, create a time series object, `tsobj`.

```
tsobj = timeseries
```

Then enter the object name followed by a period separator (`.`), such as

```
tsobj.
```

Press **Tab**. MATLAB displays a list of properties and methods for the object, `tsobj`.



Viewing Function Syntax Hints While Entering a Statement

- “What Are Function Hints?” on page 3-34
- “Basic Steps for Using Function Hints” on page 3-34
- “Interpreting Function Hints” on page 3-36
- “Getting More Information While Using Function Hints” on page 3-37

- “Modifying Statements While Using Function Hints” on page 3-37
- “Closing the Pop-Up Window” on page 3-37
- “Enabling or Disabling Function Hints” on page 3-37

What Are Function Hints?

Function hints display allowable input arguments for a function while you enter a statement in the Command Window or Editor. Function hints:

- Appear in a temporary pop-up window.
- Are useful when you only need a reminder of the syntax for a function.
- Provide a link to the reference page for more information.
- Are available for all functions provided with MathWorks products. The syntax comes from the function reference page.
- Work for functions you create. The syntax comes from the function definition statement (first executable line) in the MATLAB program file. The file must be on the search path or in the current folder.

Function hints only display input argument syntax. They do not show output argument syntax. To access output argument syntax while using the function hints, click the **More Help** link to view the reference page.

Basic Steps for Using Function Hints

These steps illustrate using function hints in the Command Window for the `size` function.

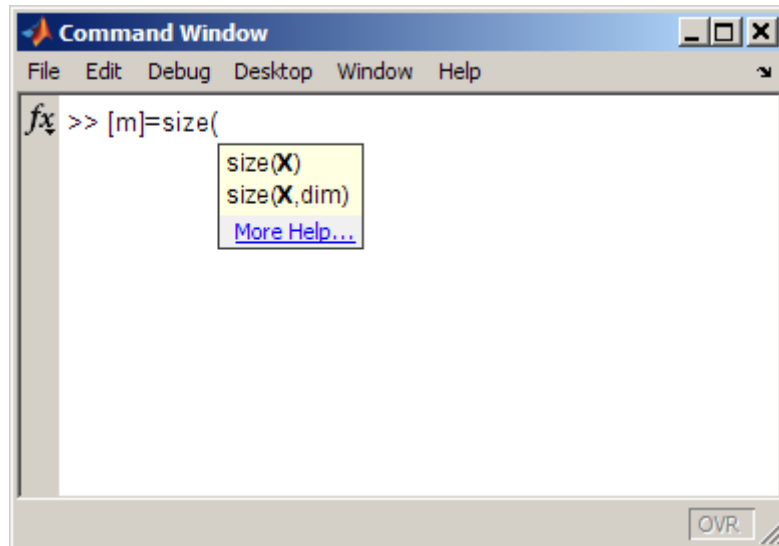
- 1 Type any output arguments and the equal sign. For example, type:

```
[m]=
```

- 2 Type the function name and the opening (left) parenthesis, and then pause. For example, type:

```
size(
```

A yellow pop-up window appears, displaying the syntax options for the function. In this example, the pop-up window indicates that you can enter a single argument, **X**, or two arguments, **X** and **dim**.



- 3 Type a variable name for the first input argument. You can type a variable for any argument that appears in bold in the pop-up.

Note Enter your variable names, and not the argument names shown in the pop-up window.

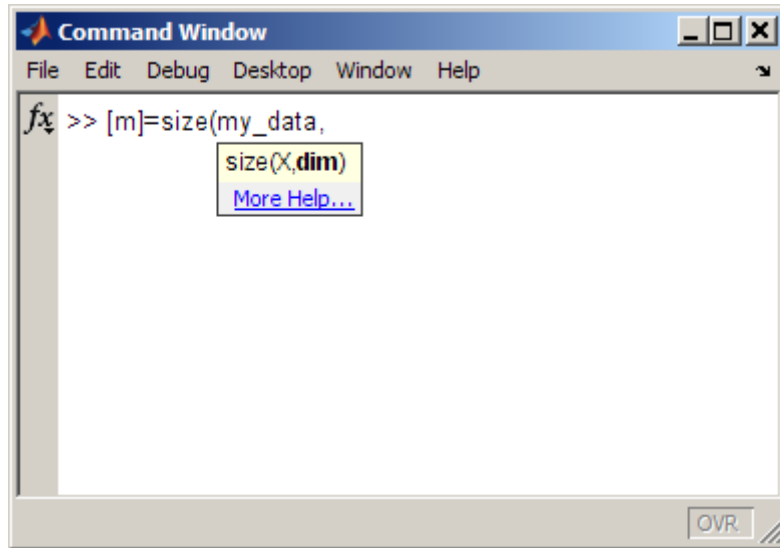
In this example, for the input argument **X**, type your variable:

```
my_data
```

- 4 Enter more arguments:
 - a Type a comma after the argument you just entered.

The syntax options in the pop-up window change, based on the argument you just entered.

For this example, the pop-up window now shows only the `X, dim` option. `dim` is bold now, because you can enter it.



- b** Type the next input argument. It can be any argument that is bold in the pop-up.

For example, to run `size` for the third dimension, enter `3` for `dim`.

Continue entering more input arguments in the same way.

- 5** When you finish entering arguments, type the closing (right) parenthesis.

The pop-up window closes.

Interpreting Function Hints

- For an overloaded function name, the syntax for the method includes valid objects of the method currently in the workspace.
- MATLAB cannot always determine the appropriate hints correctly. Some allowable arguments may not appear, or could be in plain text when they should be bold.

Getting More Information While Using Function Hints

- Click **More Help** in the pop-up window. MATLAB replaces the pop-up window with a small help window containing the complete reference page for a function. See also “Specifying Where Help from the Editor and Function Browser Displays ” on page 4-32.
- Use function hints along with other methods for getting assistance, including tab completion, the Function Browser, and Help on Selection.

Note When you click anywhere, or move the cursor away from the arguments you are entering, the pop-up window automatically closes.

Modifying Statements While Using Function Hints

Modify your statement while the pop-up window is open, and the function hints change based on your modifications. For example, delete a variable you already typed and that argument appears in the pop-up window.

Closing the Pop-Up Window

- Click anywhere, or move the cursor away from the arguments you are entering.
- Close the pop-up window by pressing **Esc** (escape).
- After closing the pop-up window, reopen it by clearing the arguments you entered and clearing the left parenthesis. Then, enter the left parenthesis and pause to redisplay the hints.

Enabling or Disabling Function Hints

To prevent function hints from appearing, or to show them if they are not appearing, use keyboard preferences.

- 1** Select **File > Preferences > Keyboard**.
- 2** In **Function hints**, select the check boxes for the tools where you want hints to appear. Clear the check boxes for the tools where you do not want hints to appear.

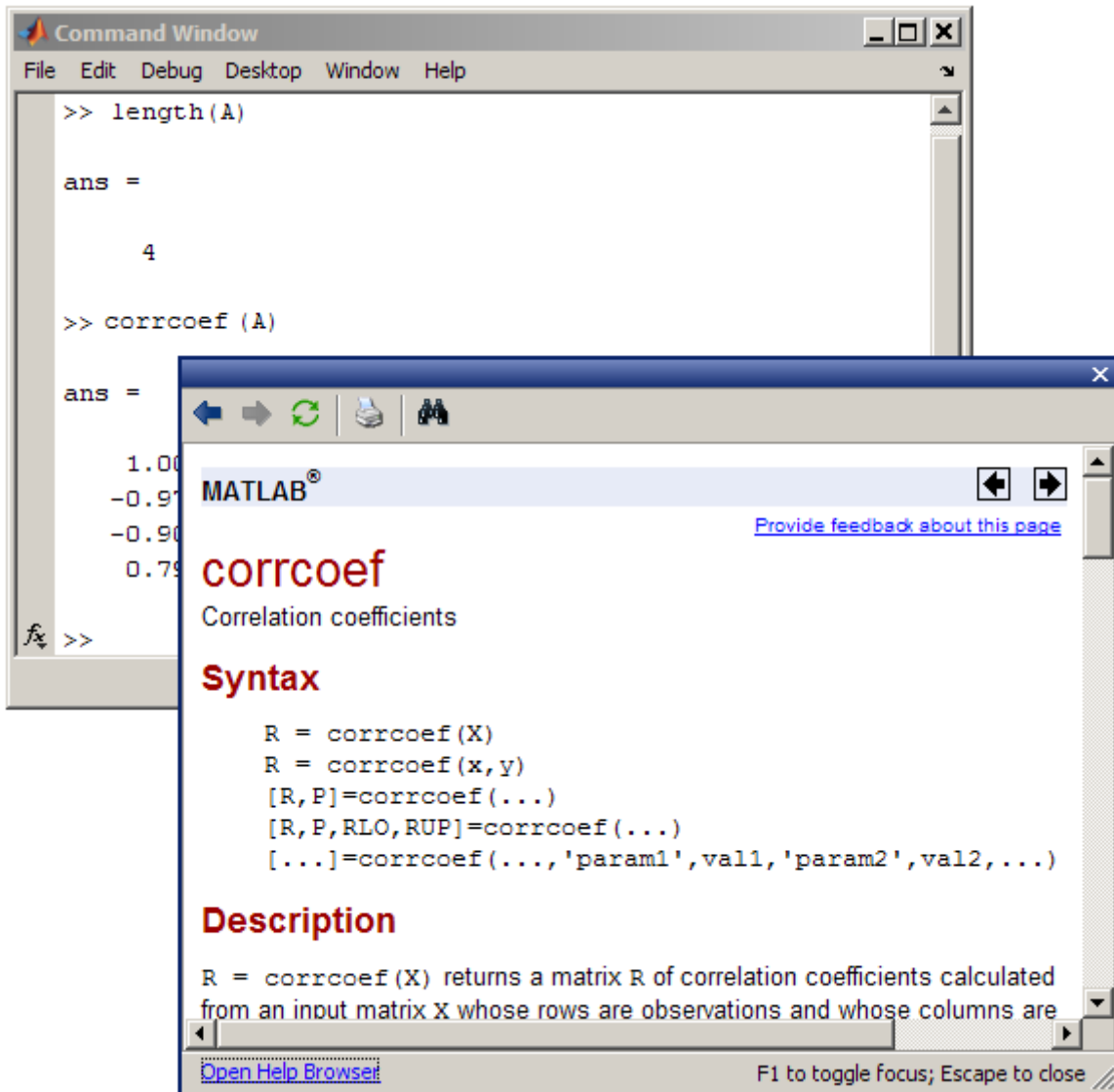
3 Click **OK**.

Getting Help for a Function Shown in the Command Window or Editor

From the Command Window or Editor, display the complete reference page for a function:

1 Right-click in a function name and select **Help on Selection**.

The reference page for the function opens in a small help window. The following illustration shows help on selection for the `corrcoef` function in the Command Window.



2 Close the small help window by pressing **Esc** (escape).

Customize the Help on Selection feature:

- Specify a preference so that help on selection opens the reference page in the Help browser instead of the small help window. To set the preference, select **File > Preferences > Help**.

When help on selection opens the reference page in the Help browser, you cannot toggle focus between the reference page and the Command Window or Editor.

- Specify the products to look in. See “Specifying Which Documentation to Display” on page 4-28 and “Help for Overloaded Functions” on page 4-10.
- Change the small help window font. To change the font, select **File > Preferences > Fonts > Custom**. The small window uses the HTML proportional text font.

Finding Functions Using the Function Browser

- “What Is the Function Browser?” on page 3-40
- “Basic Steps for Using the Function Browser” on page 3-41
- “Interpreting Search Results in the Function Browser” on page 3-45
- “Viewing the Full Reference Page from the Function Browser” on page 3-45
- “Repeating a Search” on page 3-46
- “Customizing the Function Browser” on page 3-46

What Is the Function Browser?

- Provides quick access to the syntax for a function and a description of the syntax.
- Helps you find the names of functions.
- Works in the desktop.


The Function Browser looks for functions using the reference pages for installed MathWorks products.

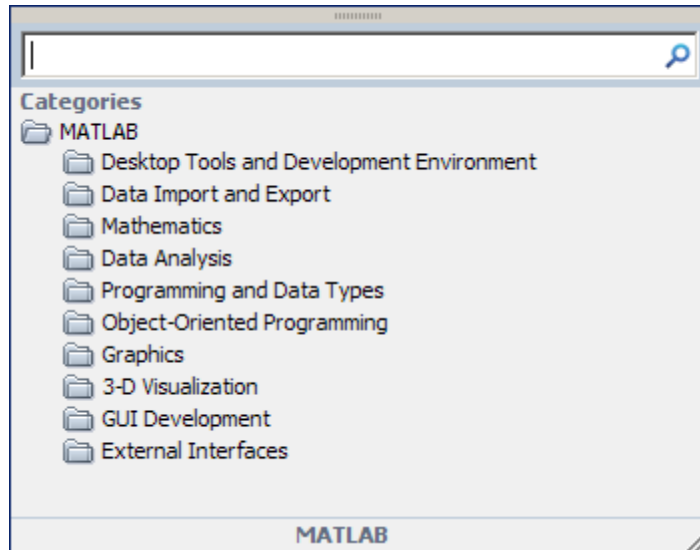
- You cannot use the Function Browser for blocks. Instead use the doc function or the Help browser.

- You cannot use the Function Browser to find functions you created or that other users provided. Instead, use “Finding Files and Folders” on page 7-27.

Basic Steps for Using the Function Browser

- 1 Open the Function Browser by selecting **Help > Function Browser**.

In the Command Window or Editor, another way to open it is by clicking the Browse for functions button, . To show or hide the button, see “Customizing the Function Browser” on page 3-46.



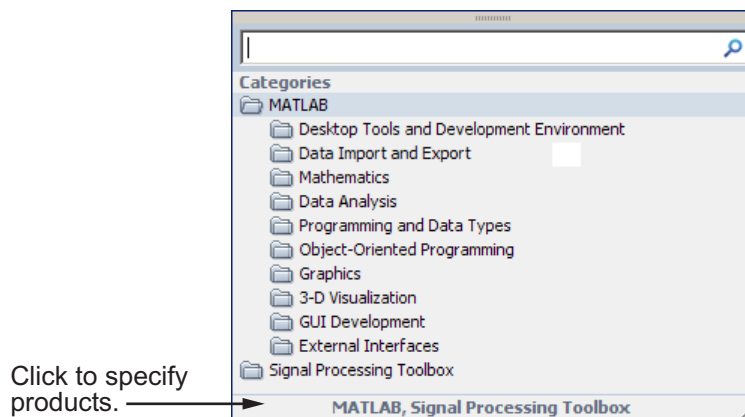
The Function Browser closes when you move the pointer outside of it. To keep the Function Browser open, drag it by the top edge to a different location.

After moving the Function Browser, access it by pressing **Shift+F1**. Close it by pressing **Esc**.

- 2 Restrict the products the Function Browser looks in by using the **Filter by Product** option in Help Preferences.

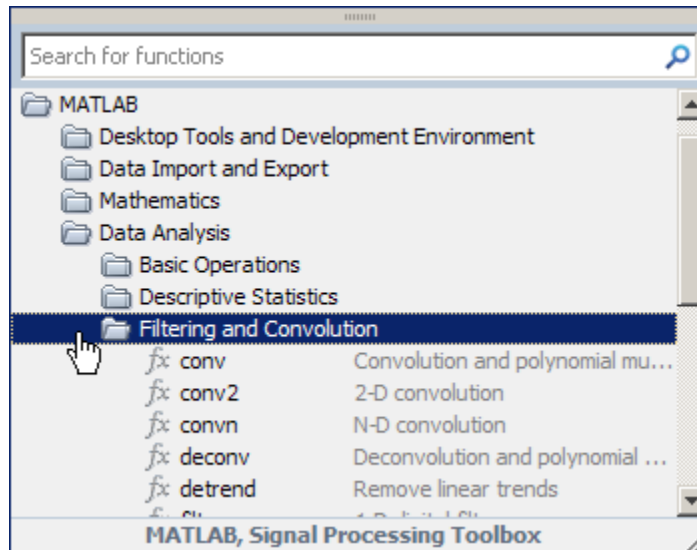
To access the preference, click the product area at the bottom of the Function Browser.

The following illustration shows the filter set to MATLAB and the Signal Processing Toolbox only. The

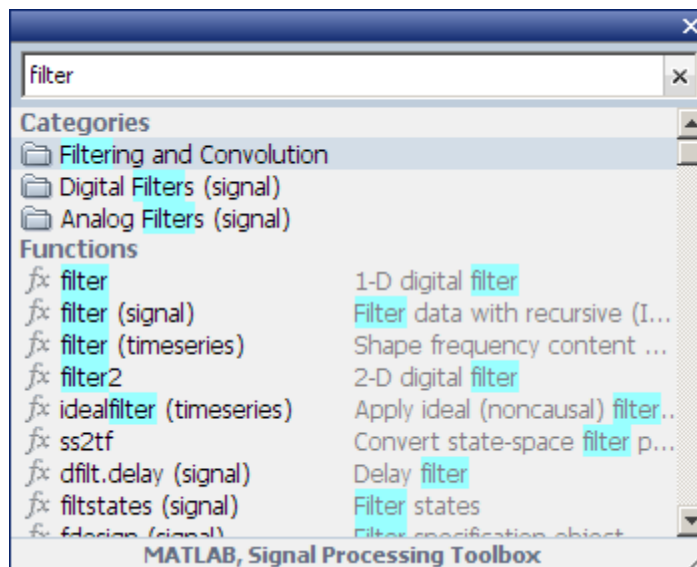


- 3 Find functions by browsing in the list of categories or by typing a search term. You can switch between these two options at any point.

The following illustration shows how you browse for functions by expanding categories of interest.

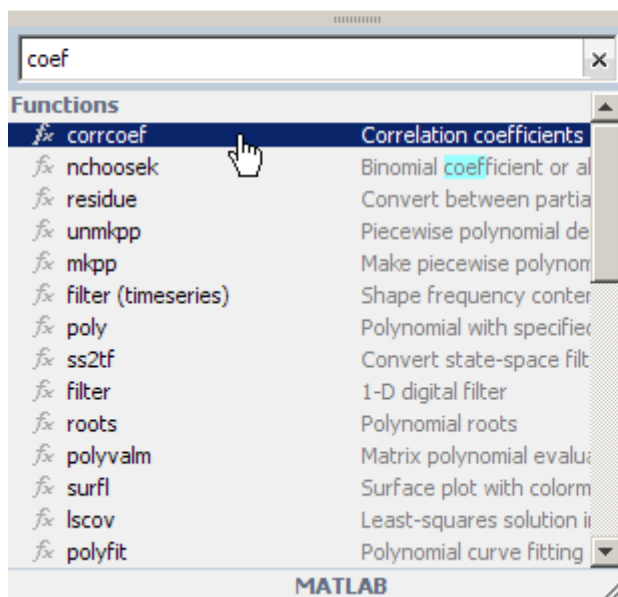


The following illustration shows results when you search, for example, for filter.



- 4 View more information about a function by moving the pointer over a function. A brief description for each of the syntax options displays in a yellow pop-up window.

The pop-up window automatically closes when you move your pointer to a new item in the results list. To keep the pop-up window open, drag it by the top edge to a different location.



corrcoef [More Help...](#)

Correlation coefficients

`R = corrcoef(X)` returns a matrix `R` of correlation coefficients calculated from an input matrix `X` whose rows are observations and whose columns are variables. The matrix `R = corrcoef(X)` is related to the covariance matrix `C = cov(X)` by

$$R(i, j) = \frac{C(i, j)}{\sqrt{C(i, i)C(j, j)}}$$

`corrcoef(X)` is the zeroth lag of the normalized covariance function, that is, the zeroth lag of `xcov(x, 'coeff')` packed into a square array.

- 5 Choose a function you want to use and perform any of the following:

- Add a function name after the cursor in the Command Window or Editor by double-clicking the name in the Function Browser results list.

- Copy and paste the function name into any tool or application by dragging the name from the Function Browser.
- Right-click the function name in the Function Browser to display other options.

Interpreting Search Results in the Function Browser

Parentheses Indicate Location of Function. For results in products other than MATLAB, the product folder appears in parentheses.

When more than one function in MATLAB has the same name, the folder for the overloaded function appears in parentheses.

For example:

- `filter` is in MATLAB
- `filter (signal)` is in the Image Processing Toolbox™
- `filter (timeseries)` is in the `timeseries` folder in MATLAB

Highlights in Search Results. In the results, the search term is highlighted in blue.

When a result does not include any blue highlighting, the matching term is not part of the name, syntax, or brief description. The term is somewhere else in the reference page.

Results for a Two-Letter Search Term. For faster performance, when you type only two letters, the Function Browser looks only for exact matches.

Viewing the Full Reference Page from the Function Browser

View the complete reference page for a function in a small help window by doing either of the following:

- From the Function Browser, right-click the function. From the context menu, select **Help on Function**.
- From the Function Browser pop-up window, click **More Help**.



To open the reference page in the Help browser instead of the small window, set the preference in **File > Preferences > Help**.

Repeating a Search

As you type, the Function Browser displays a history of similar search terms that you previously entered, just below the search field. To perform a search again, select an item from the list and press **Enter**.

To view the entire search history for the current session, press the down arrow key when the search field is empty. To close the history, press **Esc** (escape).

Customizing the Function Browser

- Show or hide the Browse for functions button  in the Command Window using Command Window preferences.
- Show or hide the Browse for functions button  in the Editor using Toolbars preferences for the Editor.
- Change the font used in the Function Browser by selecting **File > Preferences > Fonts**. The Function Browser uses the desktop text font. The pop-up window uses the HTML proportional text font.

See Also

- “Using the Code Analyzer Report” on page 10-22
- “Getting Help for Functions and Blocks” on page 4-8
- Chapter 4, “Getting Help and Product Information”

Controlling Output in the Command Window

In this section...

“Echoing Execution” on page 3-47

“Suppressing Output” on page 3-47

“Paging of Output in the Command Window” on page 3-47

“Formatting and Spacing Numeric Output” on page 3-48

“Number of Characters in Command Window Display” on page 3-49

“Clearing the Command Window” on page 3-50

“Printing Command Window Contents” on page 3-50

“Keeping a Session Log” on page 3-51

Echoing Execution

Echoing command execution is useful for debugging or for demonstrations. It enables you to view commands as they execute. To display each function within a statement as it executes, type `echo on` in the Command Window. For details, see the `echo` reference page.

Suppressing Output

If you end a statement with a semicolon (`;`) and then press **Enter** or **Return**, the MATLAB software runs the statement, but does not display any output. This is particularly useful when you generate large matrices. For example, running the following code creates `A` but does not show the resulting matrix in the Command Window:

```
A = magic(100);
```

Paging of Output in the Command Window

Issue the `more` on command to control the paging of output in the Command Window. This is useful, for example, if output in the Command Window exceeds the visible portion of the window. You then have to scroll backward to review all the output. By default, `more` is `off`.

After you type `more` on, MATLAB displays only a page (a screen full) of output, pauses, and displays

```
--more--
```

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

Key	Action
Enter or Return	To advance to the next line
Space Bar	To advance to the next page
q	To stop displaying the output

Formatting and Spacing Numeric Output

By default, numeric output in the Command Window is displayed as 5-digit scaled, fixed-point values, called the short format. To change the numeric format of output for the current and future sessions, set the Command Window preference for text display. The text display format affects only how numbers are shown, not how MATLAB computes or saves them.

Function Alternative

Use the `format` function to control the output format of the numeric values displayed in the Command Window. The format you specify applies until you change it or until the end of the session.

Examples of Formats

Here are a few examples of the various formats and the output produced from the following two-element vector `x`:

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

format short
    1.3333    0.0000

format short e
    1.3333e+000    1.2345e-006

format +
++
```

A complete list and description of available formats is in the reference page for `format`. For more control over the output format, use the `sprintf` and `fprintf` functions.

Controlling Spacing

To control spacing in the output, use the Command Window preference for text display or the `format` function. Use

```
format compact
```

to suppress blank lines, allowing you to view more information in the Command Window. To include the blank lines, which can help make output more readable, use

```
format loose
```

Number of Characters in Command Window Display

The maximum line length for Command Window display is 25,000 characters. If the output from a statement exceeds this limit, the Command Window truncates the output and displays the following message at the end of the line of output:

```
Output truncated. Text exceeds maximum line length of 25,000
characters for Command Window display.
```

Clearing the Command Window

Clear the Command Window view without clearing the workspace by selecting **Edit > Clear Command Window**.

Afterwards, unless you have changed the keyboard shortcut for it, you can use the up arrow to recall previous functions from the command history.

For more information, see:

- “Confirmation Dialogs Preferences” on page 2-132
- “Customizing Keyboard Shortcuts” on page 2-79

Function Alternative

Use `clc` to clear the Command Window. Or, use the `home` function to move the prompt so that the screen is clear, but you can still scroll up to see the Command Window contents.

Printing Command Window Contents

To print the contents of the Command Window, use one of the methods described in the table that follows.

To	Do This
Print the complete Command Window contents.	Select File > Print .
Print a portion of the Command Window contents.	Select the text you want to print, and then select File > Print Selection .
Specify printing options for the Command Window. (For example, you can specify a header.)	Select File > Page Setup . For more information, see “Printing and Page Setup Options for Desktop Tools” on page 2-115.

Keeping a Session Log

The diary Function

The `diary` function creates a copy of your session in MATLAB on a disk file, including keyboard input and system responses, but excluding graphics. You can view and edit the resulting text file using any text editor, such as the MATLAB Editor. To create a file on your disk called `sept23.out` that contains all the functions you enter, as well as output from MATLAB, type the following:

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

To stop recording the session, type:

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

To view the file, type:

```
edit('sept23.out')
```

Other Session Logs

There are two other means of viewing session information:

- The Command History window contains a log of all functions executed in the current and previous sessions—see “Using the Command History Window” on page 3-66
- The `logfile` startup option—see “Startup Options” on page 1-14.

Finding Text in the Command Window

In this section...

“Introduction” on page 3-52

“Finding Text Currently Displayed in the Command Window” on page 3-52

“Increasing the Amount of Information Available for Searching in the Command Window” on page 3-53

“Using Incremental Search in the Command Window” on page 3-53

Introduction

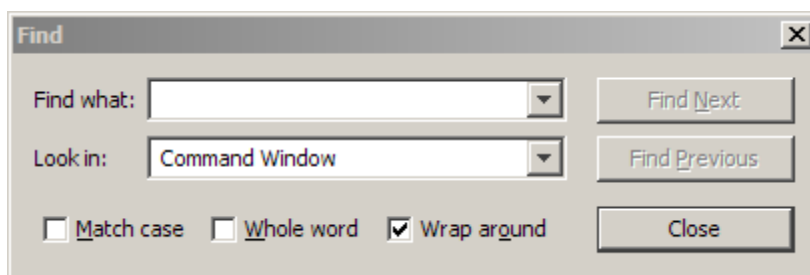
You can search for specified text that appears in the Command Window, where the text is either part of input you supplied, or output displayed by the MATLAB software. After finding the text, you can copy and paste it to the prompt in the Command Window to run it, or into a file.

Finding Text Currently Displayed in the Command Window

To search for specified text currently displayed in the Command Window:

- 1 Select **Edit > Find** when the Command Window is active.

The Find dialog box opens.



- 2 Complete the dialog box, and then click **Find Next** or **Find Previous**.

The search begins at the current cursor position. MATLAB finds the text you specified and highlights it.

3 Repeat step 2 to find another occurrence.

MATLAB beeps when a search for **Find Next** reaches the end of the Command Window, or when a search for **Find Previous** reaches the top of the Command Window. If you have **Wrap around** selected, it continues searching after beeping.

To search for the specified text in other MATLAB desktop tools, change the selection in the **Look in** field.

Increasing the Amount of Information Available for Searching in the Command Window

To increase the amount of information displayed in the Command Window so that more text is available for searching:

1 Select **File > Preferences > Command Window**, and then increase the setting for the “Number of lines in command window scroll buffer” on page 3-64.

2 Do *not* clear the Command Window.

In other words, do not enter `clc` or select **Edit > Clear Command Window**.

Using Incremental Search in the Command Window

With the incremental search feature, the cursor moves to the next or previous occurrence of the specified text in the Command Window. It is similar to the Emacs search feature. The following sections provide details:

- “Example of Using Incremental Search” on page 3-54
- “Summary of Keyboard Shortcuts for Incremental Searches” on page 3-57
- “Case Sensitivity in Incremental Searches” on page 3-58

Example of Using Incremental Search

You control incremental search using keyboard shortcuts. The following example demonstrates how to use the incremental search feature in the Command Window when:

- MATLAB is running on a Windows or UNIX system and the **Active settings** field in the Keyboard Shortcuts Preference dialog box specifies the Emacs Default Set.
- MATLAB is running on a Macintosh system and **Active settings** field in the Keyboard Shortcuts Preference dialog box specifies the Macintosh Default Set.

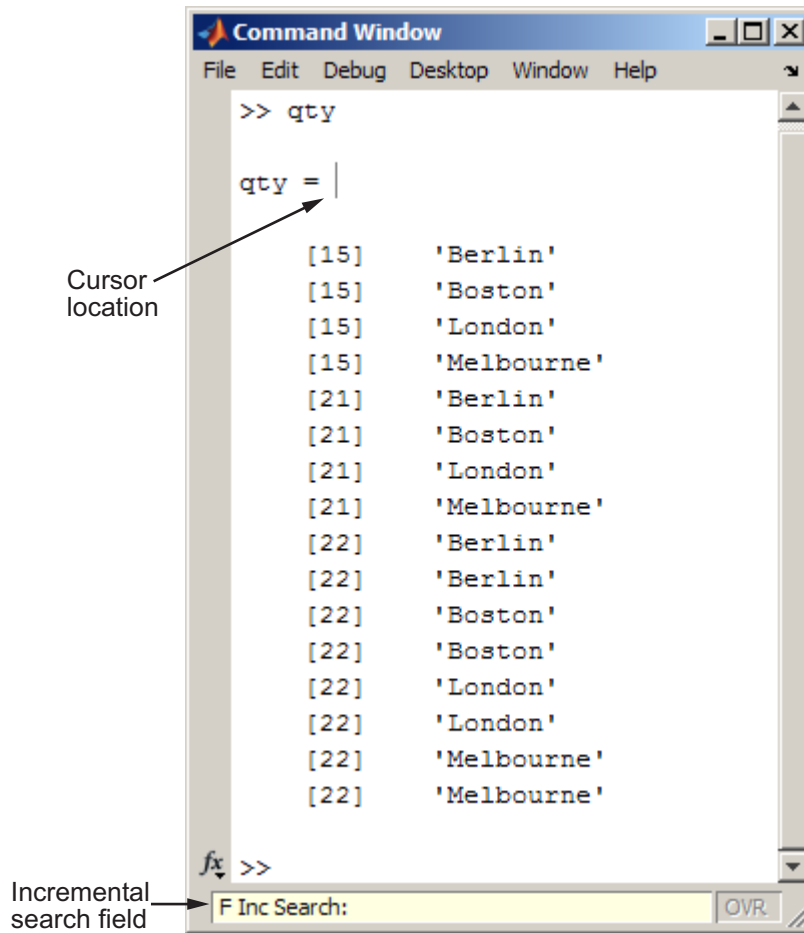
For details, see “Choosing a Set of Keyboard Shortcuts” on page 2-70.

- 1 Clear the Command Window to clear it of existing text and add this text:

```
qty = {15, 'Berlin'; 15, 'Boston'; 15, 'London'; ...  
15, 'Melbourne'; 21, 'Berlin'; 21, 'Boston'; 21, 'London'; ...  
21, 'Melbourne'; 22, 'Berlin'; 22, 'Berlin'; 22, 'Boston'; ...  
22, 'Boston'; 22, 'London'; 22, 'London'; 22, 'Melbourne'; ...  
22, 'Melbourne'}};  
  
clc  
  
qty
```

- 2 In the Command Window, position the cursor after the equal sign (=).
- 3 Begin an incremental search by pressing **Ctrl+S**.

An incremental search field appears at the bottom of the Command Window with the text `F incSearch`. The `F` indicates a forward search.



- 4 Begin a search for Boston by typing b.

The first occurrence of b highlights.



If you mistype in the **Inc Search** field, use the **Backspace** key to remove the previous letter or letters and make corrections.

- 5 Type the next letter of the text you want to find. Type **o** to specify the next letter in Boston.

The **Inc Search** field contains **bo**, and the **bo** in Boston highlights.

- 6 To complete the highlighted word, press **Ctrl+W**.

Boston highlights and appears in the **Inc Search** field.

- 7** Find subsequent occurrences of the word by pressing **Ctrl+S** one or more times.

If MATLAB beeps, it means either that the text was not found, or the search wrapped past the bottom (or top) of the Command Window and is continuing at the top (or bottom).

- 8** Find previous occurrences of the word by pressing **Ctrl+R**.

The incremental search field begins with R to indicate a backward (reverse) search.

- 9** Search for a string that does not appear in the Command Window text by typing **x**.

MATLAB beeps and **Failing** appears in the incremental search field.

- 10** Automatically remove characters back to the last successful search by pressing **Ctrl+G**.

- 11** End incremental searching by pressing **Esc** (escape) or **Enter**.

The **Inc Search** field disappears. The cursor remains at the position where the text was last found, with the search text highlighted.

Summary of Keyboard Shortcuts for Incremental Searches

The following table summarizes the keyboard shortcuts you can use for performing incremental search actions. Except for the keyboard shortcuts that initiate an incremental search, you cannot customize these shortcuts. For information on choosing a keyboard shortcuts **Active settings** file, see “Choosing a Set of Keyboard Shortcuts” on page 2-70.

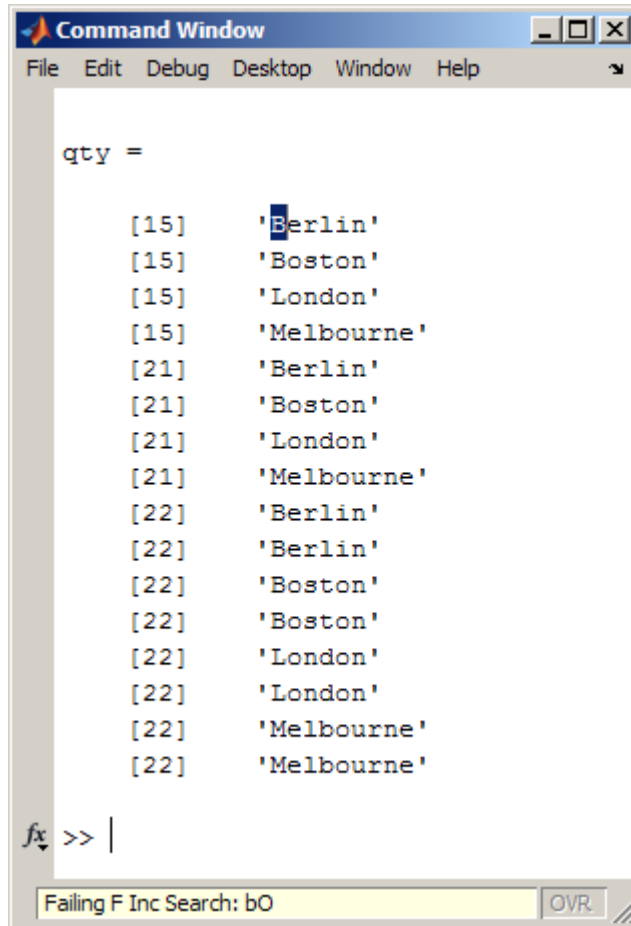
Action	Active Settings File	Keyboard Shortcut
Initiate a forward incremental search.	Windows Default Set	Ctrl+Shift+S
Initiate a forward incremental search.	Emacs Default Set or Macintosh Default Set	Ctrl+S

Action	Active Settings File	Keyboard Shortcut
Initiate a backward incremental search.	Windows Default Set	Ctrl+Shift+R
Initiate a backward incremental search.	Emacs Default Set or Macintosh Default Set	Ctrl+R
Complete a partially highlighted string of characters.	Any	Ctrl+W
Find the next occurrence of a string of characters.	Any	Ctrl+S
Remove characters from the Inc Search field, back to the last successful search	Any	Ctrl+G
End an incremental search.	Any	Esc (escape), or Enter , or any other key that is not a character or number

Case Sensitivity in Incremental Searches

When you enter lowercase letters in the **Inc Search** field, incremental search looks for both lowercase and uppercase instances of the letters. For example, if you enter **b**, incremental search looks for **b** and **B**. However, if you enter uppercase letters, incremental search only looks for instances that match the case you entered.

For example, suppose you enter **b0** in the **Inc Search** field when the Command Window contains the text shown in the image that follows. Incremental search finds the **b** in **Berlin**, but does not find any additional matching text.

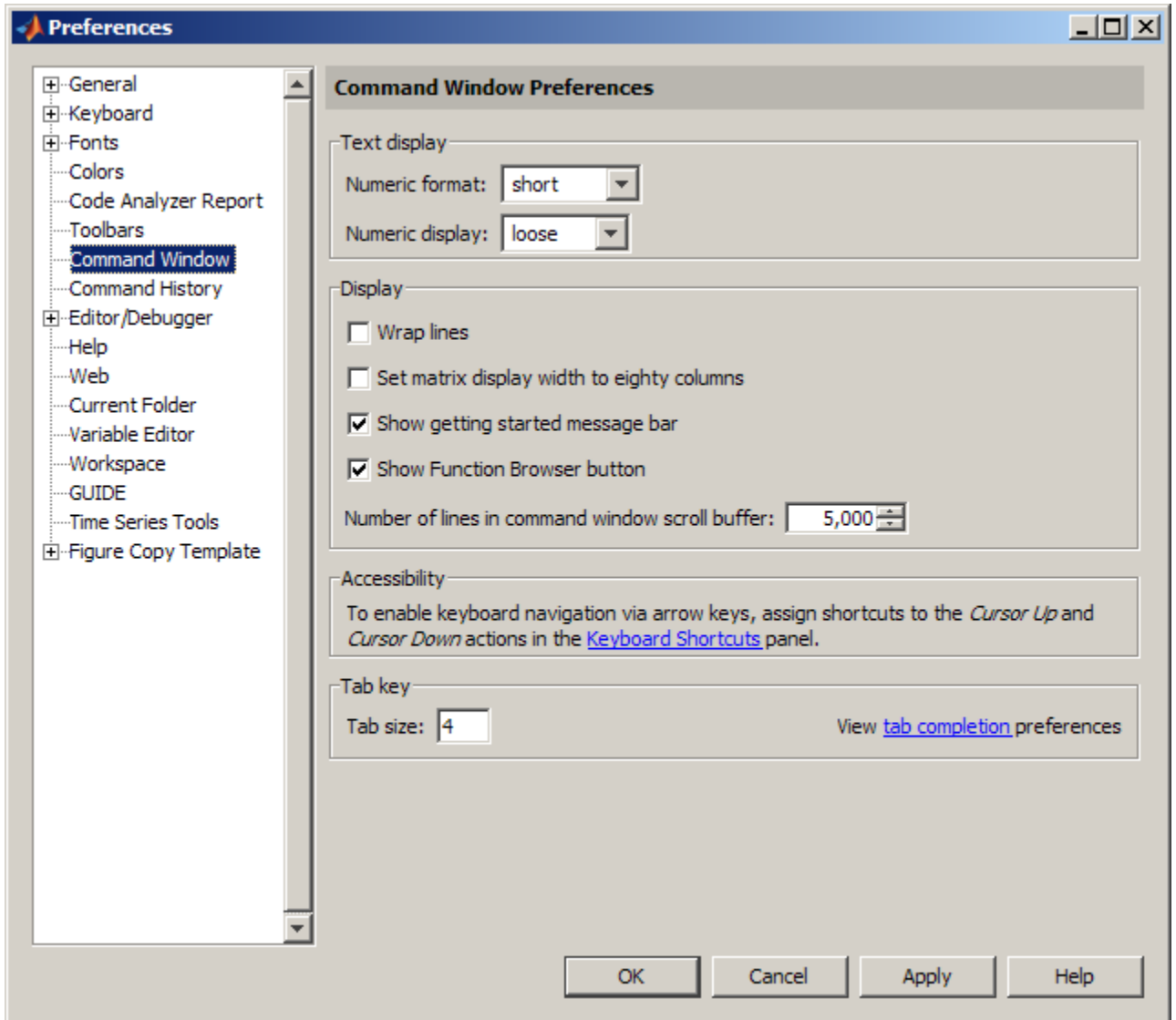


Preferences for the Command Window

In this section...
“Text, Display, Accessibility, and Tab Size Preferences” on page 3-60
“Additional Settings That Affect the Command Window” on page 3-64

Text, Display, Accessibility, and Tab Size Preferences

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



Text Display

Specify the format, that is, how output appears in the Command Window.

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

Numeric display. Specify spacing of output in the Command Window. To suppress blank lines, use `compact`. To display blank lines, use `loose`. For more information, see the reference page for `format`.

Display

Wrap lines. Select to make a single line of input or output in the Command Window break into multiple lines in order to fit within the current width of the Command Window. This is useful for console mode. With this option selected, an entire line is visible without scrolling, and the horizontal scroll bar does not appear because it is not needed. With this option cleared, use the horizontal scroll bar to view the entire contents of the line.

Set matrix display width to eighty columns. When selected, MATLAB displays 80 characters of matrix output in a single row, and then continues displaying output in a new row, regardless of the width of the Command Window. Use the horizontal scroll bar if the width of the Command Window is less than 80 characters.

With the check box cleared, a row of matrix output fills the width of the Command Window, and then continues displaying output in a new row. Note that if the **Wrap lines** preference is also selected, and the width of the Command Window is less than 80 characters, each row of 80 characters of matrix output wraps to fit within the width of the Command Window.

To determine the number of characters and lines that will display in the Command Window, given its current size, use:

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

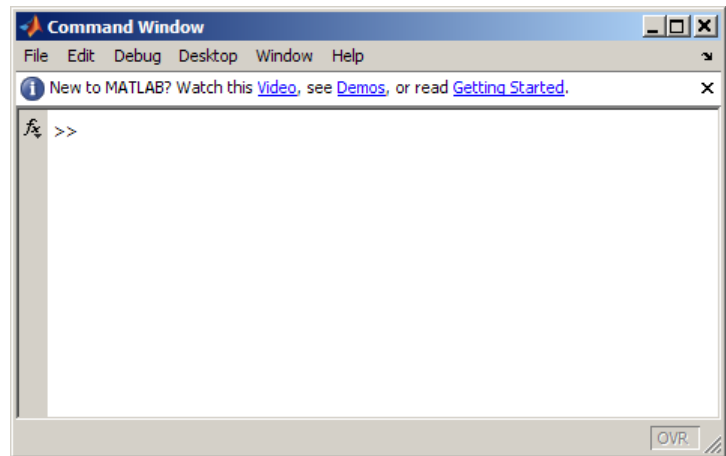
When the matrix display width preference is *not* selected, the number of characters for the width is based on the current width of the Command Window. For example, a result of 50, 25 means 50 characters will display

across the Command Window, and 25 lines will display. However, with the preference selected, the result for that same size Command Window is 80, 25.

Show getting started message bar. The message bar in the Command Window includes links to a video, demos, and information on getting started with MATLAB. If you want to remove the message bar in the Command Window, click the Close box in the right corner of the bar. If you close the message bar, you can still access the documentation and demos it linked to—for more information, see Chapter 4, “Getting Help and Product Information”.

If you closed the message bar and want to show it again, select the **Show getting started message bar** check box in the Command Window **Display** preferences.

Getting started
message bar. →



Show Function Browser button. The Function Browser button f_x appears to the left of the prompt in the Command Window. You use it to access the Function Browser. If you do not want the button to appear because of the space it requires, you can hide it by clearing the **Show Function Browser button** check box. When the button is not shown, you can access the Function Browser by pressing **Shift+F1** or by right-clicking in the Command Window and selecting **Function Browser** from the context menu. For more information about the Function Browser, see “Finding Functions Using the Function Browser” on page 3-40.

Number of lines in command window scroll buffer. Set the number of lines maintained in the Command Window, from 1,000 to 25,000. This is the number of lines you can see when you scroll vertically. A larger buffer means you can view more lines and it provides a larger base for search features, but requires more memory.

This preference setting does not impact the number of lines you can recall when you use the up arrow key in the Command Window (unless you have changed the keyboard shortcut for the up arrow key). By default, you can use the up arrow key, to recall all lines shown in the Command History window, regardless of how many lines you can see in the Command Window.

Accessibility

Click the **Keyboard Shortcuts** link to assign keyboard shortcuts to the **Cursor Up** and **Cursor Down** actions in the Command Window. These actions enable you to move the cursor in the lines above the command line prompt without using the mouse. For more information, see “Customizing Keyboard Shortcuts” on page 2-79. For information about using Command Window keyboard shortcuts and JAWS software, see “Command Output Not Read” on page 2-166.

Tab key

Tab size. Number of spaces assigned to a tab stop when displaying output. The default is four spaces, except on UNIX⁸ platforms where the default is eight spaces. This does not apply when the tab completion preference is selected.

Additional Settings That Affect the Command Window

For information on additional preferences settings that affect the look and behavior of the Command Window, see the following topics:

- “Setting Fonts Preferences for Desktop Tools” on page 2-141

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

- “Confirmation Dialogs Preferences” on page 2-132
- “Setting Keyboard Preferences for Desktop Tools” on page 2-138

Using the Command History Window

In this section...
“Overview of the Command History Window” on page 3-66
“Viewing Statements in the Command History Window” on page 3-68
“Performing Actions on Statements in the Command History Window” on page 3-68
“Searching in the Command History Window” on page 3-70
“Printing the Command History Window” on page 3-76
“Deleting Entries from the Command History Window” on page 3-76

Overview of the Command History Window

The Command History window displays a log of the statements most recently run in the Command Window. If you have an active Internet connection, you can watch the Working in the Development Environment video demo for an overview of the major functionality.

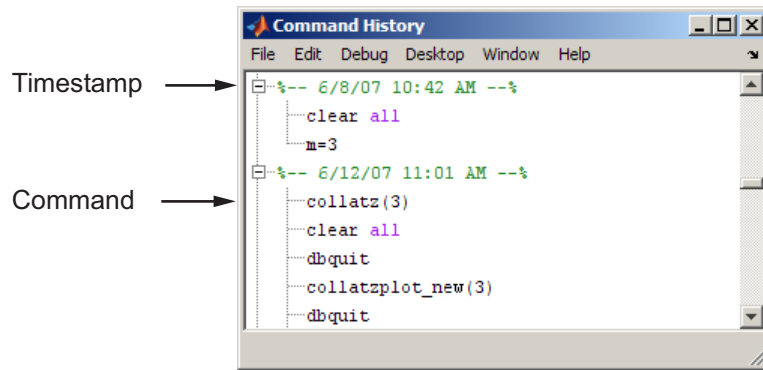
To show or hide the Command History window:

- Select **Desktop > Command Window**.

A check mark next to Command Window indicates it is displaying. No check mark indicates it is hidden.

- Type `commandhistory` in the Command Window.

MATLAB opens the Command History window if it is closed, or selects it if it is open.



In addition, MATLAB provides other options for viewing a history of statements, as described in the following sections:

- “Recalling Previous Lines in the Command Window” on page 3-21
- The diary function reference page
- “Startup Options” on page 1-14, which includes the logfile startup option

Command History File

When you recall lines in the Command History window and the Command Window (as described in “Recalling Previous Lines in the Command Window” on page 3-21), MATLAB uses the command history file, `history.m`.

The history file:

- Resides in the folder returned when you type `prefdir` in the Command Window
- Loads when MATLAB starts
- Stores a maximum of 20,000 bytes
- Deletes the oldest entries, as needed, to maintain the maximum number of bytes

Statements saved to the history are those that run in the Command Window. This includes statements you run using the **Evaluate Selection** item on context menus in tools such as the Editor, Command History, and Help

browser. The history does not include every action taken in MATLAB, however, For example, if you run the statement and then modify the value of `a` in the Variable Editor, there is no record in the history that you modified the value of `a`.

```
a = 1:10
```

MATLAB automatically saves the command history file throughout the session according to the **Saving** preference you specified. You can choose to automatically exclude certain statements from being written to the command history file with the **Settings** preference. For details, see “Preferences for Command History” on page 3-78.

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. The following table summarizes the methods you can use to view statements in the Command History Window.

Action	How to Perform the Action
Move through the Command History window.	Use the mouse to slide the scroll bar or press the Up Arrow and Down Arrow keys.
Hide history for a session.	Click <input type="checkbox"/> or select a timestamp, and then press the - key on the numeric keypad.
Show hidden history for a session.	Click <input type="checkbox"/> or select a timestamp, and then press the + key on the numeric keypad.

Performing Actions on Statements in the Command History Window

You can select entries in the Command History window and then perform the following actions for the selected entries.

Action	How to Perform the Action
Run statements in the Command Window,	<p>Do one of the following:</p> <ul style="list-style-type: none">• Double-click an entry or entries in the Command History window. For example, double-click <code>edit myfile</code> to open <code>myfile.m</code> in the Editor.• Right-click an entry and select Evaluate Selection from the context menu.• Select an entry and press Enter or Return.
Edit and run statements in the Command Window.	<p>Do either of the following:</p> <ul style="list-style-type: none">• Select an entry or entries, select Copy from the context menu, and then paste the selection into the Command Window.• Drag a selection to the Command Window. Then, in the Command Window, edit the statements, and press Enter or Return.
Copy statements to another window.	<p>Do either of the following:</p> <ul style="list-style-type: none">• Select an entry or entries, and then select Copy from the context menu. Paste the selection into an open file in the Editor or any application.• Drag the selection from the Command History window to an open file or another application.

Action	How to Perform the Action
Create a file from a statement or statements.	Select an entry or entries, and then right-click and select Create Script from the context menu. The Editor opens a new file that contains the statements you selected from the Command History window.
Create a shortcut from a statement or statements.	Do either of the following: <ul style="list-style-type: none">• Select an entry or entries, and then right-click and select Create Shortcut from the context menu.• Drag the selection to the Shortcuts toolbar. The Shortcut Editor opens and the selected statements appear in the Callback field. For more information, see “Running Frequently Used Statement Groups with MATLAB Shortcuts” on page 2-57.

Searching in the Command History Window

There are two types of search in the Command History window:

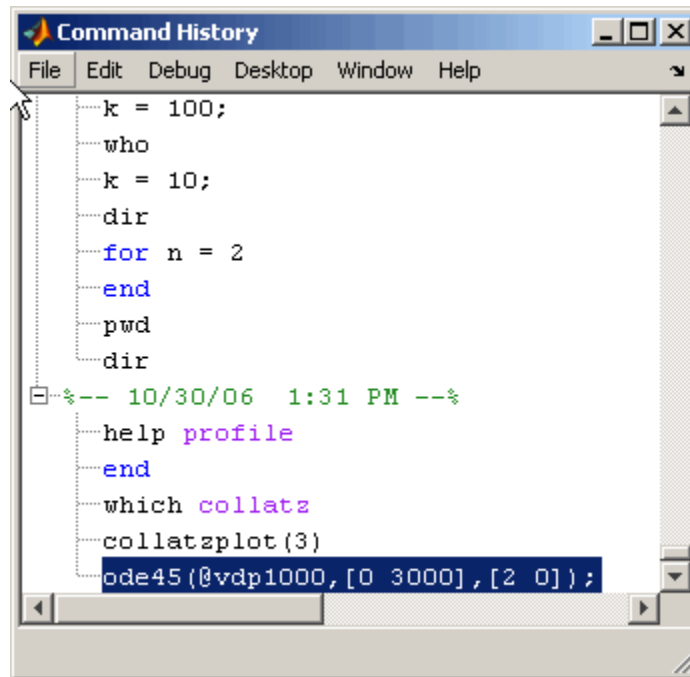
- “Finding Next Entry By Letter” on page 3-70
- “Finding Text” on page 3-75

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

Finding Next Entry By Letter

Type a letter in the Command History window. The Command History window searches backwards to find the last previous entry that begins with that letter as illustrated in this example:

- 1 Position the cursor at anywhere in the Command History window.



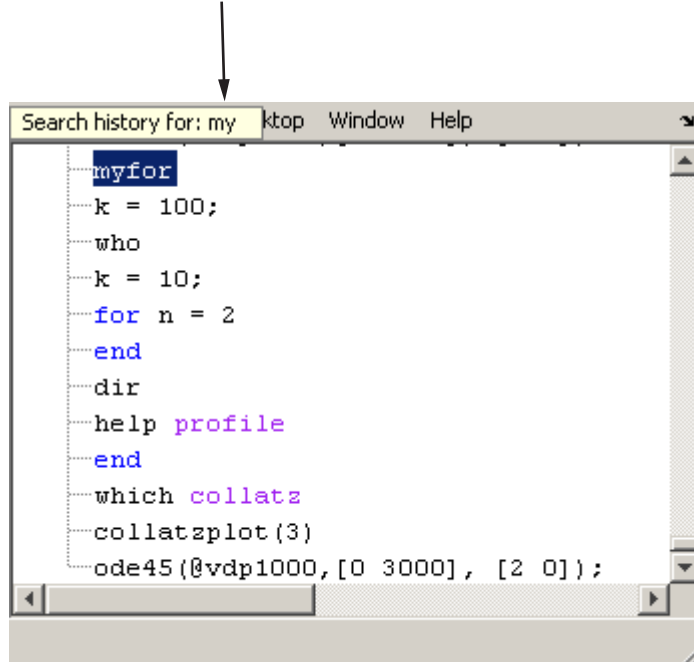
- 2 Type the first letters of the entry you want to find. For example, type my.

The Command History window searches backwards and selects the previous entry that begins with the letters you typed; in this example, you typed my, and the Command History finds myfor.

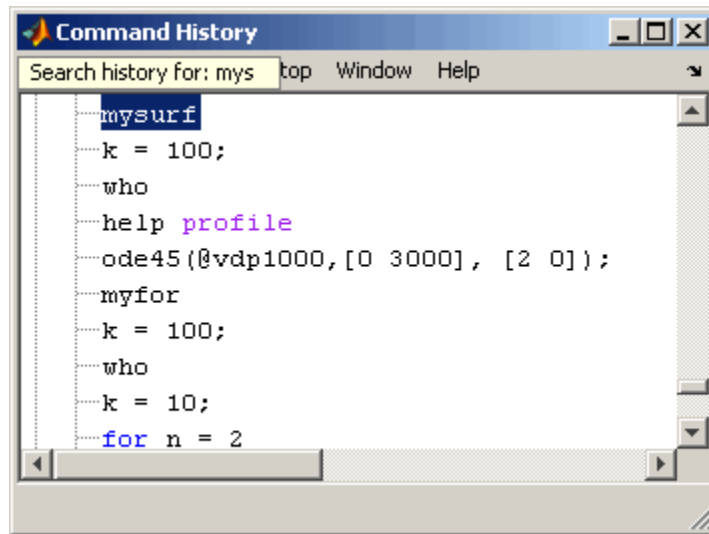
As you begin typing a tooltip with the text: Search history for:, appears at the top of the Command History window. This tooltip keeps track of your search target as you type additional letters to narrow the focus of your search.

If the search finds a matching entry in a session that is collapsed, it expands the session and selects the entry.

Incremental search target. Changes as you type additional letters.

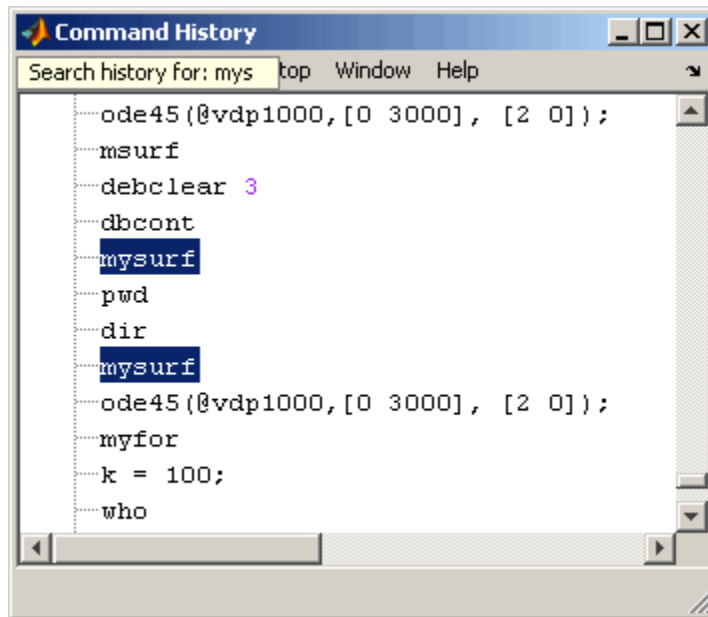


- 3 Type an **s** to extend the search to **mys**. The Command History window continues to search backwards, stopping next at the function **mysurf**.

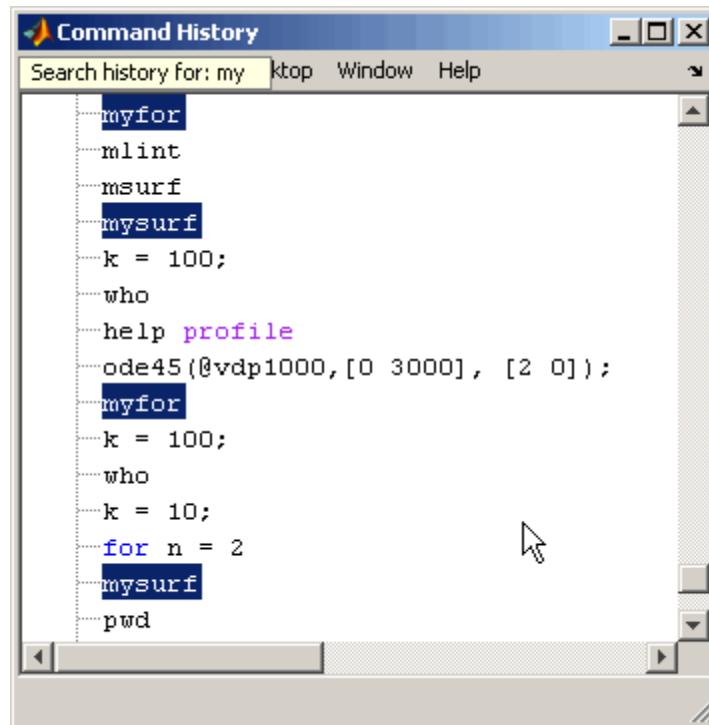


Finding Multiple Occurrences of the Entry. You can use the up and down arrow keys to search for the next or the previous occurrence of the entry you just found.

When you press **Ctrl** and the up or down arrow key, each occurrence of the entry remains highlighted while you search for additional instances.

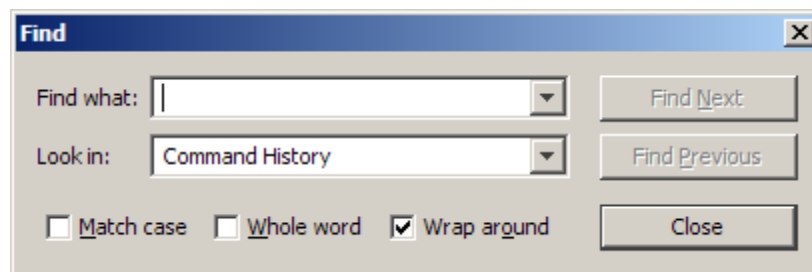


To highlight all instances of the entry, press **Ctrl+A**. In the example below, all instances of entries beginning with **my** are highlighted.



Finding Text

Select **Find** from the **Edit** menu to search for specified text using the Find dialog box. Complete the dialog box. The search begins at the current cursor position. MATLAB finds the text you specified and highlights it. Click **Find Next** or **Find Previous** to find another occurrence. Find looks for visible entries only, that is, it does not find entries in collapsed nodes.



MATLAB beeps when a search for **Find Next** reaches the end of the Command History window, or when a search for **Find Previous** reaches the top of the Command History window. If you select **Wrap around**, the search continues after the beep.

To search for the specified text in other MATLAB desktop tools, change the selection in the **Look in** field.

Printing the Command History Window

To print the contents of the Command History window, select **File > Print** or **Print Selection**. Specify options for printing by selecting **File > Page Setup**. For example, you can print the history with a header. For more information, see “Printing and Page Setup Options for Desktop Tools” on page 2-115.

The printed version is sized to fit the page. If there is a long statement in the Command History, the reduced page size might be difficult to read. As a workaround, either use **Print Selection**, where the long statement is not part of the selection, or remove any extremely long statements from the Command History before printing it.

Deleting Entries from the Command History Window

Delete entries from the Command History window when you think there are too many and it becomes inconvenient to find the ones you want. All entries remain until you delete them, or until the command history file exceeds its maximum size, at which point MATLAB automatically deletes the oldest entries. For more information, see “Viewing Statements in the Command History Window” on page 3-68.

After deleting entries from the Command History window, you cannot recall those statements in the Command Window (as described in “Recalling Previous Lines in the Command Window” on page 3-21).

To delete specific entries in the Command History window:

- 1 Select the entries to delete, using one of these methods:
 - Select a single entry.

- **Shift**+click or **Ctrl**+click to select multiple entries.
- Select the timestamp for a session to select all entries for that session. Then use **Shift**+click or **Ctrl**+click to select multiple timestamps with all their entries.

2 Right-click and select **Delete Selection** from the context menu, or press the **Delete** key.

A confirmation dialog box might appear; for more information, see “Confirmation Dialogs Preferences” on page 2-132.

To delete all entries in the Command History window, do one of the following:

- Select **Edit > Clear Command History**.
- Select **Clear Command History** from the context menu.

Preferences for Command History

In this section...
“Overview of Command History Preferences” on page 3-78
“Settings” on page 3-78
“Saving” on page 3-79
“See Also” on page 3-80

Overview of Command History Preferences

Using Command History preferences, you can choose to exclude statements from the command history file, `history.m`, and specify how often to save it. The command history file is used for both the Command History window and statement recall in the Command Window.

To set preferences for the command history file, select **File > Preferences > Command History**.

Tip For information on changing the date format in the Command History window, see “Customizing the Column Display” on page 7-19. Change the date format in the Command History using the method described for changing the date format in **Date Modified** column of the Current Folder browser.

Settings

Specify the types of statements to exclude from the command history file. Note that when you exclude statements from the command history file, you cannot recall them in the Command Window as described in “Recalling Previous Lines in the Command Window” on page 3-21, nor can you view them in the Command History window.

Save Exit/Quit Commands

Select the check box to save exit and quit commands in the command history file.

Save Consecutive Duplicate Commands

Select the check box if you want consecutive executions of the same statement to be saved to the command history file.

For example, with this option selected, run `magic(5)`, and then run `magic(5)` again. The command history file saves two consecutive entries for `magic(5)`. With this option cleared, for the same example, the command history file saves only one entry for `magic(5)`. If you then run `magic(10)`, the command history file saves both entries, `magic(5)` followed by `magic(10)`.

Saving

Use **Saving** preferences to specify how often to automatically save the command history file during a session of running the MATLAB software. By default, MATLAB saves the history after every statement. This allows you to more easily recover your state in the event of an abnormal termination, because you can reconstruct it using the history.

Save History File On Quit

Select this option to save the command history file when you end the session of MATLAB. If the session does not end via a normal termination, that is, via the `exit` or `quit` functions, **File > Exit MATLAB**, or the MATLAB desktop Close box, the history file is not saved for that session.

Save After n Commands

Select this option to save the command history file after `n` statements are added to the file. For example, when you select the option and set `n` to 10, after every 10 statements are added, the history file is automatically saved. Use this option instead of **Save History File on Quit** if you do not want to risk losing entries to the saved history because of an abnormal termination, such as a power failure.

Don't Save History File

Select this option if you do not want to save the command history file. This feature is useful when multiple users share the same machine and do not want other users to view the statements they have run.

Note that any entries already in the `history.m` file remain. Before setting this preference, you might want to remove any existing entries. Follow the instructions in “Deleting Entries from the Command History Window” on page 3-76.

See Also


- “Using the Command History Window” on page 3-66
- Additional preferences that relate to the Command History:
 - “Setting Fonts Preferences for Desktop Tools” on page 2-141
 - “Confirmation Dialogs Preferences” on page 2-132

Getting Help and Product Information

- “Overview of Help” on page 4-2
- “Using the Help Browser” on page 4-4
- “Searching the Documentation” on page 4-14
- “Learning from Demos” on page 4-25
- “Configuring the Help Browser” on page 4-28
- “Using Printed Documentation” on page 4-35
- “Additional Help and Learning Resources” on page 4-37

Overview of Help

You can get help using MATLAB and other MathWorks products in a number of ways, depending on your needs. For example:

- To get quick help for a function in the Command Window, use the `help` function or the Function Browser .
- To browse all HTML documentation and demos, use the Help browser **Contents** pane.
- To search all HTML documentation and demos, use the Help browser search field.
- To get context-sensitive help, use the **Help** button that certain graphical user interfaces (such as Preferences and Set Path) provide.

The Help browser is the main conduit for all product documentation. It provides access to the following types of documentation for every product you have installed:

General Release Notes	High level descriptions of updates to all MathWorks products
Getting Started guides	Overview introductions to using a product
User Guides	Comprehensive descriptions of the capabilities and ways of using a product, including many examples and links to related information
Reference pages	Complete descriptions of functions and blocks, including syntax, limitations, examples and links to related features
Examples you can run	Code you can select and execute that illustrate product capabilities and programming techniques
Demos	Standalone executable examples, often with explanatory text
Product Release Notes	Descriptions of what's new for each product in a release, with links to user guides and compatibility considerations

Printed documentation	Links to complete sets of user guides, release notes, and reference documentation in Adobe® Portable Document Format (PDF) files on the MathWorks Web site
Information on the MathWorks Web site	Links to supplemental demos, MATLAB Central, Technical Support, and information on platform requirements, books, and learning opportunities

You can access everything the Help browser provides except for executable examples on the MathWorks Web site.

For more information, see “Getting Help” in the MATLAB Getting Started guide.

In addition to getting help and using demos, you can add your own help and demos for MATLAB programs you create. For more information, see “Providing Your Own Help and Demos” on page 5-8.

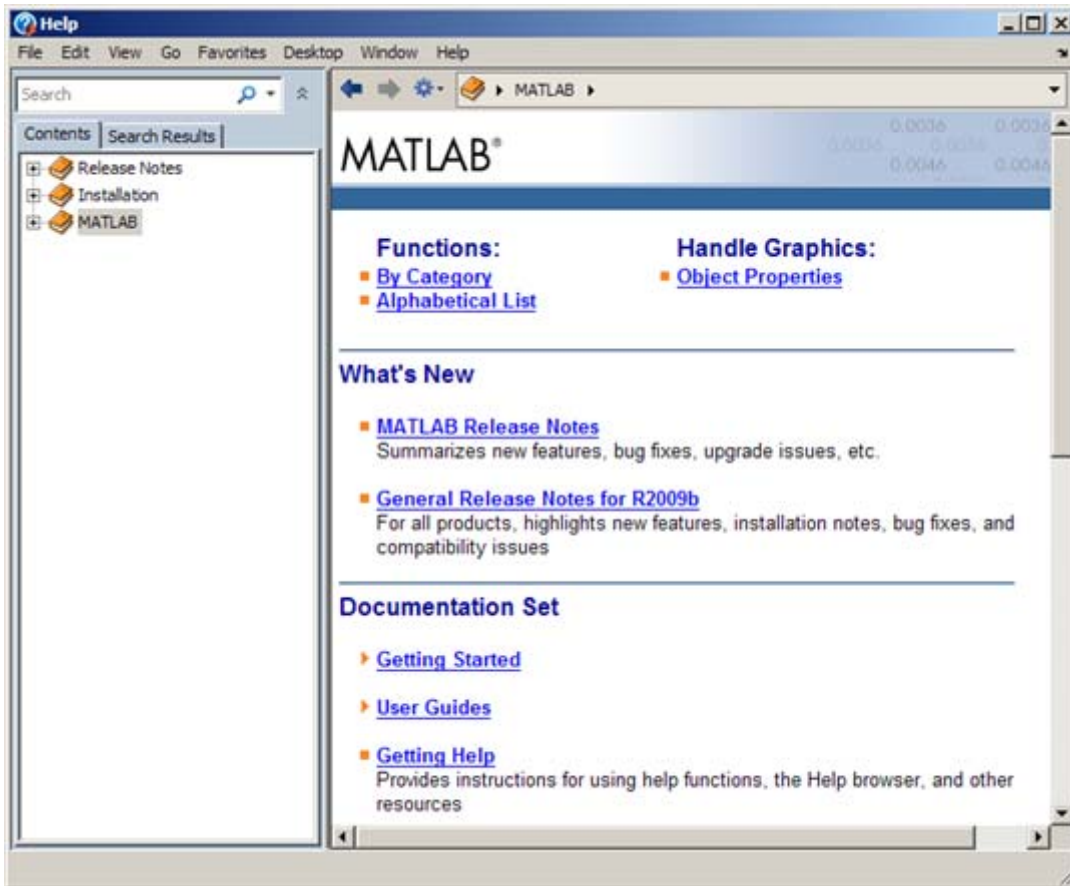
Using the Help Browser

In this section...
“About the Help Browser” on page 4-4
“Getting Help for Functions and Blocks” on page 4-8
“Accessing a Specific Page” on page 4-12
“See Also” on page 4-13

About the Help Browser





- “Opening the Help Browser” on page 4-6
- “Using the Help Navigator” on page 4-7

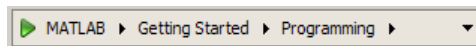
The Help browser contains two panes. On the right is the *viewing* pane, where you read documentation. The pane on the left is the *help navigator*, where you browse tables on contents and results from searching the documentation. The following illustration shows how the Help browser looks when you first open it (in the case that only MATLAB is installed).



The home page for each product is called a *roadmap*. Click on any product in the help navigator to display its roadmap. Click the underlined blue links on the roadmap to display documentation for that product in the viewing pane. Clicking a link with an orange triangle ▶ next to it expands the roadmap to reveal additional links.

The Help browser has buttons and other controls beneath its menu bar for you to:

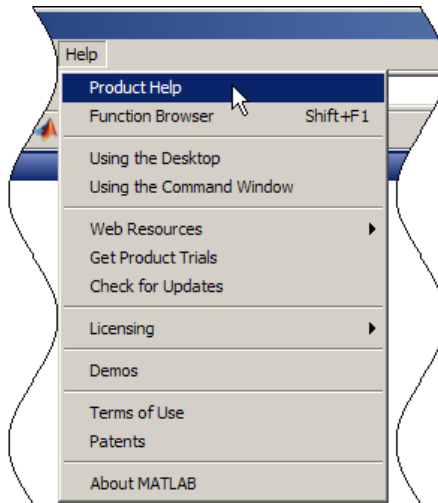
- Search all documentation for words, phrases, or Boolean expressions that you type in the search field .
- Navigate backward  and forward  through pages you have viewed
- Find text on a page, print a page, execute examples, and navigate using the **Actions**  drop-down menu
- Backtrack using the *navigation bar*



Opening the Help Browser

When you start MATLAB, the Help browser is closed. You can open the Help browser in different ways:

- Click **Help > Product Help** from the Desktop menu bar or from any desktop component

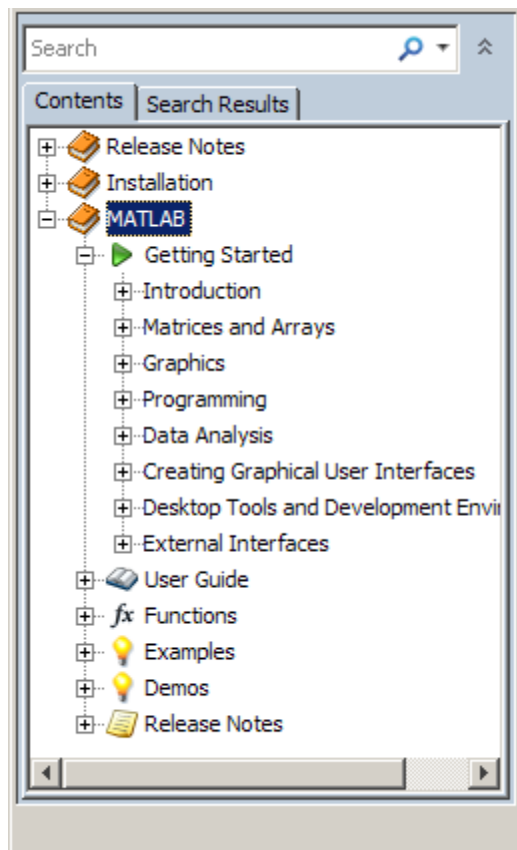


- Type `doc` or `doc function_name` in the Command Window
- Type `docsearch` followed by one or more search terms in the Command Window



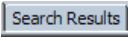
You can dock the Help browser in your desktop, like other desktop components. If you do not need the Help browser to remain open, you can close or minimize it at any time. If you close and reopen the Help browser, it always opens displaying the roadmap for MATLAB, not the last page you viewed.



Using the Help Navigator

The help navigator occupies the left pane of the Help browser. When you click a product name, icon, or the + (plus sign) beside it, the item expands into a table of contents. Each subsection expands to show lower-level topics, as the following illustration shows.



The help navigator has these main components:

- The contents pane, which displays a table of contents when you click the **Contents** tab 
- The search field , where you type text you want to find in the documentation
- The **Search Results** tab , which displays the results of your most recent search in the contents pane.

To enlarge the area of the viewing pane, you can hide the help navigator by clicking the up arrows  to the right of the search field. The up arrows then become down arrows  that you click to display the help navigator again.

Getting Help for Functions and Blocks

- “Help for Functions and Blocks” on page 4-8
- “Help for Overloaded Functions” on page 4-10

Help for Functions and Blocks

As you work with MathWorks products, you might need help for a function or block. You can use different ways to get help, depending on the tool you are using when you need the help and your preference.

When You Are Using...	How to Get Help	For More Information
Any desktop tool	Select Help > Function Browser .	“Finding Functions Using the Function Browser” on page 3-40
Help browser Contents pane	<ol style="list-style-type: none"> Expand the listing for a product. Expand the Functions or Blocks entry to view all functions or blocks in the product. 	“Browsing for Documentation and Demos”

When You Are Using...	How to Get Help	For More Information
	<p>3 Select a function or block to view its reference page.</p>	
Help browser search	<p>1 Search for a function or block name.</p> <p>2 Sort results by type, and then view results for the Reference type.</p> <p>3 Select a function or block to view its reference page.</p>	“Searching for Documentation and Demos”
Page displayed in the Help browser	<p>When a function name <i>is</i> a link, click the link to view the reference page for the function.</p> <p>When a function name is <i>not</i> a link:</p> <p>1 Select a function name on the page.</p> <p>2 Right-click, and do one of the following:</p> <ul style="list-style-type: none"> • Display the reference page for the selected function in the Help browser, by selecting Help on Selection. 	None

When You Are Using...	How to Get Help	For More Information
	<ul style="list-style-type: none"> • Open the function file in the Editor, by selecting Open Selection. 	
Current Folder browser	<ul style="list-style-type: none"> • View a brief description for a file in the details pane. • View the reference page in the Help browser by right-clicking the file and select View Help. 	<ul style="list-style-type: none"> • “Viewing File Descriptions” on page 7-20 • “Viewing Help for a MATLAB Program File” on page 7-24
Command Window or Editor	Right-click on the function name and select Help on Selection .	“Getting Help for a Function Shown in the Command Window or Editor” on page 3-38
Command Window	<ul style="list-style-type: none"> • To display a reference page, run <code>doc name</code>. • To display brief help in the Command Window, run <code>help name</code>. 	<ul style="list-style-type: none"> • <code>doc</code> • <code>help</code>

Help for Overloaded Functions

When there is more than one function with the same name (called an *overloaded* function), typing `help` in the MATLAB Command Window for that function lists all the overloaded functions at the end of the help text. For example, the help text for the `loglog` plotting function is:

```
help loglog
```

```
LOGLOG Log-log scale plot.
```

```
LOGLOG(...) is the same as PLOT(...), except logarithmic
scales are used for both the X- and Y- axes.
```

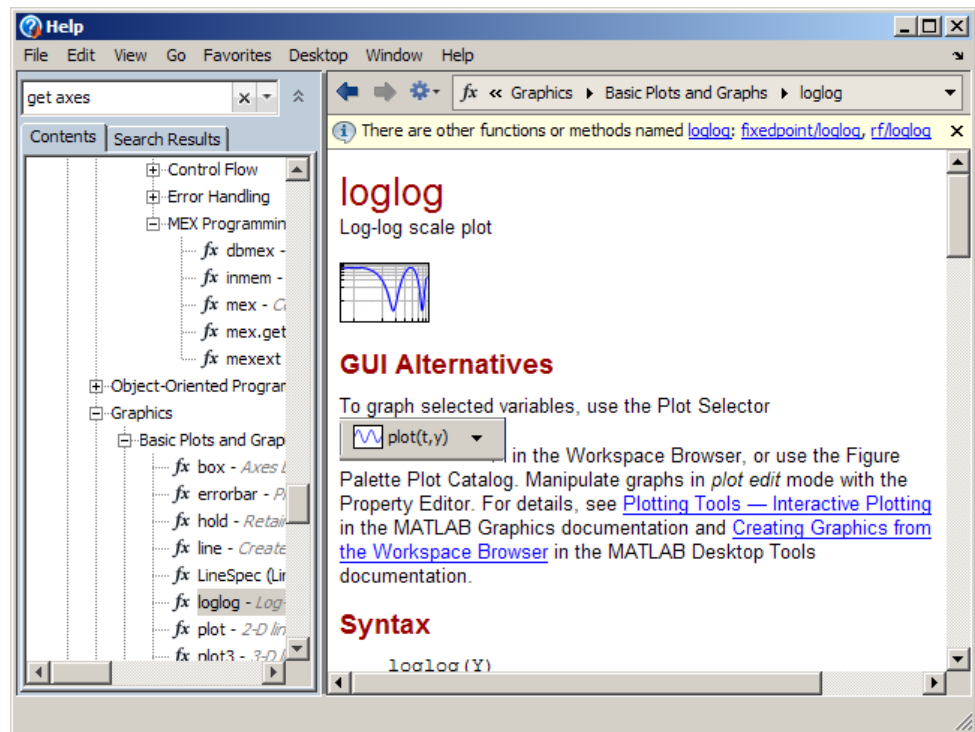
```
See also plot, semilogx, semilogy.
```

Overloaded methods:
 distributed/loglog
 rfckt.loglog
 frd/loglog

Reference page in Help browser
 doc loglog

All of the function names are links. The `See also` and `Overloaded methods` items display Command Window help when you click them. The `Reference page in Help browser` link displays documentation in the Help browser.

When you click the `doc loglog` link, the reference page opens in the Help browser and the overloaded methods display as links in an information bar at the top of the page, as shown here.



Another way to view a reference page is to select the name of a function in the Command Window or in the Editor, right-click, and select **Help on Selection**. Or, you can select the name and press **F1**, (This is the default hot key for Help on Selection. You can change the hot key by modifying your Preference for that keyboard shortcut.)

Accessing a Specific Page

- “Bookmarking Favorite Pages” on page 4-12
- “Getting the Link to a Page” on page 4-12

Bookmarking Favorite Pages

You can bookmark your favorite pages in the documentation and your favorite demos. In MATLAB, bookmarks are called *favorites*.

Use the **Favorites** menu to add, go to, and organize favorites

MATLAB saves favorite shortcuts, with special values for the callback and category that makes the shortcut go to a page in the Help browser. Therefore, when you save a favorite, do *not* change the **Callback** or **Category**.

Note You cannot migrate favorites saved in one MATLAB release to a new release.

Getting the Link to a Page

To tell someone else about a specific page in the documentation, you can send them a link to the page.

To get the link for a page (URL) shown in the Help browser:

- 1 With the page displayed in the Help browser, select **View > Page Location**. The Help Page Location dialog box opens.

2 Copy the link from one of the fields:

- If the link is for someone with the same release of MATLAB, use the link to the Help browser.
- If the link is for someone with a different release of MATLAB, or who does not have the product, use the link to the Web site. To view the page on the Web site, click **Go**. For more information, see “Product Documentation at the MathWorks Web Site” on page 4-39.

See Also

- “Getting Help for Files Created by Others” on page 5-2
- “Providing Your Own Help and Demos” on page 5-8
- “Product Documentation at the MathWorks Web Site” on page 4-39
- `doc` function to go directly to a reference page or roadmap page
- `demo` function to go directly to a demo page
- `docsearch` function to go directly the first search result for the search term you specify

Searching the Documentation

In this section...
“Performing a Simple Search” on page 4-14
“Improving Search Results” on page 4-14
“Advanced Search Techniques” on page 4-20
“Searching Within a Page” on page 4-23

Performing a Simple Search

To search documentation and demos for keywords:

- 1** Enter one or more keywords in the search field in the Help browser. All words must exist on a page for it to qualify as a search result. To search for an exact phrase, enclose it in double quotation marks, for example, “help report”.
- 2** Press **Enter**.

A listing of pages containing your search terms appears in the Search Results pane of the Help Navigator. When you select one of the results by clicking it, your search terms are highlighted in color in the viewing pane, each word or phrase in a distinct color.

Note The search engine ignores capitalization and punctuation.

For additional information about performing a basic search, see “Searching for Documentation and Demos” in the MATLAB Getting Started Guide.

Improving Search Results

- “Too Many Search Results?” on page 4-15
- “Too Few Search Results?” on page 4-15
- “Using Search Hints” on page 4-15

- “Using Search History” on page 4-17
- “Additional Guidelines for Searching” on page 4-20

Too Many Search Results?

To reduce the number of search results in the Help browser, try:

- “Limiting Search to Certain Products” on page 4-22
- Adding words in the search field
- “Searching for an Exact Phrase” on page 4-20
- “Excluding Results That Contain Specified Words — Boolean NOT” on page 4-22

Too Few Search Results?

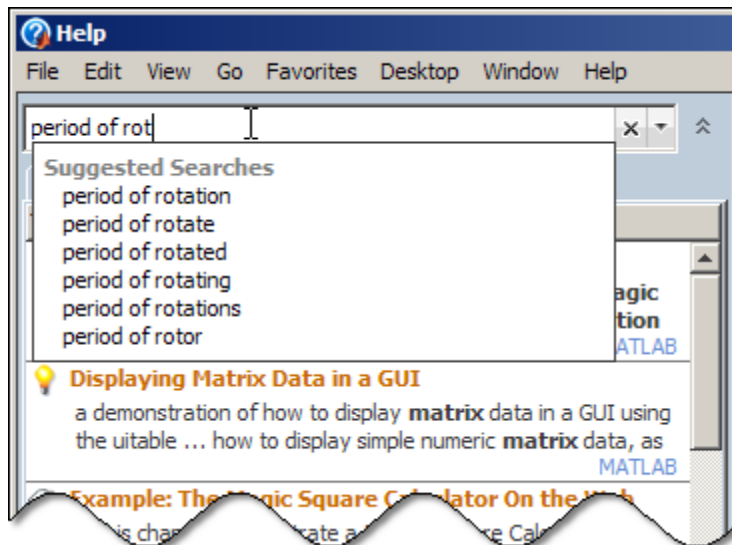
To increase the number of search results in the Help browser, try:

- Searching in more products. See “Limiting Search to Certain Products” on page 4-22.
- “Searching for Part of a Word — Using Wildcards” on page 4-21.
- Removing search words. All search terms must exist on a page for that page to appear as a result. Terms that you deliberately exclude using NOT are excluded.
- Not searching for exact phrases. Slightly different expressions might exist than those you seek.
- “Finding Any of the Words — Boolean OR” on page 4-22.
- Looking in bug reports, solutions, and technical notes at the MathWorks Web site. Click **Search Online Support** at the bottom of **Search Results** pane.

See also “Additional Help and Learning Resources” on page 4-37.

Using Search Hints

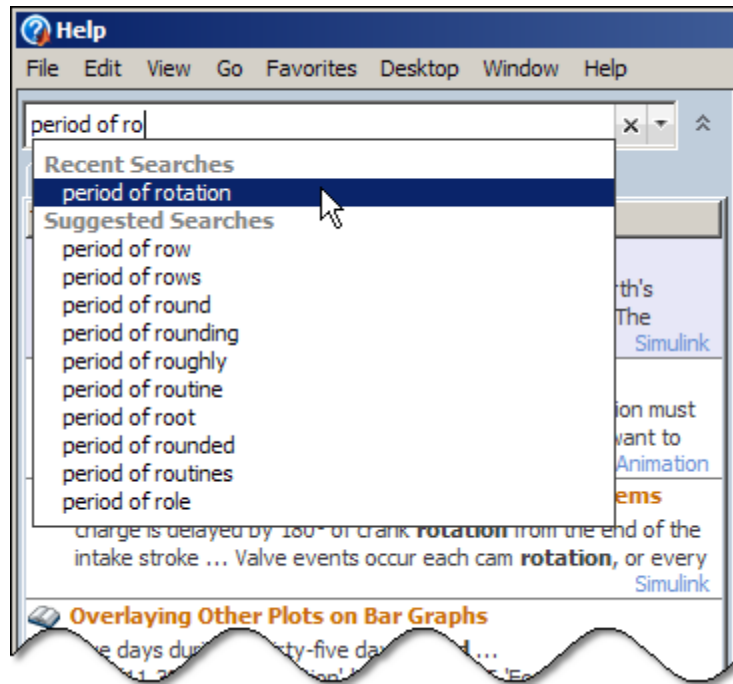
When you type in the search field, a menu of search hints labeled **Suggested Searches** appears beneath the field, as shown in the following illustration.



The Help browser bases hints on what you have typed so far and orders them by relevance. The list changes as you continue to type. If you see a suggested search phrase that you want to use, do either of the following:

- Select a suggestion in the drop-down menu with the mouse pointer and click to choose the selected item.
- Press **Down Arrow** key to navigate to the item (or **Up Arrow** to move back up the list). Press **Enter** to choose the selected item.

After you make one or more searches, the drop-down menu includes a section at its top labeled **Recent Searches**, as shown in the following illustration.



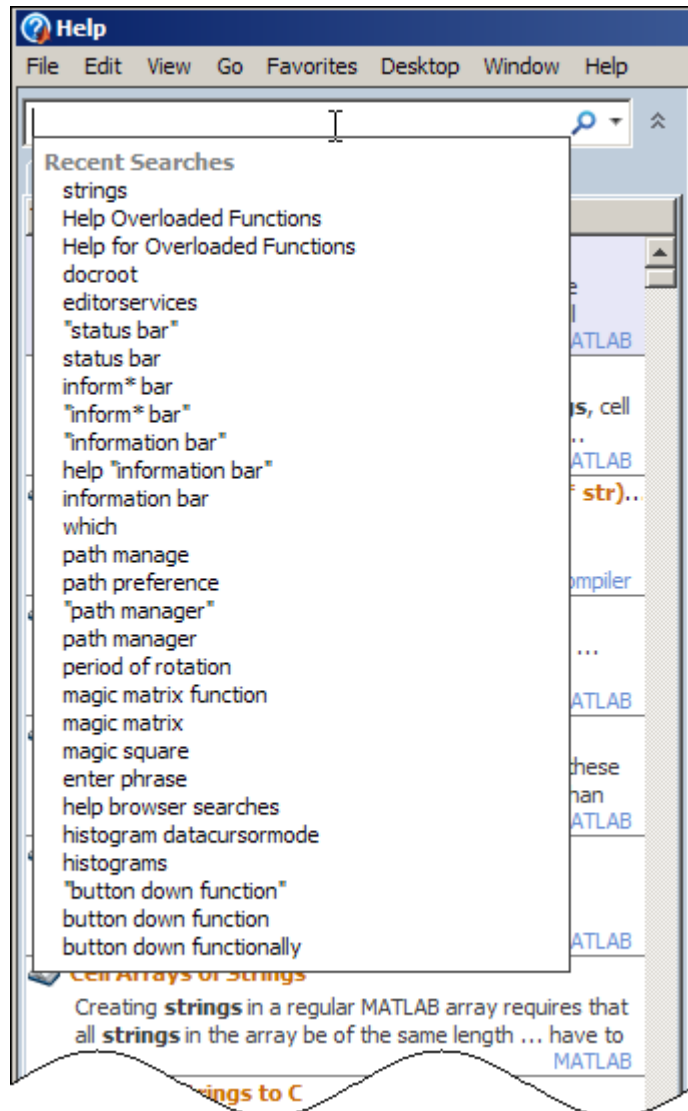
Choose one of the Recent Searches items to repeat a search.

Using Search History

When you type words and phrases in the search field, MATLAB remembers them by maintaining a *search history*, like it does for commands you type in the Command Window. Your search history persists between MATLAB sessions, as well. To use a search term you have used before without retyping it, click the down arrow ▼ to the right of the search field and select **Show Search History**, as the following illustration shows.



Click the search field and press the down-arrow key to reveal your search history. The history drops down below the search field. Select an item from it or type a new search term in the search field and press **Enter**.



Narrow the list of search terms by starting to type a word. The list shortens to include only those terms that begin with the letters you type.

Additional Guidelines for Searching

If the Help browser does not find what you want, try changing your search criteria based on how search works:

- Search ignores common, insignificant words, such as **a**, **an**, **the**, and **of**, unless they are part of an exact phrase. Note that search interprets each asterisk (*) in a search term as a wildcard character.
- Search is case insensitive.
- Search finds letters and digits, but not symbols such as punctuation marks, plus, minus, and so forth. See “Searching for Special Characters and Symbols” on page 4-23.
- Search looks in the following places:
 - Documentation — Text and code shown in the Help browser
 - Product demos — Comments and code in program files and models
 - GUI-based demos — Help comments in the program file
 - Video demos — The title

Advanced Search Techniques

- “Searching for an Exact Phrase” on page 4-20
- “Searching for Part of a Word — Using Wildcards” on page 4-21
- “Limiting Search to Certain Products” on page 4-22
- “Finding Any of the Words — Boolean OR” on page 4-22
- “Excluding Results That Contain Specified Words — Boolean NOT” on page 4-22
- “Using Multiple Boolean Operators” on page 4-23
- “Searching for Special Characters and Symbols” on page 4-23

Searching for an Exact Phrase

To reduce the number of irrelevant results the Help browser finds, specify an exact phrase by enclosing the words in quotation marks. Search accepts more than one exact phrase.

For example:

- Find sections that contain *plot tools*, in that sequence, with no words between them:

```
"plot tools"
```

- Find sections that contain both *"plot tools"* and *"figure palette"*:

```
"plot tools" "figure palette"
```

Searching for Part of a Word – Using Wildcards

The Help browser searches for each entire search word you specify. For example, if you search for *plo*, you are unlikely to find pages containing *plot*. To search for part of a word in the Help browser, use the wildcard character (*) in place of characters in a search word. Search accepts more than one wildcard character.

For example:

- Find *plot*, *plot3*, *plotted*, *plots*, or *plotting*:

```
plot*
```

- Find *plot tools* or *plotting tool*:

```
plot* tool*
```

When you use wildcards, the Help browser normally returns more results than it does without wildcards. Add search terms or preface a term with the Boolean operator NOT to limit results.

Restrictions When Using Wildcards.

- You must use two or more characters with a wildcard. For example, *p** fails.
- You cannot use wildcards within an exact phrase. For example, *"plot* tool"* fails.
- Do not begin a search word with a wildcard character. For example, **tool* fails.

Limiting Search to Certain Products

By default, the Help browser searches in the documentation and demos for all installed products.

To specify which products to search, use the product filter. See “Specifying Which Documentation to Display” on page 4-28.

After performing a search, to organize results by product, click the **Product** column header.

To search in the documentation for products that are *not* installed, search the documentation on the MathWorks Web site. For more information, see “Product Documentation at the MathWorks Web Site” on page 4-39.

Finding Any of the Words – Boolean OR

For more search results in the Help browser, look for sections that contain any of the search words by including OR between words:

- Include a space before and after OR
- Use all capital letters for OR.

For example, find sections that contain plot or graph:

```
plot OR graph
```

Excluding Results That Contain Specified Words – Boolean NOT

To find sections that contain specified search words, but do not contain other specified words, include NOT before the words to exclude. Using NOT reduces irrelevant search results in the Help browser.

For example, find sections that contain "plot tools", but do not contain "time series":

```
"plot tools" NOT "time series"
```

Using Multiple Boolean Operators

The Help browser search evaluates NOT operators first, OR operators second, and AND operators last.

By default, search looks for pages that contain all the search words, which is called a Boolean AND. If there is no Boolean operator before a word, search assumes that there is an AND before it.

When you construct a search with multiple operators, including AND before a word helps you understand the search.

For example, find pages that contain either `plotting tool` or `plot tools` and contain `workspace`, but do not contain `time series`:

```
"plotting tool" OR "plot tools" NOT "time series" AND workspace
```

Searching for Special Characters and Symbols

To find a symbol or special character using the Help browser, look for the word instead of the symbol or character.

For example, look for `plus` instead of `+`.

Other ways to find information about special characters and symbols are:

- Checking the Operators and Special Characters category in the MATLAB Function Reference
- Searching in the PDF documentation, which supports searching for special characters and symbols. For details, see “Accessing and Printing PDF Documentation” on page 4-35.

Searching Within a Page

Using Highlighted Search Words

When viewing a page from the Help browser search results, scroll through the page to view the search words, which appear with highlights.

The highlighting clears when you go to another section by clicking a link or using the navigation bar.

To clear the highlights while viewing the page, right-click on the page and select **Refresh**

To restore the highlights after clearing them, go to a different page. Then, select the search result again.

Using the Find Tool

To locate search words or any word on a page in the Help browser, select **Edit > Find** and use the **Find** dialog box. To go to the next instance on the page, use the keyboard shortcut, **F3**. Use **Shift+F3** to go to the previous instance. The find tool looks for partial words. For example, `plot` finds `plot` and `plotting`. You can change these and other keyboard shortcuts by selecting **File > Preferences** and customizing key assignments in the Keyboard Shortcuts pane.

Learning from Demos

In this section...

“About Demos” on page 4-25

“Types of Demos” on page 4-25

“Accessing Demos” on page 4-26

“Running Demos” on page 4-26

About Demos



MATLAB provides demonstrations of product features so you can see how the features work. Some of the demos include executable code.

Note Related to demos are examples in the documentation. Examples generally contain code that you can execute and adapt.



To view the demos that are available, as well as documentation examples, use the Help browser. For more information, see “Browsing for Documentation and Demos” in the MATLAB Getting Started Guide.

You can run the code in the demos and executable documentation examples. You can also use the code in your work. For more information, see “Running Demos and Code in Examples” in the MATLAB Getting Started Guide.

Types of Demos

Icon	Type	Description	Example
	Script	Script file demos:	In the Help browser Contents pane, select
	GUI	Standalone tool for exploring a feature.	In the Help browser Contents pane, select MATLAB > Demos > Graphics > Vibrating Logo .

- Are created by publishing a code script to HTML output using the Editor.
- Can run without stopping, or cell-by-cell. A cell is a section of code that begins with two comment symbols (%%).

Icon	Type	Description	Example
	Model	Simulink block diagram.	In the Help browser Contents pane, select Simulink > Demos > Automotive Applications > Engine Timing Simulation .
	Video	Video demos: <ul style="list-style-type: none"> • Are movies that highlight key features. • Play in your system Web browser using the Adobe Flash Player plug-in. • Could require an Internet connection. 	In the Help browser Contents pane, select MATLAB > Demos > Getting Started > Importing Data from Files .

Accessing Demos

Demos are available in the Help browser:

- Browse for demos in a product using the Help browser **Contents** pane. For more information, see “Browsing for Documentation and Demos”.
- Search for a topic of interest in the Help browser. Sort results by type, and then select a result from the **Demos** type. For more information, see “Searching for Documentation and Demos”.
- Use the demo function.

Running Demos

To run a MATLAB code demo from start to finish, see “Running a Script Demo”.

To run a MATLAB code demo section-by-section:

- 1 Display the demo in the Help browser.
- 2 On the demo page, click **Open filename in the Editor**.
- 3 In the Editor, select **Cell > Evaluate Current Cell and Advance**.

To run a MATLAB code demo in the Command Window:

- 1** Click **Run in the Command Window**.
- 2** Scroll up in the Command Window to see the start of the instructions.
- 3** Follow the instructions.

Configuring the Help Browser

In this section...

“Adjusting the Help Browser Layout” on page 4-28

“Specifying Which Documentation to Display” on page 4-28


“Accessing English Documentation on Japanese Systems” on page 4-29


“Customizing Help Browser Fonts and Colors” on page 4-30

“Preferences for Configuring Help Windows, Search History, and PDF Readers” on page 4-32

Adjusting the Help Browser Layout

By default the Help browser opens with the Help Navigator pane (the **Contents** and **Search Results**) on the left and the viewing pane on the right. If you make the Help browser window narrow, the Help Navigator pane moves above the viewing pane. This layout provides more space for displaying pages. Use this configuration when you want to reference the Help browser while working with a tool, for example, the Editor or figure windows. The Help browser uses a top-bottom layout when you dock it next to another tool in the desktop.

To provide more space for viewing a page in the Help browser, click the up arrows , located to the right of the search field. The Help Navigator pane closes.

To reopen the pane, click the down arrows  that now display in the same location.

Specifying Which Documentation to Display

By default, the Help browser and Function Browser show the documentation and demos for all installed MathWorks products. However, you can hide the documentation for any product if you have products you tend not to use or you want to locate relevant information more quickly. To specify the products that the Help browser and Function Browser display:

- 1 Select **File > Preferences > Help**.

- 2 Under **Filter by Product**, select the **Selected products** radio button.
- 3 From the list of installed products, select the products you want the Help browser to include.
- 4 Click **OK**.

Note The Release Notes Help Preferences entry refers to the general Release Notes overview document for all products in a release. It does not apply to the product-specific release notes, which are part of the documentation for a product.

Accessing English Documentation on Japanese Systems

Many MathWorks products provide documentation translated into Japanese. The translated documentation is usually one release behind the product. However, the Help browser can also access the English documentation for the release you are using. To read documentation for the current version in English, use the **Language** panel in the Help Preferences dialog to switch from Japanese to English. You can toggle the preference setting, enabling you to revert to Japanese at any time. The option changes the language used in the Help browser and for GUI context-sensitive help, but it does *not* affect the language appearing in menus or elsewhere in products. For example, the `help` command still provides help in Japanese, even after you switch the Help browser to English. For more information, see “Obtaining Documentation in Different Languages” on page 4-39.

Note The **Language** preference is available only when the system locale is Japanese and the Japanese documentation is installed. If the documentation for a product is not translated, the Help browser displays the English documentation for it no matter how you set the preference.

Customizing Help Browser Fonts and Colors

You set font characteristics and colors for the Help browser as you do for other desktop tools, with some exceptions. Some settings apply to text, others to code, and font styles apply only in the Help Navigator.

Specifying the Font Name, Style, and Size

You specify fonts for the Help browser **Contents** and **Search Results** panes separately from the fonts for the display pane.

By default, the pane for **Contents** and **Search Results** uses the desktop text font. You can change the font family, style (bold or italic), and size.

The font for Help browser display pane is specified by the **HTML Proportional Text** setting in the MATLAB Fonts Custom Preferences dialog box. By default, HTML Proportional Text tools use a custom font (Sans Serif, 10 pt.). You can change these settings, but not all changes you make affect the display pane in the Help browser:

- Changing the font size applies to all text and code in the display pane.
- Changing the font family applies to *text*, but not *code* in the display pane.
- Changing the font style to bold or italic does not apply to the display pane.

For related information, see “Setting Fonts Preferences for Desktop Tools” on page 2-141.

Example — Specifying Fonts for the Help Browser. Specify Microsoft Comic Sans® MS, italic, 14 pt. font for the Help browser display pane:

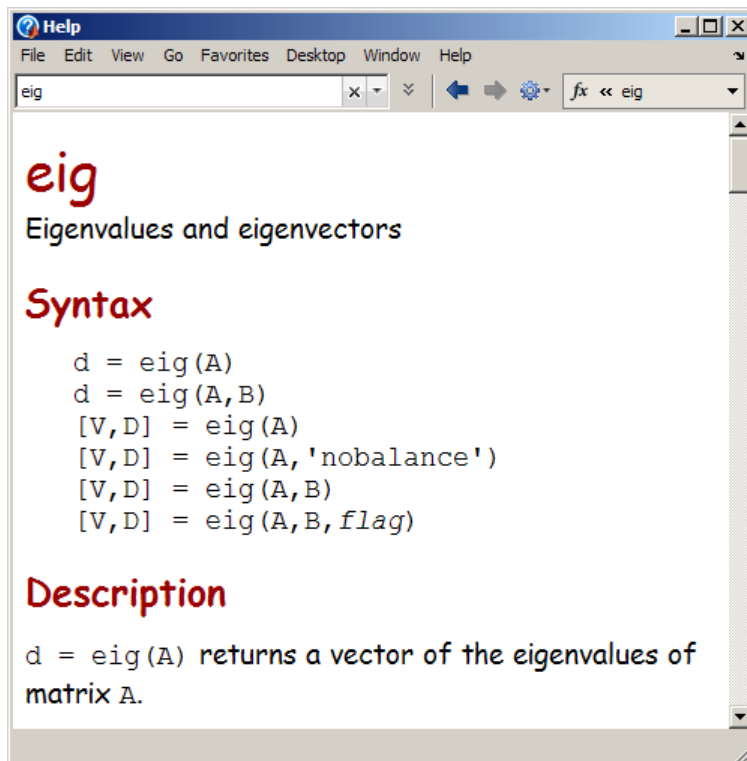
- 1** Select **File > Preferences > Fonts > Custom**.
- 2** From the **Desktop tools** list, select **HTML Proportional Text**.
- 3** For **Font to Use**, select **Custom**, and specify the characteristics:
 - Family — Comic Sans MS
 - Size in points — 14
 - Style — bold

4 Click **Apply** or **OK**.

The font for the page display pane uses the new settings. The font in other HTML Proportional Text tools, like the MATLAB Web browser also uses the new settings.

The change to the bold style has no effect.

Code on the page does not use the HTML Proportional Text font. The change to Comic Sans MS does not affect it.



Specifying Text and Background Colors

Specify the background and text color used in the pane for **Contents** and **Search Results** the same way you do for other desktop tools. See “Setting Colors Preferences” on page 2-150.

Note You cannot specify text, background, or hyperlink colors for the Help browser display pane.

Preferences for Configuring Help Windows, Search History, and PDF Readers

- “Specifying Where Help from the Editor and Function Browser Displays ” on page 4-32
- “Specifying a Search History Limit” on page 4-33
- “Specifying the PDF Reader Location — UNIX Platforms Only” on page 4-33

Specifying Where Help from the Editor and Function Browser Displays

By default, MATLAB displays the reference page in a small window when you:

- Get help on a selection from the Command Window or Editor.
- Click **More Help** from the Function Browser or function hints pop-up windows.

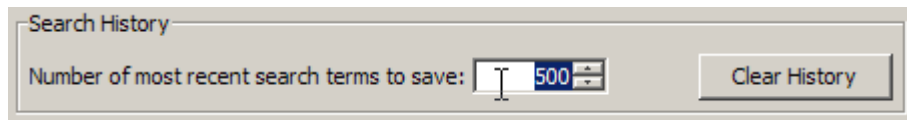
To display the reference page in the Help browser:

- 1** Select **File > Preferences > Help**.
- 2** For **Help on Selection and More Help**, select **In Help browser**.

This preference does not apply when you get help for a selected function from the Current Folder browser or Help browser. Then, the reference page opens in the Help browser.

Specifying a Search History Limit

Your search history is a list of terms that you have used when searching in the Help browser. MATLAB maintains this list for your current and previous sessions. By default, your 500 most recently used terms are available to you via a drop-down list under the Help Navigator search field. To change the limit for the number of items in your search history, go to **File > Preferences > Help** and, in the **Number of most recent search terms to save** field, enter or select the number you want. You can erase your entire search history by clicking the **Clear History** button. The following illustration shows these controls.



For more information about search history, see “Using Search History” on page 4-17.

Specifying the PDF Reader Location – UNIX Platforms Only

To display the PDF version of the documentation, the Help browser needs to locate the PDF reader on your system. For example, the Adobe® Acrobat® product is a PDF reader that many people use on a variety of platforms.

On Microsoft Windows and Apple Macintosh platforms, MATLAB obtains the PDF reader location from the operating system.

On UNIX platforms, the default PDF reader is Acrobat® and MATLAB automatically determines its location, if it exists. To use a different PDF reader:

- 1 Select **File > Preferences > Help**.
- 2 For **PDF reader**, enter the full path to the application.
- 3 Click **OK**.

Note The **PDF reader** preference is only for UNIX platforms. It does not appear in the **Help Preferences** pane for other platforms.

For related information, see “Accessing and Printing PDF Documentation” on page 4-35.

Using Printed Documentation

In this section...

“Printing from the Help Browser” on page 4-35

“Accessing and Printing PDF Documentation” on page 4-35

“Obtaining Printed Manuals” on page 4-36

Printing from the Help Browser

- 1 In the Help browser, display the page you want to print.
- 2 Select **File > Print**.
- 3 A system print dialog opens for you to select a printer and set properties for the print job.

Accessing and Printing PDF Documentation

Most of the documentation you access from the Help browser is available in PDF format. Use PDF documentation to print:

- Tables of contents
- Multiple pages of documentation
- Pages using a book style format rather than a Web style format

Accessing the PDF documentation from the Help browser requires:

- A PDF reader, for example, Adobe Acrobat or Apple Preview.
- An Internet connection, because the PDF documentation is only stored at the MathWorks Web site.

To access and print the PDF documentation:

- 1 In the Help browser **Contents** pane, select a product. The product roadmap page opens.

- 2 For **Printable (PDF) Documentation on the Web**, select the link for the document you want to print.

The Help browser accesses the PDF document from the MathWorks Web site and opens it in your PDF reader. It can take some time for the PDF to open and load completely if the bandwidth of your Internet connection is limited.

On UNIX⁹ platforms, if the PDF documentation fails to open, check the **Help Preferences**. See “Specifying the PDF Reader Location — UNIX Platforms Only” on page 4-33.

- 3 Locate the pages to print. You can use the Table of Contents and the Index, both of which provide links.
- 4 Print the documentation from the PDF reader.

You can save the PDF to local storage if you intend to refer to it again.

Obtaining Printed Manuals

Some MathWorks products provide printed “Getting Started” manuals.

Note Printed manuals contain a small subset of the online documentation and might not include the most current information.

To find out what printed manuals are currently available, contact MathWorks Sales.

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

Additional Help and Learning Resources

In this section...
“Obtaining Information About your Installation” on page 4-37
“Obtaining Technical Support” on page 4-38
“Product Documentation at the MathWorks Web Site” on page 4-39
“Newsgroup for MathWorks Products” on page 4-40
“File Exchange — Files Created By Other Users” on page 4-40
“Blogs for MathWorks Products” on page 4-40
“Newsletters for MathWorks Products” on page 4-40
“Seminars and Webinars for MathWorks Products” on page 4-40
“Training for MathWorks Products” on page 4-41

Obtaining Information About your Installation

MATLAB software can tell you what products are installed, their versions, and other information about your license and platform. This information is important to have in the event you contact technical support.

Type of Information You Want	To Get the Information
Version and license for Installed product	<p>From the product, select Help > About.</p> <p>Or use functions:</p> <ul style="list-style-type: none"> • <code>license</code> — for the license number • <code>ver</code> — for version numbers for MATLAB and libraries • <code>version</code> — for version numbers for MathWorks products

Type of Information You Want	To Get the Information
Processor speed for the MATLAB version currently running	In MATLAB, select Help > About MATLAB . The About MATLAB dialog box shows 32-bit or 64-bit.
arch value used for the mex function	In MATLAB, select Help > About MATLAB . The About MATLAB dialog box shows the arch value, for example win32. Or use the <code>computer</code> function.
Passcodes and licenses	From any desktop tool, select Help > Web Resources > MathWorks Account .

For related information concerning system requirements, and procedures for installing , activating, and uninstalling products, see the installation guide for your computer platform.

Obtaining Technical Support

Technical support provides bug reports and workarounds, solutions to questions, published books and more.

Ways to access technical support:

- From MATLAB, select **Help > Web Resources > Support**.
- From a Web browser, go to <http://www.mathworks.com/support>.
- Search in the support solutions, technical notes, and bug reports from MATLAB:
 - 1** Perform a search in the Help browser.
 - 2** At the bottom of the **Search Results** pane, click **Search Online Support for**.

The results display in your system Web browser.

Product Documentation at the MathWorks Web Site

The MathWorks Web site provides documentation for the latest version of all MathWorks products. To view the Web site documentation, you need an Internet connection and a Web browser. You can view documentation on the Web site even if MATLAB is *not* installed on your system.

To access documentation for MATLAB or other MathWorks products:

- From MATLAB, select **Help > Web Resources > Support**.
- In a Web browser, go to <http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml>. Select the product containing the documentation you want to read, or type search terms into the search box on that page to locate specific content.
- From a page you are viewing in the Help browser, you can usually go to the same page in the documentation on the Web site. With the page displayed in the Help browser, select **View > Page Location**. The Help Page Location dialog box opens. Click **Go** to open the page in your system browser. See “Getting the Link to a Page” on page 4-12.

A good way to get an overview of what’s new or different in a release is to read the general Release Notes, which, for the version of MATLAB you are using, is also the first item in the Contents pane of the Help Navigator.

The documentation on the Web site can inform you about:

- MathWorks products that you do not have installed
- The latest release, if you are running an older version of MATLAB or another product

For more information, See “Product Documentation at the MathWorks Web Site” on page 4-39.

Obtaining Documentation in Different Languages

MathWorks usually provides Japanese translations of product documentation about 2 months after new versions of products first ship. When you install the new version of most products, the *previous version* of translated documentation is installed on your system along with the

current version of the English documentation. To read the most current documentation for your release, you can switch the Help browser to English. After the translations of the latest documentation are available, you can download and install them in your products. At that time, you can find a link to download the latest translated documentation at <http://www.mathworks.co.jp/access/helpdesk/help/helpdesk.html>. For information about documentation in languages other than English, contact your MathWorks sales and service office.

Newsgroup for MathWorks Products

Access a user forum, the Usenet newsgroup `comp.soft-sys.matlab` (also known as `cssm`), to find, ask for, or provide help.

Select **Help > Web Resources > MATLAB Newsgroup Access**, or go to <http://www.mathworks.com/matlabcentral>.

File Exchange — Files Created By Other Users

Before you code your own programs, see if someone has already programmed a similar application. Use program files created by other MATLAB users to save you time and provide you with new ideas for your own work. See Chapter 8, “File Exchange — Finding and Getting Files Created by Other Users”.

Blogs for MathWorks Products

Read weekly commentary from the people who design and build MATLAB and other MathWorks products.

Newsletters for MathWorks Products

Read News and Notes and the MATLAB Digest online. Select **Help > Web Resources > MATLAB Newsletters**, or go to <http://www.mathworks.com/company/newsletters/>.

Seminars and Webinars for MathWorks Products

See <http://www.mathworks.com/company/events/>.

Training for MathWorks Products

For details, select **Help > Web Resources > Training**, or go to <http://www.mathworks.com/services/training>.

Customizing Help and Demos

- “Getting Help for Files Created by Others” on page 5-2
- “Providing Your Own Help and Demos” on page 5-8
- “Adding Demos to the Help Browser” on page 5-52
- “Addressing Validation Errors for info.xml Files” on page 5-61

Getting Help for Files Created by Others

In this section...
“About Help for Files Created by Others” on page 5-2
“Getting Command-Line Help for Externally Supplied Program Files” on page 5-2
“Viewing a Help Summary for Externally Supplied Files” on page 5-3
“Accessing Help for Externally Supplied Class Files” on page 5-3
“Accessing Externally Supplied Documentation in the Help Browser” on page 5-6
“Accessing Externally Supplied Demos in the Help Browser” on page 5-7

About Help for Files Created by Others

When you use functions and applications not provided by MathWorks, you can also access help and demos for them provided by their authors. File creators can provide help in several forms. If you can, ask the creators what help they provided, or try to find the information as described in the following topics.

Getting Command-Line Help for Externally Supplied Program Files

A *function* or a *script* is a MATLAB program file with a `.m` extension. To get help for an externally supplied function or script:

- 1 Verify that the file is in the current folder or a folder that is on the search path.

- 2 In the Command Window, type `help filename`.

If the creator of the file included help comments at the top of the file, then help for the file appears in the Command Window.

To provide help for your own files, see “Adding Help for Your Program Files” on page 5-9.

Viewing a Help Summary for Externally Supplied Files

You can get a help summary for all files in a folder. The MathWorks family of products refers to collections of program files that work together as *toolboxes*. A toolbox is a collection of programs and other files that work with MathWorks products. Toolboxes generally augment the capabilities of MATLAB and Simulink in particular domains.

To enable a help summary, each folder containing the externally supplied functions must contain a file named `Contents.m`. Look for a `Contents.m` file in the folder. If this file exists, you can view the help summary as follows:

- 1 Make sure MATLAB can access the folder you want to get the help summary for. Either designate the folder the current folder, or add the folder to the search path.
- 2 Run `help foldername`.

A summary description of all files in the folder displays in the Command Window.

- 3 To display the help summary information in the Help browser instead of the Command Window, run `helpwin foldername`.

You can see the help resources for all files in a folder by running a Help Report from the Current Folder browser. If no `Contents.m` file exists, the Help Report can create one for you.

To see the version number for the collection of files in the folder, run `ver`.

To provide a help summary for your own files, see “Creating a Help Summary for Your Program Files” on page 5-12.

Accessing Help for Externally Supplied Class Files

You can get help for class files created by others if the author included help comments in the class file.

To get help, run `help classname`. The help text can display hyperlinks to help for properties and methods of the class. If help for an item exists, that

help displays in the Command Window when you click the corresponding hyperlink. The help for a class definition looks like that shown in the following example.

```
>> help MyClass
MyClass Sample class

MyClass Properties:
  prop1 - first property
  prop2 - second property

MyClass Methods:
  method1 - first method
  method2 - second method
```

If you run

```
doc classname
```

The same help for the class displays in the Help browser, looking like that shown in the following figure.

[MATLAB File Help: MyClass](#)[View code for MyClass](#)[Default Topics](#)

MyClass

MyClass Sample class

MyClass Properties:

[prop1](#) - first property

[prop2](#) - second property

MyClass Methods:

[method1](#) - first method

[method2](#) - second method

Class Details

Sealed false

Construct on load false

Constructor Summary

[MyClass](#) Sample class

Property Summary

[prop1](#)

[prop2](#) Help text for prop1

Method Summary

[method1](#) Help text for method1

[method2](#) Help text for method2

To get help for related files in the class, you can click the links on the class page in the Command Window or Help browser. You can also run any of the following commands:

- `doc classname.methodname`
- `doc classname.propertyname`
- `doc classname.eventname`

For an example, see “Example of Help for a Externally Supplied Class” on page 5-14.

You can also access the help when you create an instance of a class and open the object in the Variable Editor. Click the class name link, which is below the toolbar in the Variable Editor. For more information, see “Getting Help for Objects and Properties from the Variable Editor” on page 6-33.

To provide help for your class files, see “Adding Help for Classes You Create” on page 5-13.

Accessing Externally Supplied Documentation in the Help Browser

If a program author provided HTML documentation for an application, you can read it in the Help browser or a Web browser. Look in the Help browser **Contents** pane for any entries that are not from the MathWorks.

Pages of externally provided HTML documentation display as entries in the appropriate section of the **Contents** listing. For example, documentation for an author’s blockset appears alphabetized with other blocksets. Documentation for an author’s toolbox appears with other toolbox entries, also in alphabetical order.

To browse the documentation, expand the entry for that blockset or toolbox in the **Contents** pane.

You can search for words in the documentation if the author provided a search database.

Even if you cannot see externally supplied documentation in the **Contents** pane, you can still view it. You can display any HTML page in the Help browser with the `web` function. For example, to display the reference page for `my_function` in the Help browser, obtain the path to it and run:

```
web('C:\myfiles\mytoolbox\my_function.html', '-helpbrowser')
```

If you do not include '-helpbrowser', the page opens in a separate browser window.

Note The `doc` function does *not* display externally supplied reference pages. However, externally-supplied class definitions do appear.

To provide documentation for the Help browser for your own programs, see “Adding HTML Help Files to the Help Browser” on page 5-17.

Accessing Externally Supplied Demos in the Help Browser

If an author provided HTML documentation for demos, you can read that documentation in the Help browser or a Web browser. Demos from external authors appear in the last entry in the **Contents** pane, labeled **Other Demos**.

To browse the externally supplied demos, expand the **Other Demos** entry. If you do not see the **Other Demos** entry, right-click in the **Contents** pane and select **Refresh demos**. If the **Other Demos** entry still does not appear, either the externally supplied demo folder is not on the search path or the folder does not contain the appropriate content.

To troubleshoot problems displaying other authors' demos or to learn how to display your own demos, see “Adding Demos to the Help Browser” on page 5-52.

Providing Your Own Help and Demos

In this section...

“About Providing Help and Demos” on page 5-8

“Adding Help for Your Program Files” on page 5-9

“Adding HTML Help Files to the Help Browser” on page 5-17

About Providing Help and Demos

You can provide help and demos for the files you create and have them appear formatted like the help and demos MATLAB provides. Including help can be worthwhile for you and others with whom you share your files. As the following table explains, you can provide help in various forms. The table presents guidelines for creating the kinds of help that best suit your program files and the people who need to use them.

Type of Help	Description	See
Help comments	<ul style="list-style-type: none"> • Describe individual program files you create • Provide formatted comments at the start of a MATLAB program file • Display the help comments when you type <code>help filename</code> • Easy to provide 	“Adding Help for Your Program Files” on page 5-9
Contents.m file	<ul style="list-style-type: none"> • Describes a collection of program files • Provides a summary file for all files in a folder • Displays the summary when you type <code>help foldername</code> • Can include a version number • Easy to provide • Can be empty to avoid listing folder contents 	<ul style="list-style-type: none"> • “Creating a Help Summary for Your Program Files” on page 5-12

Type of Help	Description	See
MATLAB class files	<ul style="list-style-type: none"> • Describes classes you create • Provides help in the class definition file, and optionally for class methods, properties and events • View the help by running <code>help classname</code> or <code>doc classname</code> • Easy to provide, but requires object-oriented programming knowledge to create classes 	“Adding Help for Classes You Create” on page 5-13
Documentation in the Help browser	<ul style="list-style-type: none"> • Supports graphics, images, stylized text, and page formatting. Suited for how-to and conceptual information that helps others run your files • Can include reference pages for functions and blocks • Can include a search database to support searching your documentation • More effort than providing help in program files • Requires the ability to create HTML files and edit XML files 	“Adding HTML Help Files to the Help Browser” on page 5-17
Demos in the Help browser	<ul style="list-style-type: none"> • Suited for explaining how something works, step-by-step. Supports graphics, images, stylized text, and page formatting • Others can view, edit and run your demos • Can be generated from code scripts • View the demos using the Contents pane in the Help browser 	“Adding Demos to the Help Browser” on page 5-52

Adding Help for Your Program Files

You can include help text in each program file that you create which displays like the help for MATLAB functions when you use the `help` function. For more information, see “Getting Command-Line Help for Externally Supplied

Program Files” on page 5-2. You can also create a help summary to describe all program files within a folder and for MATLAB Classes you define.

- “Providing Help Within a Program File” on page 5-10
- “Creating a Help Summary for Your Program Files” on page 5-12
- “Adding Help for Classes You Create” on page 5-13

Providing Help Within a Program File

You can and should provide help text at the top of any file with a `.m` extension that you create. Help consists of lines of comments at the beginning of a file. For information on formatting the help, see “Basic Parts of a Program File”. For examples of help for program files, open the MATLAB function files you are familiar with in the Editor. For information about formatting help in the Editor, see “Adding Comments” on page 9-40.

At the end of your help text, add the names of related functions on a line that begins with `% See also`. The list of names can include MATLAB functions, toolbox functions, and your own functions. The `help` command displays each of these function names as a hyperlink to its help, if the function exists on the search path.

After the `See also` line, if any, end your help text with a blank line (without a `%`).

You can include hyperlinks (in the form of URLs) to HTML files or Web sites anywhere in your help text. Create hyperlinks by including an HTML `<A> ` anchor element. Within the anchor, use `amatlab:` statement (pronounced *matlabcolon*) to execute a web command. For example:

```
% For more information, see <a href="matlab:  
% web('http://www.mathworks.com')">the MathWorks Web site</a>.
```

When you are connected to the Internet and click the link the MathWorks Web site, MATLAB opens a Web browser window to display the URL.

For related information, see “Displaying Hyperlinks in the Command Window” on page 3-12.

Tip To make your help easy for readers to follow, be consistent in how you structure it. For example, follow the style that MATLAB functions use.

To help you create and manage help for your own files, use the “Generating a Summary View of the Help Components in Functions and Scripts” on page 10-8.

The `collatz.m` example file illustrates help text formatting for a function:

```
function sequence=collatz(n)
% COLLATZ 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
%
%   See also COLLATZPLOT, LENGTH, MYFILES/LENGTH.

sequence = n
...
```

Click [here](#) to add the help examples folder to the search path. Then, when you run `help collatz`, MATLAB displays:

```
COLLATZ 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

See also collatzplot, myfiles/length.
```

If functions listed in the See also line exist on the search path or current folder, MATLAB converts them to lowercase and displays them as hyperlinks. Otherwise, MATLAB prints the function names as they appear in the help text.

Creating a Help Summary for Your Program Files

Provide a summary file for your own collection of program files using the same method as MATLAB. In MATLAB, each folder containing program files includes is a file named `Contents.m` (with a capital C) that lists the functions in the folder with a brief description of each.

Running `help foldername` displays the text from the `Contents.m` file for that folder. The displayed help has hyperlinks to help for the individual functions. Running `helpwin foldername` displays the same information in the Help browser.

To create your own `Contents.m` files:

- In the Editor, display a `Contents.m` file provided with MATLAB to see its structure. Most folders in the program tree contain a `Contents.m` file.
- Read about “Displaying and Updating a Report on the Contents of a Folder” on page 10-11 to learn how to easily create and maintain your `Contents.m` files.
- Provide your own toolbox name, a version, and a date in the first two lines of the `Contents.m` file, which the `ver` function displays:

```
% Toolbox description
% Version xxx dd-mmm-yyyy
```

Do not include any spaces in the date. Use this format: 12-Mar-2010.

Tip If you do *not* want your users to see a summary of your toolbox functions, place an empty `Contents.m` file in the toolbox folder. An empty `Contents.m` causes `help foldername` to report `No help found for foldername`.

The Upslope Area toolbox example folder contains a `Contents.m` file. For more information, see “Learning to Add Help from Examples” on page 5-18.

You can create a categorical listings of functions for the Help browser by marking up your `Contents.m` file and publishing it to HTML. To learn more, see “Creating Function and Block Category Listings” on page 5-38.

Adding Help for Classes You Create

If you create your own MATLAB classes, you can provide help for the class by including comments in the class definition file:

- Provide help about the class in comment lines directly following the `classdef` statement.
- Add a comment line directly after the constructor method for the class.
- Add comments directly after other methods and next to property definitions.

List the properties and methods of the class within the first block of comments after `classdef`. If you format the list in as described here, MATLAB renders the property and method names you list as hyperlinks to their definitions, which appear later in the same file:

- 1 To create links from this section to your class properties, add a line:

```
% Classname Properties:
```

where *Classname* is the name from your `classdef`. Be sure to put a colon (`:`) after `Properties`.

- 2 List your property names (with optional same-line descriptions) on the following lines. For example:

```
% prop1 - first property
% prop2 - second property
```

- 3 List the methods; enter the class name followed by `Methods:` (include a colon). Then, list your methods (with optional same-line descriptions) on the following lines, as follows:

```
% MyClass Methods:
% method1 - first method
% method2 - second method
```

View help for your class in the Command Window:

```
help classname
```

To view the same help for the class in the Help browser, run:

```
doc classname
```

Note You do not need to prepare HTML versions of class definition file help. MATLAB generates an HTML page from each class definition automatically and displays it in the Help browser.

For more information about getting help for classes, see “Accessing Help for Externally Supplied Class Files” on page 5-3. To learn more about how to create class definitions, see “Defining and Organizing Classes”.

Example of Help for a Externally Supplied Class. The following example shows help for a class file, `sads.m`, an example provided with MATLAB documentation. If you create help for your class files, the help should look and work like this example.

Follow these steps to see the help for the example.

1 Make sure you can access the examples folder by either:

- Changing to Designating the folder containing the example file as your current folder:

```
cd(fullfile(matlabroot, 'help', 'techdoc', 'matlab_env', 'examples'))
```

- Adding the examples folder to the search path:

```
addpath(fullfile(matlabroot, 'help', 'techdoc', 'matlab_env', 'examples'))
```

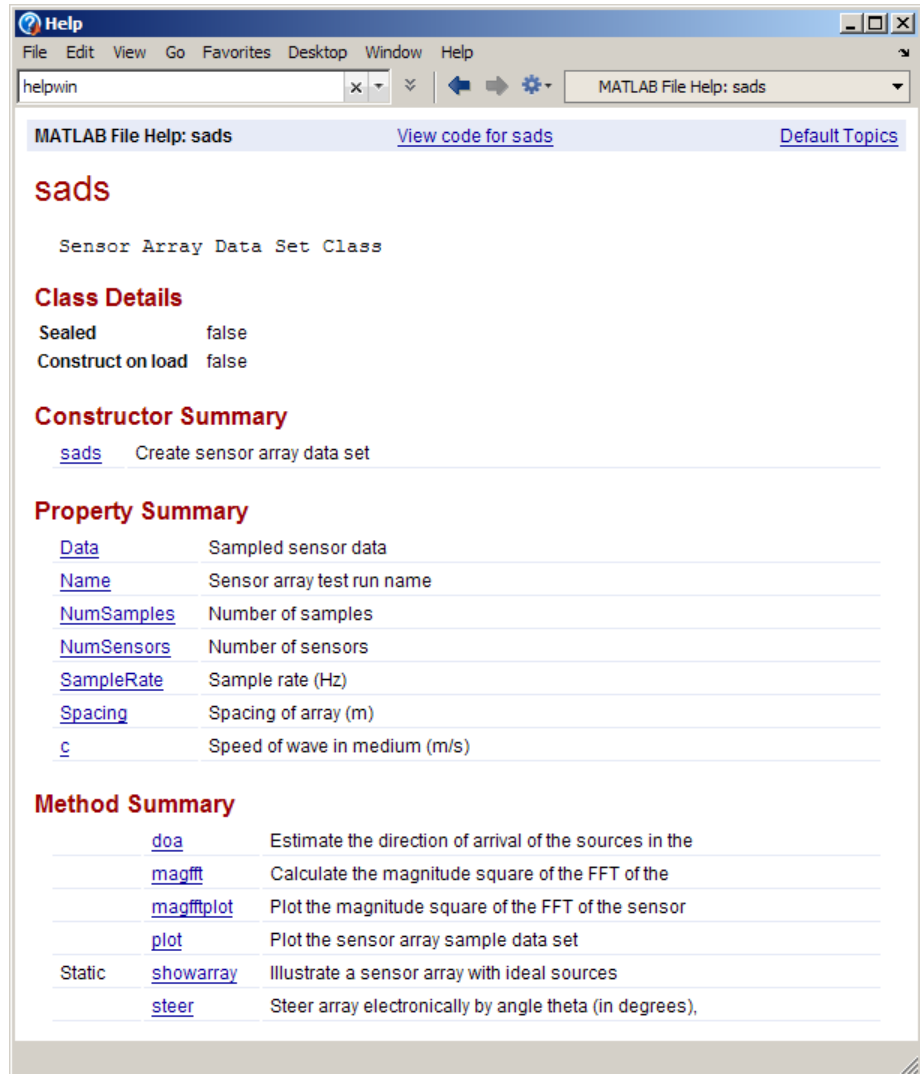
Or, [Click here](#) to add the help examples folder to the search path.

2 Open the class file in the Editor to see the help comments.

```
open('sads.m')
```

3 View help for the `sads` class in the Help browser:

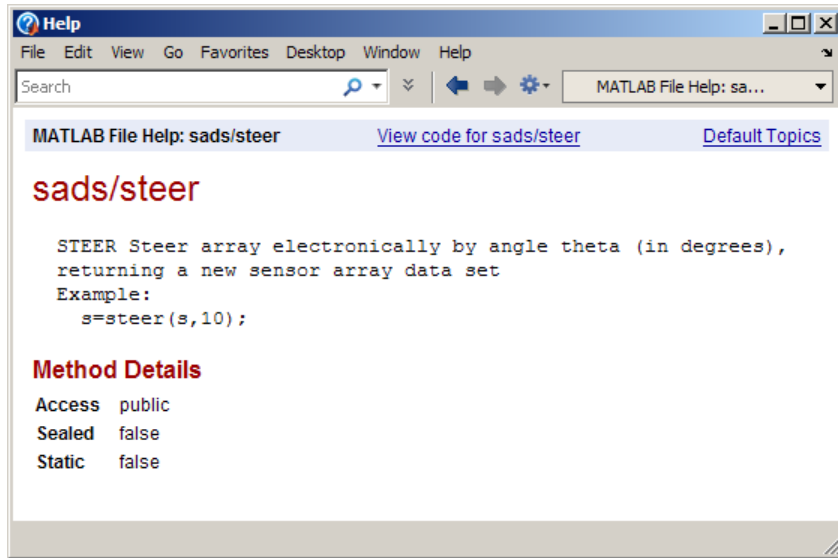
```
doc sads
```



- 4 Access more information by following links on the class help page or by using the `doc` function. For example, to get help for the `steer` method, do one of the following:

- Click the `steer` link under Method Summary.

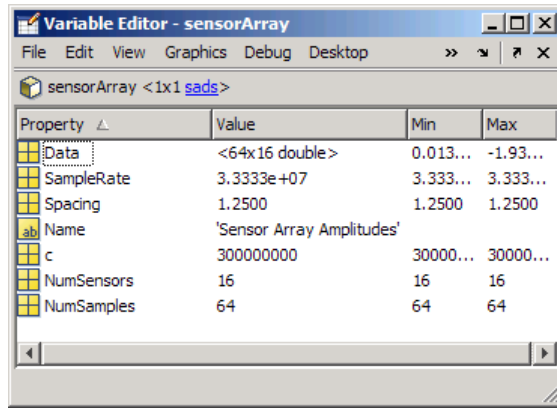
- Run `doc sads.steer`.



You also can open the `sads.m` file by clicking the **View code for sads** link at the top of the `sads` help page. For instructions to display the `sads` help page, see “Example of Help for an Externally Supplied Class” on page 5-14.

- 5 Next, view information about `sads` objects another way. Create an instance of `sads`, for example, `sensorArray`, and then open that object in the Variable Editor:

```
loadparameters
sensorArray=sads(Data, Wavelength, SampleRate, Spacing, Name);
openvar sensorArray  /% or double-click sensorArray in the Current Folder browser
```



Adding HTML Help Files to the Help Browser

MathWorks and third parties provide a rich set of toolboxes, blocksets, and target and link products. Almost all such products come with documentation that displays in the Help browser.

If you create a toolbox that works with MathWorks products—even if it only contains a few functions—you can include with it HTML help files that you and others can access using the Help browser. Providing HTML help files for your toolbox allows you to include figures, diagrams, screen captures, equations, and formatting to make your toolbox more usable. For more information, see these topics:

- “Types of Documentation You Can Provide” on page 5-18
- “Learning to Add Help from Examples” on page 5-18
- “Summary of Creating and Installing HTML Help Files” on page 5-19
- “Organizing Your Documentation” on page 5-20
- “Creating Function Reference Pages” on page 5-33
- “Creating Function and Block Category Listings” on page 5-38
- “Making Your HTML Help Files Searchable” on page 5-45
- “Summary of Workflow for Providing HTML Help Files” on page 5-46

Types of Documentation You Can Provide

Consider providing the following types of content in your documentation:

- A start page for your toolbox (called a “roadmap”)
- A quick introduction to your toolbox (“getting started guide”)
- A detailed explanation of using your toolbox (“user guide”)
- Function or block reference pages
- A list of examples, hyperlinked to the documentation set
- Release notes, describing improvements, limitations, known bugs, and so forth
- PDF versions of your HTML files (typically accessed from the roadmap page)

Except for the PDF version of documentation, each of these types of help is a set of one or more HTML pages that you create in the Editor, word processing software, or an HTML authoring environment. Many such applications can also export their source documents as PDF files.

Note You are free to organize and format your help documentation as you choose. However, if you structure your help files similarly to documentation from MathWorks, people who use it will understand where to find specific types of information.

Learning to Add Help from Examples

To learn how to create documentation for the Help browser, refer to examples. This documentation provides two folders that you can copy. These folders contain:

- Template XML files containing required and optional sections, with explanatory comments

You find this folder in
matlabroot/help/techdoc/matlab_env/examples/templates. Always

work with *copies* of the files it contains when making modifications. You must edit in your own content to the templates to make them usable.

- A complete toolbox with code you can run (called Upslope Area Toolbox), accompanied by extensive HTML documentation that you can view in the Help browser

Find this example in the folder `matlabroot/help/techdoc/matlab_env/examples/upslope`. You can use functions from the toolbox and view the help by adding the folder to the search path. However, if you choose to modify any of the files it contains, copy the entire `upslope` folder to a working folder.

Note Some functionality of Upslope Area Toolbox depends on Image Processing Toolbox. If you have Image Processing Toolbox installed, clicking [here](#) brings you to its documentation.

The following sections primarily discuss the XML template files for `info.xml` and `helptoc.xml`, showing you how to modify them to create a documentation set. The examples folder provides templates only for XML files, not HTML files. Therefore, to understand how the XML files access HTML documentation files and what those files contain, refer to corresponding files in the example Upslope Area Toolbox folder.

Summary of Creating and Installing HTML Help Files

To add your own documentation to the Help browser, you need to:

- 1 Decide what types of documentation you want to provide and create HTML help files for your toolbox. See “Types of Documentation You Can Provide” on page 5-18.
- 2 Create an `info.xml` and `helptoc.xml` files based on examples. See “Organizing Your Documentation” on page 5-20.
- 3 Optionally create a search database to include your HTML help files in the Help browser search results. See “Making Your HTML Help Files Searchable” on page 5-45.

- 4 Add the HTML files to the Help browser, by editing and incorporating XML and other special files you create. For step-by-step instructions, see “Summary of Workflow for Providing HTML Help Files” on page 5-46.
- 5 Provide the help files to your program users, along with instructions for including the files in the Help browser.

To create HTML help files, use the MATLAB Editor, another text editor, or an HTML editing tool. If you have an XML authoring system, you can develop documentation in that environment and export it as HTML files.

If you use the Editor, enabling syntax highlighting and indenting features will help as you author HTML and XML files. The editor can automatically color syntax for .htm, .html, and .xml source files.

Tip To customize the syntax highlighting and indenting in the Editor, select **File > Preferences > Editor/Debugger > Language**, and choose XML/HTML.

Verify how your HTML files appear in the Help browser. To view an HTML help file that you created, use the `web` function. For example, display an HTML file from the set of examples provided for this topic:

```
web(fullfile(matlabroot, 'help', 'techdoc', 'matlab_env' , ...
    'examples', 'upslope', 'html', 'upslope_functions_by_cat.html'))
```

Organizing Your Documentation

After you decide which types of documentation to show in the Help browser, you need to provide HTML and XML files, and link them to work together. The following sections describe how to set up your help documentation.

- “Setting Up a Help Folder” on page 5-21
- “XML Files Required to Add Documentation and Demos” on page 5-23
- “Identifying a Help Folder: the info.xml File” on page 5-24
- “Customizing the info.xml Template File” on page 5-26
- “More About the info.xml File” on page 5-27

- “Creating the Table of Contents File: helptoc.xml” on page 5-29
- “More About the helptoc.xml File” on page 5-32

Note To view the content and organization of the Upslope Area toolbox documentation example, place it on the search path by clicking [here](#), or run this command:

```
addpath(fullfile(matlabroot, 'help', 'techdoc', 'matlab_env', 'examples', 'upslope'))
```

Setting Up a Help Folder. Make a folder to hold HTML and XML files. The folder can contain subfolders to organize HTML and image files. It can also contain MATLAB program files for your toolbox, or you can locate your code files in a different folder. A typical toolbox folder contains the following kinds of elements:

Note Folders and file names that you specify are in italics in this listing. Folders are prefixed with a forward slash (/). On the right side, comments that are in italics are directives for you to follow.

/mytoolbox

Top level folder for toolbox documentation; can also contain your program files.

info.xml

Indicates to MATLAB that this folder contains Help browser documentation, and points to content. *Required; must have this file name.*

<code>*.m</code>	Program code, data, GUI, and other files for your toolbox. You can also locate program files in a separate folder or a subfolder of this one that you place on the search path. Also include a <code>Contents.m</code> file here summarizing the program files.
<code>*.mat</code>	
<code>*.fig</code>	
<code>...</code>	
<code>/html</code>	Optional subfolder for your HTML documentation content; it can have any name, which must be specified in your <code>info.xml</code> file.
<code>helptoc.xml</code>	Defines hierarchy of help files. <i>Required; must have this file name.</i>
<code>mytoolbox_ product_page.html</code>	Roadmap (start page) for your documentation. <i>Use folder name followed by "_product_page.html". Optional but recommended.</i>
<code>getting_started_1.html</code>	Optional content for getting started guide.
<code>...</code>	
<code>getting_started_n.html</code>	
<code>user_guide_1.html</code>	Content for user guide.
<code>...</code>	
<code>user_guide_n.html</code>	
<code>helpfuncbycat.html</code>	Optional functions-by-category listing containing links to function reference HTML files.
<code>release_note_1.html</code>	Release notes files.
<code>...</code>	
<code>release_note_n.html</code>	
<code>/graphics</code>	Optional subfolder for images used in HTML pages; when you use a subfolder, HTML <code><image></code> elements must specify this path to image files.

<i>image_1.png</i>	Bitmap graphics files (usually <i>.gif</i> , <i>.png</i> , or <i>.jpg</i>). Do not store custom icons for the TOC here, as they cannot be found.
<i>...</i>	
<i>image_n.gif</i>	
<i>/reference</i>	Optional subfolder for function or block reference pages; when you use a subfolder, HTML <code><A></code> hyperlinks to reference pages must specify this path.
<i>function_1.html</i>	Function/block reference page files.
<i>...</i>	
<i>function_n.html</i>	

XML Files Required to Add Documentation and Demos. The Help Browser relies on several files coded in XML to recognize and present the contents of documentation and demos. These files always have the same names:

- *info.xml* — Required file that indicates that a folder contains documentation or demos.
- *helptoc.xml* — Required file that provides a structure for presenting the documentation set in the **Contents** pane.
- *demos.xml* — Optional file to add your demos to the **Other Demos** section of the **Contents** pane. See “Adding Demos to the Help Browser” on page 5-52.

In addition, you must create and provide the HTML content pages referenced by these files and graphic files for images that they display.

Identifying a Help Folder: the info.xml File. The `info.xml` file specifies the content type, name, and icon to display for your documentation set. It also identifies where to find your HTML help files, and defines items you add to the **Start** button. You must create a file named `info.xml` for each toolbox you document. When you include a file having this name in a folder and then add that folder to the search path, MATLAB adds the documentation for your toolbox to the Help browser **Contents** pane. The folder that `info.xml` identifies as `<help_location>` must contain your HTML documentation and a file named `helptoc.xml`.

The following listing is a template for `info.xml` that you can adapt to describe your toolbox:

```
<productinfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="optional">
  <?xml-stylesheet type="text/xsl"href="optional"?>
  <!-- info.xml file for the mytoolbox toolbox -->
  <!-- Version 1.0 -->
  <!-- Copyright (date) (owner)..-->

  <!-- Supply the following six elements in the order specified -->
  <!-- (Required) element; matlabrelease content is not currently used -->
  <matlabrelease>2010a</matlabrelease>
  <!-- (Required) The name element appears in the Contents pane -->
  <name>MyToolbox</name>
  <!-- (Required) The type element identifies your package; pick one: -->
  <!-- matlab, toolbox, simulink, blockset, links_targets -->
  <type>toolbox</type>
  <!-- (Optional) icon file to display in the Start button -->
  <icon>sampleicon.gif</icon>
  <!-- (Required if you supply help) relative path to help (HTML) folder -->
  <help_location>./HTMLFolderName</help_location>
  <!-- (Required if you supply help) icon used in the Help browser TOC -->
  <help_contents_icon>$toolbox/matlab/icons/bookicon.gif</help_contents_icon>

  <!-- - - - - - Start menu - - - - - -->
  <!-- Optional list of entries to display on Start Menu -->
  <!-- Callback items are function calls or commands for toolbox -->
  <!-- Refresh the Start button to see your entries -->
  <!-- Remove this list if you do not want a Start button entry -->
```

```

<list>
  <listitem>
    <!-- The label provides the text for this menu item -->
    <label>MyToolbox Documentation</label>
    <!-- This callback is a command to open your documentation -->
    <callback>
web ./html/mytoolbox_product_page.html -helpbrowser
    </callback>
    <!-- Menu item icon (a toolbox icon from the help browser ) -->
    <icon>$toolbox/matlab/icons/bookicon.gif</icon>
  </listitem>
  <listitem>
    <!-- A menu item label for a opening a GUI -->
    <label>MyToolbox GUI</label>
    <!-- A command to open a GUI, if any -->
    <callback>mytoolboxgui</callback>
    <!-- The icon for this menu item -->
    <icon>$toolbox/matlab/icons/figureicon.gif</icon>
  </listitem>
  <listitem>
    <!-- A menu item label for a opening a demo -->
    <label>MyToolbox Demo</label>
    <!-- A command to open a demo if any -->
    <callback>mytoolboxdemo</callback>
    <!-- The icon for this menu item -->
    <icon>HelpIcon.DEMOS</icon>
  </listitem>
  <listitem>
    <!-- Include call to open your Web site, if any -->
    <label>MyToolbox Web Site</label>
    <callback>web http://www.mathworks.com -browser</callback>
    <icon>$docroot/techdoc/matlab_env/examples/webicon.gif</icon>
  </listitem>
  <!-- Add listitems for other features of your toolbox... -->
</list>
<!-- - - - - - Start menu - - - - - -->
</productinfo>

```

Note To avoid XML validation errors, include all required elements in the order specified by the template file. If you are not adding a toolbox to the **Start** button menu, omit the lines bracketed by

```
<!-- - - - - - Start menu - - - - - -->
```

For more information, see “Addressing Validation Errors for info.xml Files” on page 5-61

Replace the contents within the `<matlabrelease>`, `<name>`, `<type>`, and `<help_location>` elements with appropriate text for your toolbox. The contents of `<help_location>` is folder name, which usually includes a relative path. Typically, you place the help folder within the folder containing the `info.xml` file. You can include comments in `info.xml` or any other XML file. For example, you can add copyright and contact information. Lines starting with `<!--` and ending with `-->` contain comments.

When you add a help folder to the Help browser for the first time, take the following actions:

1 Add your toolbox or blockset folder to the search path

Make sure that the folder you are adding is *not* your current folder when you perform this step.

2 Open MATLAB Preferences from the **File** menu.

3 Click **Help**, and then select the **All Products** button.

After MATLAB has identified your folder as a toolbox or blockset and displayed it in the Help browser, you can remove products from the Help browser you do not need to show.

Customizing the info.xml Template File. To specify and structure your own documentation content, copy, edit, and save the template file, as follows:

1 In the Editor, open the XML template. You can either:

- Copy the preceding listing and paste it into a new blank document.

- Copy the `info_template.xml` template example file to your current folder:

```
copyfile(fullfile(matlabroot,'help','techdoc','matlab_env', ...
    'examples','templates','info_template.xml'),pwd), ...
fileattrib('info_template.xml','+w')
```

or click [here](#) to copy the template. Then, open the copy in the Editor.

- 2 Save the file as `info.xml` in your toolbox folder. Saving as a `.xml` file enables Editor syntax highlighting.
- 3 Replace italicized text in the listing with your own content.
- 4 If you are not adding any items to the **Start** menu, delete the - - Start menu - - section. If you want to customize the Start menu, you must modify the `listitem` elements. For instructions, see “Adding Your Own Toolboxes to the Start Button” on page 2-96.
- 5 Resave the `info.xml` file when you finish making changes.

More About the info.xml File. The `info.xml` file adds the HTML help files to the Help browser and items to the **Start** menu. The following table describes the example `info.xml` file provided as a toolbox template. The source file is `matlabroot/help/techdoc/matlab_env/examples/templates/info_template.xml`.

XML Tag	Description	Value in Template	Notes
<code><matlabrelease></code>	Release of MATLAB.	R2010a	Required. Not currently parsed, but indicates when you added help files.
<code><name></code>	Title of toolbox.	mytoolbox	Required. The name of your toolbox that appears in the Help browser Contents pane.

XML Tag	Description	Value in Template	Notes
<type>	Determines the toolbox location in the Help browser Contents .	toolbox	Required. Allowable values: matlab, toolbox, simulink, blockset, links_targets, other. The Upslope Area toolbox example appears with other toolboxes. The entry has the orange book icon used for toolboxes.
<icon>	Icon for your toolbox help in the Start button.	sampleicon.gif	If you add your toolbox to the Start button options and include a help entry there, specify an icon image file. For more information, see the <list><listitem> description.
<help_location>	Location of help files	./HTMLfolderName	Name of subfolder containing helptoc.xml and HTML help files you provide for your toolbox. If not a subfolder, specify the path to help_location relative to the info.xml file. If you provide HTML help files for multiple toolboxes, each help_location must be a different folder.
<help_contents_icon>	Icon to display in Help browser Contents pane	\$toolbox/matlab/icons/bookedon.gif	Required if you supply HTML help files.
<list> <listitem> ...	Entries for Start button	various	If you also want your toolbox to appear as a Start button option, add at least one listitem. For details, see “Adding Your Own Toolboxes to the Start Button” on page 2-96.

When you set up an XML file, make sure that:

- You include all required entries;
- The entries are in the same order as in the preceding list and in the template;
- File and folder names in the XML exactly match the names of your files and folders and use upper and lower case letters identically.

For examples, look at the `info.xml` file for any MathWorks product. To view one of these files:

- 1** Select **Start > Desktop Tools > View Start Button Configuration Files** files.
- 2** From the resulting Start Button Configuration Files dialog box, select the product.
- 3** Click **Open** to view the `info.xml` file in the Editor.

Note The `info.xml` files for MathWorks products contain custom constructs and features that externally supplied `info.xml` files cannot implement.

Creating the Table of Contents File: `helptoc.xml`. You must also create a file named `helptoc.xml`. Place this file in the folder containing your HTML documentation files. This folder is designated as `<help_location>` in your `info.xml` file. Within a top-level `<toc>` element, nest `<tocitem>` elements to define the structure of your table of contents. This template for `helptoc.xml` explains its organization:

```
<?xml version='1.0' encoding="utf-8"?>
<toc version="2.0">
<!-- First tocitem specifies top level in Help browser Contents pane -->
<!-- This can be a roadmap page, as shown below, or a content page -->

    <tocitem target="mytoolbox_product_page.html">MyToolbox Toolbox
        <!-- Nest tocitems to create hierarchical entries in Contents-->
        <!-- To include icons, use the following syntax for tocitems: -->
        <!-- <tocitem target="foo.html" image="HelpIcon.NAME" --> -->
        <!-- Title-of-Section </tocitem> -->
```

```

<!-- where NAME is one of the following (use capital letters): -->
<!-- FUNCTION, USER_GUIDE, EXAMPLES, BLOCK, GETTING_STARTED, -->
<!-- DEMOS, RELEASE_NOTES -->
<!-- Icon images used for these entries are also stored in -->
<!-- matlabroot/toolbox/matlab/icons -->
<!-- A Getting Started Guide usually comes first -->
<tocitem target="mytbx_gs_top.html" image="HelpIcon.GETTING_STARTED">
    Getting Started with the MyToolbox Toolbox
    <tocitem target="mytbx_reqts_example.html">System Requirements
    </tocitem>
    <tocitem target="mytbx_features_example.html">Features
        <!-- 2nd and lower TOC levels usually have anchor IDs -->
        <tocitem target="mytbx_feature1_example.htm#10187">Feature 1
        </tocitem>
        <tocitem target="mytbx_feature2_example.htm#10193">Feature 2
        </tocitem>
    </tocitem>
</tocitem>
<!-- User Guide comes next -->
<tocitem target="mytbx_ug_intro.html"
    image="HelpIcon.USER_GUIDE">MyToolbox User Guide
    <tocitem target="mytbx_ch_1.html">Setting Up MyToolbox
    </tocitem>
    <tocitem target="mytbx_ch_2.html">Processing Data
    </tocitem>
    <tocitem target="mytbx_ch_3.html">Verifying MyToolbox outputs
        <tocitem target="mytbx_ch_3a.html">Handling Test Failures
        </tocitem>
    </tocitem>
</tocitem>
<!-- Function reference next -->
<!-- The first file lists all of the functions, categorizing them -->
<tocitem target="function_categories.html">Functions
    <!-- First category, with link to anchor in above page -->
    <tocitem target="function_categories.html#1">First Category
        <!-- Inside category, list its functions alphabetically -->
        <tocitem target="function_1.html">function_1</tocitem>
        <tocitem target="function_2.html">function_2</tocitem>
        <!-- ... -->
    </tocitem>
</tocitem>

```

```

<!-- Second category, with link to anchor in above page -->
<tocitem target="helpfuncbycat.html#2">Second Category</tocitem>
  <!-- Inside category, list its functions alphabetically -->
  <tocitem target="function_3.html">function_3</tocitem>
  <tocitem target="function_4.html">function_4</tocitem>
  <!-- ... -->
</tocitem>
<!-- Third category, with link to anchor in above page -->
  <tocitem target="helpfuncbycat.html#3">Third category</tocitem>
  <!-- ... -->
</tocitem>
</tocitem>
<!-- Optional List of Examples, with hyperlinks to examples in other files -->
<tocitem target="mytbx_example.html"
  image="HelpIcon.HelpIcon.EXAMPLES">Mytoolbox Examples
</tocitem>
<!-- Optional link or links to your or other Web sites -->
<tocitem target="http://www.mathworks.com"
  image="$toolbox/matlab/icons/webicon.gif">
  MyToolbox Web Site (Example only: goes to mathworks.com)
</tocitem>
</tocitem>
</toc>

```

Be sure that file and path names exactly match those of the files and folders they identify and use upper and lower case letters identically. Your `helptoc.xml` can be shorter or longer than the template. The size of the file depends on the structure of your documentation and how many HTML files it contains.

Most tables of contents have two to four hierarchical levels. Lower levels can either specify subheadings within the top-level HTML file or separate HTML files. A `<tocitem>` can link to subheadings by specifying *anchor IDs* for them. For example, this one,

```
<tocitem target="mytbx_feature1_example.html#107">Feature 1</tocitem>
```

specifies a link to the named anchor #107 within the file `mytbx_feature1_example.html`. Anchor IDs always start with a pound sign (#).

Create anchors for referencing headings or other HTML content with `Any content` elements. If your documentation set includes HTML files that are *not* listed in `helptoc.xml`, at least one file found in the table of contents must contain hyperlinks to them, so that readers can find them. For related information, see “Creating Function and Block Category Listings” on page 5-38.

To customize the `helptoc` template file:

- 1 In the Editor, open the `helptoc` XML template. You can either:
 - Copy the preceding listing and paste it into a new blank document.
 - Copy the `helptoc_template.xml` template example file to your current folder:








```
copyfile(fullfile(matlabroot,'help','techdoc','matlab_env', ...
    'examples','templates','helptoc_template.xml'),pwd), ...
fileattrib('helptoc_template.xml','+w')
```

or click [here](#) to copy the template. Then, open the copy in the Editor.

- 2 Save the file as `helptoc.xml` in your toolbox folder. Saving as a `.xml` file enables Editor syntax highlighting.
- 3 Replace italicized text in the listing with your own content.
- 4 Resave the `helptoc.xml` file when you finish making changes.

More About the `helptoc.xml` File. The `info.xml` file inserts your toolbox in the alphabetic listing of toolboxes or blocksets in the **Contents** pane. The `helptoc.xml` defines a hierarchy of entries within it. Each `<tocitem>` entry in the `helptoc.xml` file references one of your HTML help files or anchor IDs within that product entry. The `helptoc_template.xml` file that is provided as an example has the structure most toolboxes use.

You can display icons for your **Contents** pane entries within your toolbox. To use standard MathWorks Help browser icons, include any of the following icons as `image` attributes for `<tocitem>` elements.

Icon	Use For	Image Tag String
	Getting Started Guides	HelpIcon.GETTING_STARTED
	User Guides	HelpIcon.USER_GUIDE
	Functions	HelpIcon.FUNCTION
	Blocks	HelpIcon.BLOCK
	Examples	HelpIcon.EXAMPLES
	Release Notes	HelpIcon.RELEASE_NOTES
	Demos	HelpIcon.DEMOS

To make your documentation consistent with MathWorks documentation, organize your table of contents entries in the preceding order .

Include the icons as `image` attributes in top-level TOC entries. If you provide a roadmap page, also include icons for second-level TOC entries under the roadmap. Nest `tocitem` entries for the target chapters or pages within each such TOC entries, for example:

```
<tocitem target="get_start_top.html" image="HelpIcon.GETTING_STARTED">About Mytoolbox
  <tocitem target="get_start_capabilities.html"> Capabilities </tocitem>
  <tocitem> ... </tocitem>
  ...
</tocitem>
```

The markup indicates that you have a file called `get_start1.html` that begins a getting started guide. The HTML pages it contains appear next, coded as nested `tocitem` elements.

Creating Function Reference Pages

Unless you prefer to hide a function from your users, provide an HTML reference page for it. If your program (`.m`) files contain help text, you already have the content you need to add reference pages to the Help browser. If your

program files do not yet include help text, consider adding help as a first step. For details, see “Adding Help for Your Program Files” on page 5-9.

You can create a reference page in an HTML authoring environment by importing the help text for a function and formatting the text. For example, you need to remove the percent sign (%) character from the beginning of each line of text, and make sure that spaces separate words. You can then format headings, words, phrases, and examples for HTML display. Finally, you can add `image` attributes to display graphics such as GUIs, diagrams, and graphic output from your code, and hyperlinks to *See also* items and other related documentation.

Consider creating reference pages from within MATLAB. You can use the capability of MATLAB to publish program scripts directly to HTML documents, as described in Chapter 11, “Publishing MATLAB Code”.

To transform help text from a program file into HTML using the `publish` command:

- 1** Copy the help text into a new file, and remove the code that implements the function.
- 2** Save this file as a MATLAB script (which has no initial function declaration).
- 3** Format the help text using *code cell notation*.
- 4** Publish the script as an HTML file.

The following listings illustrate such a transformation for the `upslopeArea.m` function from the example Upslope Area toolbox files to a cell script version of its help text, `upslopeArea_help.m`. Find the original program files in the examples folder `matlabroot/help/techdoc/matlab_env/examples/upslope`. See the subfolder `matlabroot/help/techdoc/matlab_env/examples/upslope/html` for the cell-scripted versions and the HTML generated from publishing those files.

After formatting and saving `upslopeArea_help.m`, the command

```
publish upslopeArea_help.m
```


generates a file named `upslopeArea_help.html` in a subfolder. By default, this folder is named `html`, but you can specify a different name for the folder by configuring the `publish` command. (Placing all your reference pages in the same folder simplifies accessing them.)

Original Upslope Area Toolbox Function `upslopeArea.m` file.

```
% upslopeArea Upslope area measurements for a DEM
%
% DESCRIPTION
% A = upslopeArea(E, T) computes the upslope area for each pixel of the
% DEM matrix, E. T is the sparse system of linear equations computed
% by flowMatrix; it represents the distribution of flow from pixel to
% pixel. A contains the upslope area for each corresponding pixel of E.
%
% Note: Connected groups of NaN pixels touching the border are treated as
% having no contribution to flow.
%
% REFERENCE
% Tarboton, "A new method for the determination of flow
% directions and upslope areas in grid digital elevation models," Water
% Resources Research, vol. 33, no. 2, pages 309-319, February 1997.
%
% ALGORITHM NOTES
% The Tarboton paper is not very specific about the handling of plateaus. For
% details of how plateaus are handled in this code, see the algorithm notes for
% the function flowMatrix. In particular, see the subfunction
% plateau_flow_weights in flowMatrix.m.
%
% EXAMPLE
% s = load('milford_ma_dem');
% E = s.Zc;
% R = demFlow(E);
% T = flowMatrix(E, R);
% A = upslopeArea(E, T);
% imshow(log(A), [])
%
% See also demFlow, dependenceMap, fillSinks, flowMatrix, postprocessPlateaus.
%
% Steven L. Eddins
```

```
% Copyright 2007-2009 The MathWorks, Inc.

function A = upslopeArea(E, T)

requiresIPT(mfilename);

% Right-side vector is normally all ones, reflecting an equal contribution
% to water flow originating in each pixel.
rhs = ones(numel(E), 1);

% Connected groups of NaN pixels that touch the border do not contribute
% to water volume.
mask = borderNans(E);
rhs(mask(:)) = 0;

A = T \ rhs;
A = reshape(A, size(E));
```

Upslope Area Toolbox Reference Page Script upslopeArea_help.m file.

```
%% upslopeArea
% Upslope area measurements for a DEM
%
%% Description
% |A = upslopeArea(E, T)| computes the upslope area for each pixel of the
% DEM matrix, |E|. |T| is the sparse system of linear equations computed
% by |flowMatrix|; it represents the distribution of flow from pixel to
% pixel. |A| contains the upslope area for each corresponding pixel of |E|.
%
% Note: Connected groups of NaN pixels touching the border are treated as
% having no contribution to flow.
%
%% Reference
% Tarboton, "A new method for the determination of flow
% directions and upslope areas in grid digital elevation models," Water
% Resources Research, vol. 33, no. 2, pages 309-319, February 1997.
%
%% Algorithm notes
% The Tarboton paper is not very specific about the
```

```
% handling of plateaus. For details of how plateaus are handled in this
% code, see the algorithm notes for the function |flowMatrix|. In
% particular, see the subfunction |plateau_flow_weights| in |flowMatrix.m|.
%
%% Example
s = load('milford_ma_dem');
E = s.Zc;
R = demFlow(E);
T = flowMatrix(E, R);
A = upslopeArea(E, T);
imshow(log(A), [])

%% See also
% <demFlow_help.html |demFlow|>, <dependenceMap_help.html |dependenceMap|>,
% <fillSinks_help.html |fillSinks|>, <flowMatrix_help.html |flowMatrix|>,
% <postprocessPlateaus_help.html |postprocessPlateaus|>.

%%
% Copyright 2007-2009 The MathWorks, Inc.
```

As you see, the script file, `upslopeArea_help.m`, does not contain the lines of code that implement the function or comments embedded in that code. However, the file does contain code for the example of using the function and all the help text. The *See also* entries to other toolbox functions are hyperlinks, which you manually edit to use the syntax `<function_name_help.html function_name>`.

Published Upslope Area Toolbox Reference Page `upslopeArea_help.html` File. When you show the published output file with `web(upslopeArea_help.html)`, the beginning of the reference page resembles the following figure.

upslopeArea

Upslope area measurements for a DEM

Contents

- [Description](#)
- [Reference](#)
- [Algorithm notes](#)
- [Example](#)
- [See also](#)

Description

`A = upslopeArea(E, T)` computes the upslope area for each pixel of the DEM matrix, `E`. `T` is the sparse system of linear equations computed by `flowMatrix`; it represents the distribution of flow from pixel to pixel. `A` contains the upslope area for each corresponding pixel of `E`.

Note: Connected groups of NaN pixels touching the border are treated as having no contribution to flow.

Reference

Tarboton, "A new method for the determination of flow directions and upslope areas in grid digital elevation models," *Water Resources Research*, vol. 33, no. 2, pages 309-319, February 1997.

Algorithm notes

The Tarboton paper is not very specific about the handling of plateaus. For details of how plateaus are handled in this code, see the algorithm notes for the function `flowMatrix`. In particular, see the subfunction `plateau_flow_weights` in `flowMatrix.m`.

Example

```
s = load('milford_ma_dem');
E = s.Zc;
R = demFlow(E);
T = flowMatrix(E, R);
A = upslopeArea(E, T);
imshow(log(A), [])
```

Near the end of the published reference page, a screen capture from `imshow` appears, automatically inserted by `publish`.

Creating Function and Block Category Listings

To make your reference pages more useful, also include a **Functions** (or **Blocks**, for blocksets) entry for them in the **Contents** pane of the Help

browser. Expanding one of these entries can display a list of categories. Each category lists the associated functions (or blocks), along with a brief description of the category and descriptions of each function (or block).

If you supply reference help files, you can provide a classified listing of them. HTML help summaries are similar to `Contents.m` files, but display in the Help browser. If you already have a `Contents.m` file that lists all your public functions, you can use it as the basis for creating a categorical listing in HTML. If you do not have a `Contents.m` file, consider creating one to round out your toolbox. For more information, see “Creating a Help Summary for Your Program Files” on page 5-12.

To include a Function-by-Category listing, create an HTML page for it. Use the following example to learn how to edit and mark up your `Contents.m` file, and then publish it to HTML. You can name the output file `helpfuncbycat.html`, as shown, or whatever you prefer. Within `Contents.m`, organize your functions or blocks into categories that you define. Each category begins a new cell. When you publish the file, each category displays as a heading and has an anchor ID (from #1 to #n).

The `helptoc_template.xml` file use category names and anchor IDs in `<tocitem>` elements in its reference section. In the template file, the section for function reference includes links to the categorical listing page, category anchors within it, and individual reference pages.

The following example shows the section of the `helptoc.xml` template file that organizes function reference pages. Publishing `helpfuncbycat.m` created anchor IDs #1, #2, #3, ... in output file `helpfuncbycat.html` to which some `<tocitem>` elements refer:

```
<toc version="2.0">
  <!-- ... -->
  <!-- Function reference next -->
  <!-- The first file lists all of the functions, categorizing them -->
  <tocitem target="function_categories.html">Functions
    <!-- First category, with link to anchor in above page -->
    <tocitem target="function_categories.html#1">First Category
      <!-- Inside category, list its functions alphabetically -->
      <tocitem target="function_1.html">function_1</tocitem>
      <tocitem target="function_2.html">function_2</tocitem>
```

```
        <!-- ... -->
    </tocitem>
    <!-- Second category, with link to anchor in above page -->
    <tocitem target="helpfuncbycat.html#2">Second Category</tocitem>
        <!-- Inside category, list its functions alphabetically -->
        <tocitem target="function_3.html">function_3</tocitem>
        <tocitem target="function_4.html">function_4</tocitem>
        <!-- ... -->
    </tocitem>
    <!-- Third category, with link to anchor in above page -->
        <tocitem target="helpfuncbycat.html#3">Third category</tocitem>
        <!-- ... -->
    </tocitem>
</tocitem>
</toc>
```

Tip Copy the preceding XML code and paste it into an editor. Delete any `<tocitem> ... </tocitem>` lines that you do not need, and replace text italicized in the listing with your own content. Then, paste your code into `helptoc.xml`, replacing the template content section for reference pages displayed here.

Italics in the listing indicate strings you need to replace with your own category, file, function, and anchor names and other text. If you place help files for functions or blocks in a subfolder of the one containing your `helptoc.xml` file, include a relative path in the `target` attribute for each `<tocitem>`. For example, if you place function reference pages in a subfolder called `/reference`, you would specify the target as follows:

```
<tocitem target="./reference/function_1.html">function_1</tocitem>
```

Adding Function Category Listings: Upslope Area Toolbox Example.

As mentioned previously, a functions-by-category listing works like a `Contents.m` file. The following example shows how the `Contents.m` file for the Upslope Area toolbox example was marked up and published to create an HTML page that categorizes the toolbox functions and links each function to its reference documentation.

Note If you perform the following procedure, first copy the Upslope Area toolbox Contents.m file to a working folder so you do not overwrite the file or the files upslope_functions_by_cat.m and upslope_functions_by_cat.html that following the example generates.

1 Edit the original Upslope Area toolbox Contents.m file:

```
% Upslope Area Toolbox
% Version 2.0 09-Dec-2009
%
% Requires Image Processing Toolbox(TM).
%
% Flow Direction.
% demFlow          - Downslope flow direction for a DEM
% facetFlow        - Facet flow direction
% flowMatrix       - Linear equations representing water flow
% pixelFlow        - Downslope flow direction for DEM pixels
%
% Preprocessing and Postprocessing.
% borderNans       - Find NaNs connected to DEM border
% fillSinks        - Fill interior sinks in a DEM
% postprocessPlateaus - Replace upslope areas for plateaus with mean value
%
% Hydrological Applications.
% dependenceMap    - Dependence map for water flow in a DEM
% influenceMap     - Influence map for water flow in a DEM
% upslopeArea      - Upslope area measurements for a DEM
%
% Display.
% visDemFlow       - Visualize flow directions in a DEM
% visMap           - Visualize influence or dependence map for a DEM
%
% Data.
% milford_ma_dem.mat - Sample DEM data provided by USGS and distributed
%                    via Geo Community (geoworld.com), a USGS data
%                    distribution partner. The data set is a 1:24,000-scale
%                    raster profile digital elevation model. Download the
%                    "Milford" file from the "Digital Elevation Models (DEM)
```

```
% - 24K Middlesex County, Massachusetts, United States"
% page:
%
% http://data.geocomm.com/catalog/US/61059/526/group4-3.html
%
% natick_ned* - Sample 1/3 arc-second DEM data for a region in Natick,
% Massachusetts. Downloaded from the The National Map
% Seamless Server (http://seamless.usgs.gov/index.php).
%
%
% Steven L. Eddins
% Copyright 2007-2009 The MathWorks, Inc.
```

2 In the Editor, mark up `Contents.m` as follows:

- a** Add a top-level heading, *Functions by Category*.
- b** Format the five categories (*Flow Direction*, *Preprocessing and Postprocessing*, *Hydrological Applications*, *Display*, and *Data*) with double percent signs (`%%`). Doing so turns the sections into code cells, which become section headings in HTML.
- c** Place an asterisk (`*`) in front of each function name to mark it as a bullet in HTML.
- d** Format each function name as a hyperlink to its own reference page. In the Upslope example, function reference pages were created by extracting the function help text to files called *functionName_help.m*, which were then published as *functionName_help.html*.

The complete markup of `Contents.m` into a functions-by-category source listing looks like the following example:

```
%% Functions by Category
% Upslope Area Toolbox
% Version 2.0 09-Dec-2009
%
% Requires Image Processing Toolbox(TM).
%
```



```

%% Flow Direction
% * <demFlow_help.html |demFlow|> - Downslope flow direction for a DEM
% * <facetFlow_help.html |facetFlow|> - Facet flow direction
% * <flowMatrix_help.html |flowMatrix|> - Linear equations representing water flow
% * <pixelFlow_help.html |pixelFlow|> - Downslope flow direction for DEM pixels
%
%% Preprocessing and Postprocessing
% * <borderNans_help.html |borderNans|> - Find NaNs connected to DEM border
% * <fillSinks_help.html |fillSinks|> - Fill interior sinks in a DEM
% * <postprocessPlateaus_help.html |postprocessPlateaus|> - Replace upslope areas
%                               for plateaus with mean value
%
%% Hydrological Applications
% * <dependenceMap_help.html |dependenceMap|> - Dependence map for water flow in a DEM
% * <influenceMap_help.html |influenceMap|> - Influence map for water flow in a DEM
% * <upslopeArea_help.html |upslopeArea|> - Upslope area measurements for a DEM
%
%% Display
% * <visDemFlow_help.html |visDemFlow|> - Visualize flow directions in a DEM
% * <visMap_help.html |visMap|> - Visualize influence or dependence map for a DEM
%
%% Data
% * milford_ma_dem.mat - Sample DEM data provided by USGS and distributed
% via Geo Community (geoworld.com), a USGS data
% distribution partner. The data set is a 1:24,000-scale
% raster profile digital elevation model. Download the
% "Milford" file from the "Digital Elevation Models (DEM)
% - 24K Middlesex County, Massachusetts, United States"
% page at http://data.geocomm.com/catalog/US/61059/526/group4-3.html.
% * natick_ned* - Sample 1/3 arc-second DEM data for a region in Natick,
% Massachusetts. Downloaded from the The National Map Seamless Server
% (http://seamless.usgs.gov/index.php).
%
%% Source
% Steven L. Eddins
% Copyright 2007-2009 The MathWorks, Inc.
    
```

- 3** Save your formatted file as `upslope_functions_by_cat.m` in your current folder (in this case, called `helptests`).

4 Publish the file, and view the resulting HTML file:

```
publish upslope_functions_by_cat.m
ans =
C:\myfiles\upslope\helptests\upslope_functions_by_cat.html

web(ans)
```

Functions by Category

Upslope Area Toolbox Version 2.0 09-Dec-2009

Requires Image Processing Toolbox™.

Contents

- [Flow Direction](#)
- [Preprocessing and Postprocessing](#)
- [Hydrological Applications](#)
- [Display](#)
- [Data](#)
- [Source](#)

Flow Direction

- [demFlow](#) - Downslope flow direction for a DEM
- [facetFlow](#) - Facet flow direction
- [flowMatrix](#) - Linear equations representing water flow
- [pixelFlow](#) - Downslope flow direction for DEM pixels

Preprocessing and Postprocessing

- [borderNans](#) - Find NaNs connected to DEM border
- [fillSinks](#) - Fill interior sinks in a DEM
- [postprocessPlateaus](#) - Replace upslope areas for plateaus with mean value

Hydrological Applications

- [dependenceMap](#) - Dependence map for water flow in a DEM
- [influenceMap](#) - Influence map for water flow in a DEM
- [upslopeArea](#) - Upslope area measurements for a DEM

Display

- [visDemFlow](#) - Visualize flow directions in a DEM
- [visMap](#) - Visualize influence or dependence map for a DEM

Data

- [milford_ma_dem.mat](#) - Sample DEM data provided by USGS and distributed via GeoWorld (www.geoworld.com), a USGS data distribution site. The data set is a 1000x1000 pixel digital elevation model (DEM) file for Milford, MA.

Making Your HTML Help Files Searchable

If you want the Help browser to include your documentation in its search results, provide a search database for your HTML help files. MATLAB can create a database for you with one command.

The example uses the `info.xml` file for the Upslope Area toolbox with the `help_location` specified as `C:\myfiles\upslope\html`.

To create the database files:

- 1 If you have not already done so, add the folder containing your `info.xml` file to the search path.

For the example, add the `C:\myfiles\upslope` folder to the path.

- 2 Create a searchable database by running

```
builddocsearchdb('full_path_to_help_location')
```

For the example, assuming your help files are in `C:\myfiles\upslope\html`, run:

```
builddocsearchdb ('C:\myfiles\upslope\html')
```

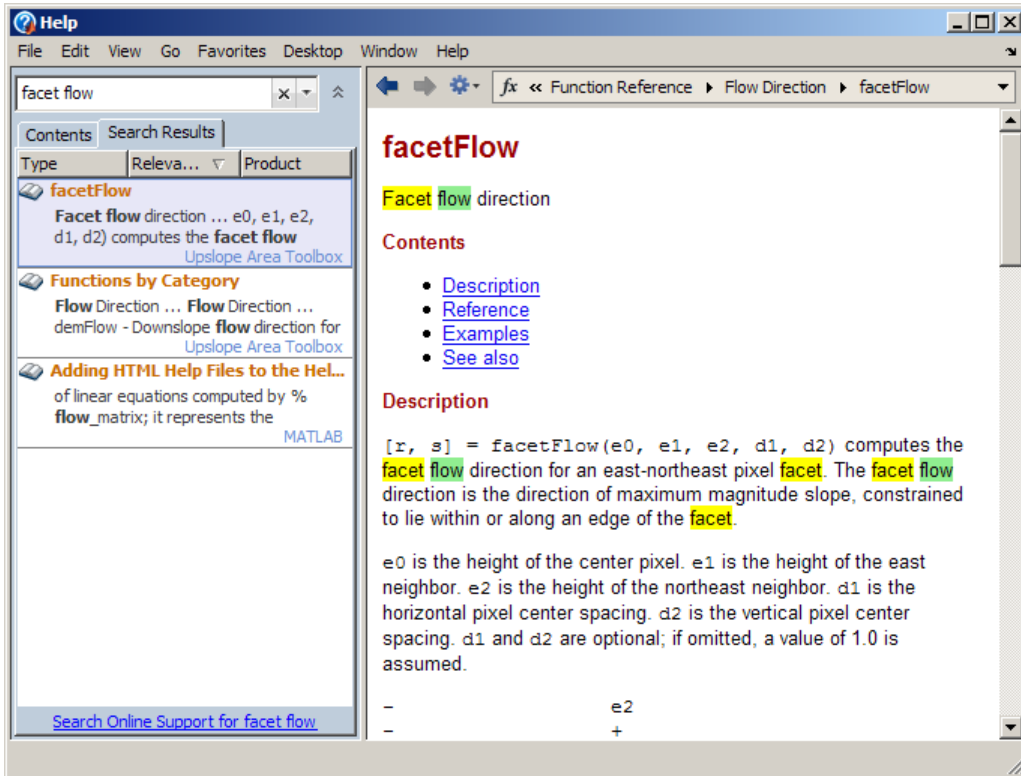
You must use the functional form when you call `builddocsearchdb` (with the folder location in single quotes inside parentheses).

`builddocsearchdb` creates a folder named `helpsearch` in the `help_location` folder. For the example, this command creates the folder `C:\myfiles\upslope\html\helpsearch`.

Each time you run it, `builddocsearchdb` generates three files in `helpsearch`:

- A file called `deletable`.
 - A file called `segments`.
 - A file having a `cfs` extension with a name that varies.
- 3 To verify that your help files can be searched, use the search field in the Help browser to search for any words in the HTML help files that you provided in the `help_location` folder.

The next figure shows a search of the Upslope Area toolbox and other documentation for the terms facet flow.



Summary of Workflow for Providing HTML Help Files

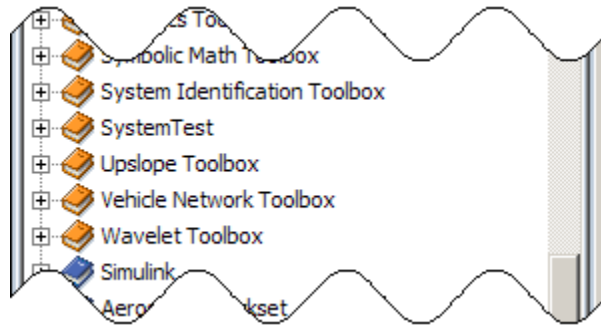
To include your HTML help files in the Help browser Contents pane, you must create and supply two XML files that the Help browser requires, plus HTML, and image files you develop for your documentation. You must also tell recipients of your software how to install these files. The following steps summarize the steps to take to add your documentation to the Help browser and distribute it to others.

This procedure uses template XML files that you need to customize. To see examples of content and how the files are organized, refer to a complete

example of user documentation, the Upslope Area toolbox. To view the Upslope documentation in the Help browser, click [here](#), or run

```
addpath(fullfile(matlabroot, 'help', 'techdoc', 'matlab_env', 'examples', 'upslope'))
```

The Upslope Area Toolbox now appears in the Contents pane (toolboxes are alphabetized), as the following graphic displays. The appearance of your contents pane depends on what products you have installed.



Tip Print or bookmark this page of instructions. Then, when you place your own HTML pages in the Help browser, you can view the instructions at the same time.

- 1 Create or choose a folder for storing your help files. You must have write access to the folder. You can use the same folder that contains your toolbox code.

For the Upslope Area toolbox example, name the folder `upslope`.

- 2 Add an `info.xml` file to the folder. This file identifies the folder as one that contains documentation. To add this file, either click [here](#) or follow these two steps:

- a Copy

```
copyfile(fullfile(matlabroot, 'help', 'techdoc', 'matlab_env',  
'examples', 'templates', 'info_template.xml'), pwd)
```

to the folder.

For example, copy the file to `upslope`.

- b** Verify that the copied `info_template.xml` file is writable. If it is read-only, make it writable with:

```
fileattrib('info_template.xml','+w')
```

- 3** Rename the copy of `info_template.xml` to `info.xml`. The file must have this name.
- 4** Within your current folder, create a new folder to contain files for the Help browser **Contents**. (The `info.xml` points to this folder as `<help_location>`.)

For the example, in `mytoolbox`, create a folder named `html`.

- 5** Make the new folder your current folder.

For the example, `cd html`.

- 6** Add a `helptoc.xml` file to the empty folder. This file organizes the Help browser table of contents for your toolbox. To add this file, either click [here](#) or follow these two steps:

- a** Copy the `helptoc_template.xml` file to your working directory:

```
copyfile(fullfile(matlabroot,'help','techdoc','matlab_env',  
'examples','templates','helptoc_template.xml'),pwd)
```

to the folder.

For example, copy the file to `mytoolbox/html`.

- b** Verify that the copied `helptoc_template.xml` file is writable. If it is read-only, make it writable with:

```
fileattrib('helptoc_template.xml','+w')
```

- 7** Rename the copy of `helptoc_template.xml` to `helptoc.xml`. It must have this name.

- 8 Move the HTML help files you created, as described in “Summary of Creating and Installing HTML Help Files” on page 5-19, to the folder containing your `helptoc.xml` file. Also move to the folder any files that the HTML files reference, such as image files.

The XML files in the `examples/templates` folder have no accompanying HTML files. You can, however, view files for the example Upslope Area toolbox and learn how its `helptoc.xml` file organizes HTML documentation content. For details, see “Organizing Your Documentation” on page 5-20

- 9 In the Editor, open, modify, and save your `info.xml`, `helptoc.xml` files.

For details about changes to make, see:

- “More About the `info.xml` File” on page 5-27
- “More About the `helptoc.xml` File” on page 5-32
- “Creating Function and Block Category Listings” on page 5-38

- 10 Verify that the Help browser **Filter by Product** preference is set so that your toolbox appears in the display. To set the **Filter by Product** preference:

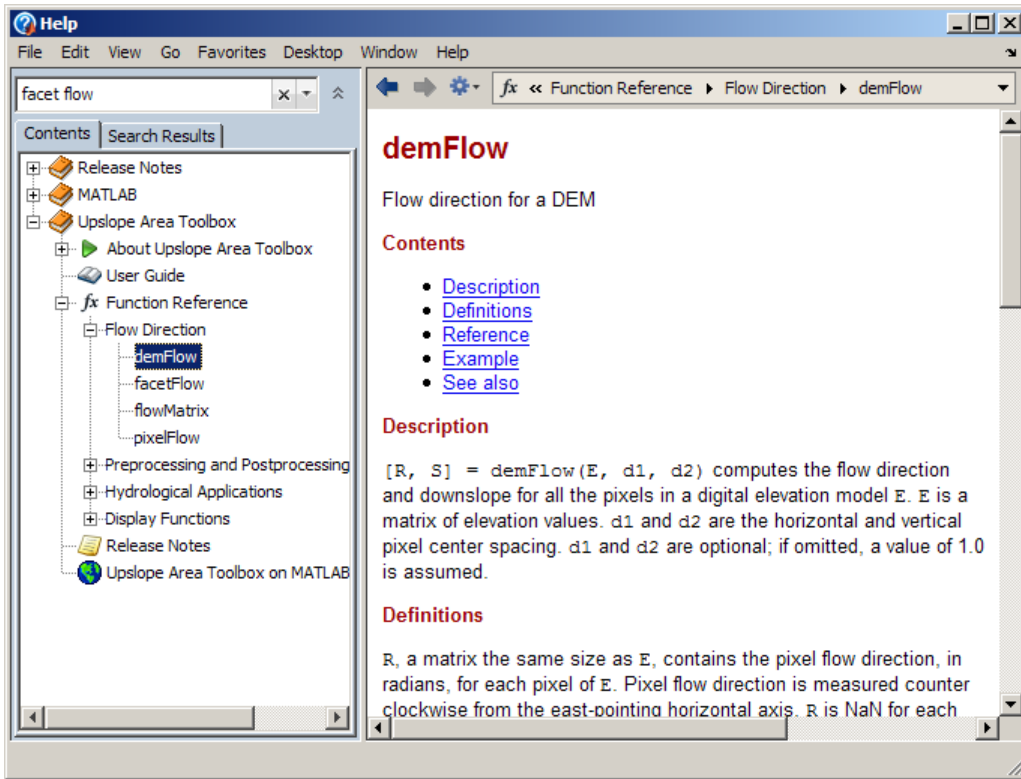
- a Access the **Help Preferences** pane by selecting **File > Preferences > Help**.

- b Under **Filter by Product**, select **All products**.

- 11 Add the folder to the search path.

For example, add `upslope` to the search path.

- 12 View your HTML help files in the Help browser **Contents** pane.



13 Review the browser display, and verify that there are no errors. MATLAB automatically validates `info.xml` files and reports any problems to the Command Window. For information about addressing the problems, see “Addressing Validation Errors for `info.xml` Files” on page 5-61.

14 If you provide your documentation to others, make sure that you include all files and folders:

- HTML files.
- Images or other files referenced by the HTML files.
- Your `info.xml` and `helptoc.xml` files.
- Your search database files, if any.

You can use `zip` or `gzip` to create an archive of the folders.

- 15** Instruct recipients of your documentation how to display it. They need to:
 - a** Unzip the archive containing the help files to any disk location they prefer to use, and add the help folder to the search path.
 - b** Verify that your toolbox is selected in the **Filter by Product** Help preferences. Selecting it enables your toolbox to appear in the **Contents** pane of the Help browser.
 - c** If your toolbox still does not appear in the **Contents** pane, suggest removing its folder from the search path and then adding back to the path. The toolbox folder cannot be the current folder during this operation.
- 16** Inform your users which documentation features you support. For example:
 - If you provided search database files, mention that Help browser search results will include your documentation.
 - Alternatively, you can instruct the users to generate a search database with the `builddocsearchdb` function after they set up your files.

Adding Demos to the Help Browser

In this section...
“About Creating Demos” on page 5-52
“Providing Demos to Others” on page 5-60

About Creating Demos

You can provide demos for toolboxes you create and make them available in the Help browser. Demos allow you to present the features of your toolbox. Adding your demos to the Help browser is the best way to make them accessible.

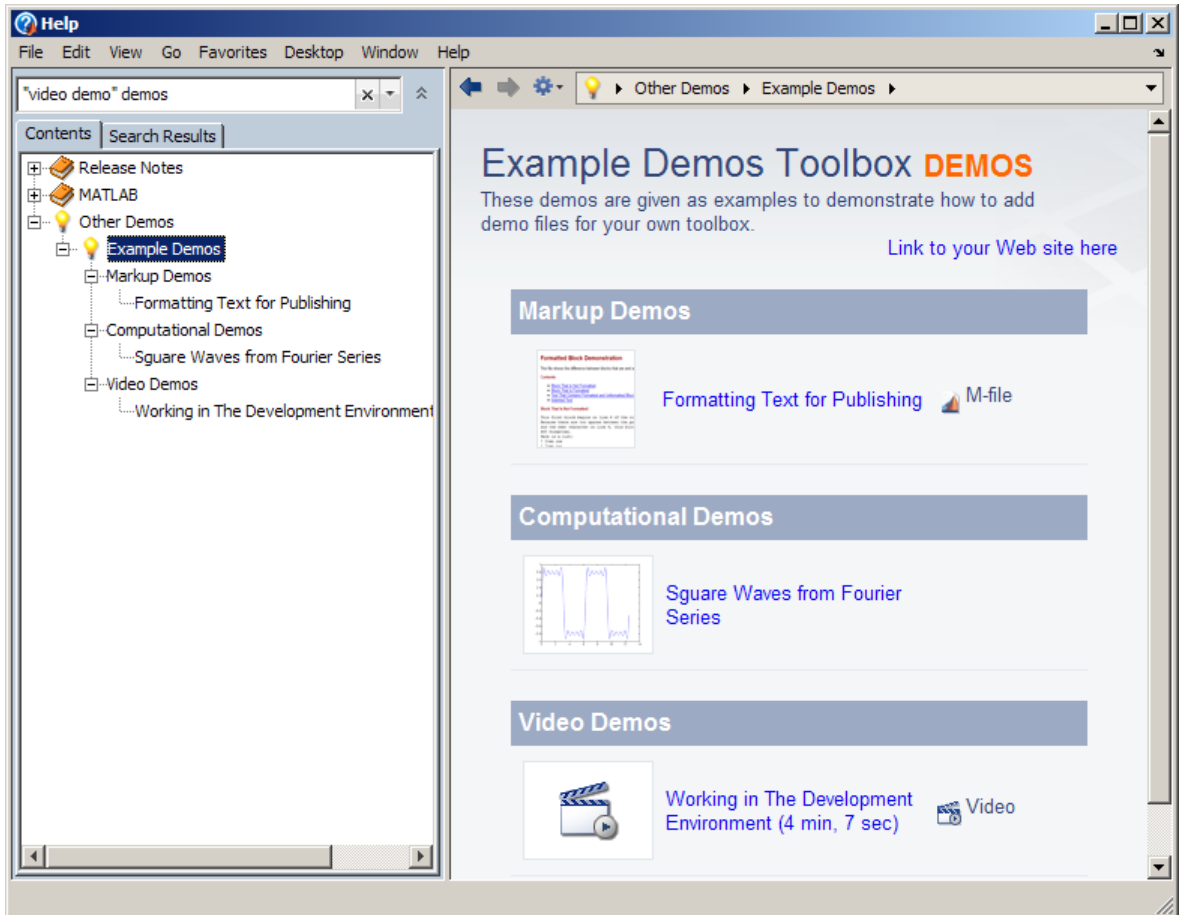
There are no requirements about the types of demos you can provide. However, if you provide the same types of demos that MathWorks products provide, users of your software are already familiar with using them. See “Types of Demos” on page 4-25.

This documentation includes an example folder which contains two demos and refers to a third one that comes with MATLAB. [Click here](#) to add this example folder to your search path, or run the following command:

```
addpath(fullfile(matlabroot,'help','techdoc','matlab_env','examples','demo_examples'))
```

The **Contents** pane of the Help browser displays an entry called **Example Demos** under the **Other Demos** entry. As shown in the following figure, within that entry you see three demos:

- Formatting Text for Publishing
- Square Waves from Fourier Series
- A video demo from MATLAB, Working in the Development Environment



To remove the demos you just added, take their folder off the search path:

```
rmrpath(fullfile(matlabroot,'help','techdoc','matlab_env','examples','demo_examples'))
```

How to Add Demos

Tip Print or bookmark this page of instructions. Then, when you view your own HTML pages in the Help browser, you can view the instructions at the same time.

After you create a demo, you can access the demo from the Help browser after you perform the following steps:

- 1 Create demos for your toolbox. See “About Creating Demos” on page 5-52.

You can effectively produce MATLAB code demos using the cell-publishing features available in the Editor. Publishing creates an HTML file that includes code, can include figures, describes how to use your code, and enables you to execute the code from the Help browser. For more information, see “Overview of Publishing MATLAB Code” on page 11-2.

- 2 Add the demos files to the Help browser using a special XML file that you create. See “Workflow for Providing Demos” on page 5-54.
- 3 Provide the demo files, along with instructions for including these files in the Help browser. See “Providing Demos to Others” on page 5-60.

The sections that follow refer to a folder of demo examples provided with this documentation. Adapt the contents of that folder to set up your own demos.

Workflow for Providing Demos

To include demos for your toolbox in the Help browser **Contents** pane, you must create and provide a `demos.xml` file and content for your demos. Specify a location to MATLAB where the files will reside:

- 1 Create or choose a folder for storing your demos files. You must have write access to the folder. If you have created a toolbox, the toolbox folder is a good location for storing related demos.

For the example, the folder is `/demo_examples`.

- 2 Create your demo files by publishing code files, constructing a GUI, or another method.

- 3 Put all the demos files you created in the folder.
- 4 Add the folder for your demos files to the search path.

Note The folder cannot be the current folder when you add it to the path or the Help browser will be unable to locate your demos.

- 5 Get the example `demos.xml` file to use as a template for your own file. Click [here](#) to copy that file to your current folder, or
 - a Copy
`matlabroot/help/techdoc/matlab_env/examples/demo_examples/demos.xml`
to the folder for your demos files:

```
copyfile(fullfile(matlabroot,'help','techdoc','matlab_env','examples','demo_examples','demos.xml')
```

- b Verify that the copied `demos.xml` file is writable. If it is read-only, make in writable with:

```
fileattrib(demos.xml','+w')
```

- 6 Edit the content of your copy of `demo.xml`, changing it to describe and point to your demo files. For details, see “More About the `demos.xml` File” on page 5-56.
- 7 View your `demos.xml` file in the Help browser. A new node, `Other Demos`, appears at the bottom of the Help browser **Contents** pane. Expand the node to view the entries you added.
- 8 If the `Other Demos` entry does not appear at the bottom of the Help Navigator, refresh the Help browser. You can refresh in two ways:
 - Right-click on `Demos` in the **Contents** pane, and select **Refresh Demos**. Doing so refreshes all demos on the search path and can take a moment.
 - Remove the folder containing your `demos.xml` file from the search path using `rmpath`. Then, use `addpath` to add your demos folder back on the search path.

Note The Help browser **Filter by Product** preference does not provide an **Other Demos** entry or list the toolbox demos you add. However, the Help browser always shows toolbox demos that you add to the search path.

More About the demos.xml File

Within the `demos.xml` file, the root tag is `<demos>`. This element includes one `<name>`, `<type>`, `<icon>` and `<description>` for the main demo page for your toolbox.

Include a `<demoitem>` for each demo you add. Provide multiple categories of demos by including a `<demosection>` for each category. Put `<demoitem>` entries within that category. If you include *any* categories, then *all* demos must be in categories. In other words, if there is even one `<demosection>`, then all `<demoitem>` tags must be within `<demosection>` tags.

Step 5 of the previous procedure tells how to obtain the example `demos.xml` file. This example contains the following XML code:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Example demos.xml file for adding demos to the Help browser -->
<!-- Your version of this file must be named "demos.xml" -->
<demos>
  <!-- Top-level Demo title in TOC -->
  <name>Example Demos</name>
  <type>toolbox</type>
  <icon>HelpIcon.DEMOS</icon>
  <description>These demos are given as examples to
  demonstrate how to add demo files for your own toolbox.
  </description>
  <website>
  <a href="http://www.mathworks.com">Link to your Web site here</a>
  </website>
  <!-- First group of demos begins here -->
  <demosection>
    <label>Markup Demos</label>
    <!-- First demo begins here -->
    <demoitem>
```

```

        <!-- How demo is described in the Contents pane -->
        <label>Formatting Text for Publishing</label>
        <!-- Type adds a system icon and this name next to demo item -->
        <type>M-file</type>
        <!-- File to display in the Viewing pane -->
        <file>./html/formatted_block_demo.html</file>
        <!-- Supply optional thumbnail for demo as a .png file -->
        <!-- Name it <demo_name>.png -->
        <!-- for this demo it is ./html/formatted_block_demo.png -->
    </demoitem>
</demosection>
<!-- Second group of demos begins here -->
<demosection>
    <label>Computational Demos</label>
    <demoitem>
        <!-- How demo is described in the Contents pane -->
        <label>Square Waves from Fourier Series</label>
        <!-- Do not add a <type> element if demo is executable -->
        <!-- File to execute for "Run this demo" -->
        <callback>fourier_demo</callback>
        <!-- File to display in the Viewing pane -->
        <file>./html/fourier_demo2.html</file>
    </demoitem>
</demosection>
<!-- Third group of demos begins here -->
<demosection>
    <label>Video Demos</label>
    <demoitem>
        <!-- Type adds a system icon and this name next to demo item -->
        <type>video</type>
        <!-- How demo is described in the Contents pane -->
        <!-- This is an actual MATLAB Flash video demo -->
        <!-- If Flash is installed, it runs in your system browser -->
        <label>Working in The Development Environment (4 min, 7 sec)</label>
        <!-- Command or file to execute for "Run this demo" -->
        <callback>
            playbackdemo('WorkingInTheDevelopmentEnvironment',...
                'toolbox/matlab/web/demos');
        </callback>
    </demoitem>

```

```

        </demosession>
    </demos>

```

Lines starting with `<!--` and ending with `-->` are comments. The code contains two `<demosession>` items, each containing one `<demositem>`. The first demo consists of HTML documentation only (a `<file>` element). The second one has both HTML documentation and MATLAB code that the reader can execute (a `<callback>` element). When you include a `callback` element, it must contain an executable command. The reader can execute that command by clicking **Run this demo** at the top of the demo page.

The next table describes the `demos.xml` file listed above, and found in the folder `matlabroot/help/techdoc/matlab_env/examples/demo_examples`.

Line	XML Tag	Notes	Value for Example
4	<code><demos></code>	The root element for a <code>demos.xml</code> file.	No value
6	<code><name></code>	Name of your toolbox or collection of demos that displays under Other Demos in the Help browser.	Example Demos
7	<code><type></code>	The product type. Allowable values are <code>matlab</code> , <code>simulink</code> , <code>toolbox</code> , <code>blockset</code> , <code>links_targets</code> , <code>M-file</code> , <code>video</code> , or <code>other</code> .	toolbox
8	<code><icon></code>	Icon for your demo. You can use a standard icon or provide a custom icon by specifying a path to the icon relative to the location of the <code>demos.xml</code> file.	HelpIcon.DEMOS
9 to 11	<code><description></code>	The description that appears in the Help browser viewing pane, on the main page for your demos.	Suggested text: “These demos are given as examples to demonstrate how to add demo files for your own toolbox.”

Line	XML Tag	Notes	Value for Example
12 to 14	<code><website> </website></code>	(Optional) Link to a Web site. For example, MathWorks demos include a link near the top, on the right: Product page at mathworks.com. Can appear anywhere before the <code></demos></code> tag.	<code> Link to your Web site here</code>
16	<code><demosession></code>	(Optional) Begins a category of demos. Each category includes a <code><label></code> , description, and at least one <code><demositem></code> . Use any number of categories.	No value required
17	<code><label></code>	Title shown in Help browser for a <code><demosession></code> .	"Markup Demos"
19	<code><demositem></code>	Use one <code><demositem></code> per demo. Contains <code><label></code> and either a <code><callback></code> or a <code><file></code> tag.	No value required
21	<code><label></code>	Title shown for <code>demositem</code> .	"Formatting Text for Publishing" (1st example <code>demositem</code>)
23	<code><file></code>	Name of HTML file describing the demo, typically produced by publish. Specify a relative path from the location of <code>demos.xml</code> .	<code>./html/formatted_block_demo.html</code> (1st example <code>demositem</code>)
33	<code><callback></code>	Name of an executable file or a MATLAB command. This file runs when you click Run this demo on the demo page.	<code>./html/fourier_demo2.html</code> (for 2nd <code>demositem</code> example)
None	<code><dependency></code>	(Optional) Specifies other products required to run the demo, such as another toolbox. The text must match a product name specified in an <code>info.xml</code> file that is on the search path or in the current folder.	Not included

Supplying Thumbnail Images for Demos. If your demo has an HTML file to describe it, you can include a *thumbnail*, a small image typifying the demo. The `demos.xml` file does not specify thumbnail images directly.

To include a thumbnail, you only need to supply a `.png` image file in the same folder as the HTML file for the demo. Keep the image size to within 96-by-64 pixels (width-by-height). Give the `.png` file the same name as the HTML file. Thus, if the `<file>` element for your demo is `./html/formatted_block_demo.html`, then your thumbnail must be named `formatted_block_demo.png` and reside in the same folder.

When you publish a MATLAB script to HTML with the `publish` command or Editor **File** menu item, you get a `.png` thumbnail file in the correct place with the correct name by default.

Providing Demos to Others

Anyone who wants to use your demos needs the files and instructions for using them:

- 1 Provide recipients with a folder containing:
 - Your demo files
 - All data, images or other files referenced by the demo files
 - Your `demos.xml` file
- 2 Instruct recipients to add the folder containing the demos files to the search path.
- 3 Inform recipients that the toolbox demos appear under **Other Demos**, the last entry in the **Contents** pane.

Addressing Validation Errors for info.xml Files

In this section...

“About XML File Validation” on page 5-61

“Entities Missing or Out of Order in info.xml” on page 5-61

“Unrelated info.xml File” on page 5-62

“Invalid Constructs in info.xml File” on page 5-62

“Outdated info.xml File for a MathWorks Product” on page 5-62

About XML File Validation

When MATLAB finds an `info.xml` file on the search path or in the current folder, it tries to add information to the Help browser or **Start** button, as specified in the `info.xml` file. MATLAB automatically validates the file against the supported schema. If there is an invalid construct in the `info.xml` file, MATLAB displays an error in the Command Window. The error is typically of the form:

```
Warning: File <yourxmlfile.xml> did not validate.
```

```
...
```

An `info.xml` validation error can occur when you start MATLAB, press the **Start** button, or add folders to the search path.

Following, are the primary causes of an XML file validation error and information to address them:

Entities Missing or Out of Order in info.xml

If you do not list required XML elements in the prescribed order, you receive an XML validation error:

```
Often, errors result from incorrect ordering of XML tags. Correct the error by updating the info.xml file contents to follow the guidelines in the MATLAB help documentation.
```

The message contains a hyperlink to the page you are now reading. For a description of the elements you need in an `info.xml` file and their required ordering, see “More About the `info.xml` File” on page 5-27.

Unrelated info.xml File

Suppose you have a file named `info.xml` that has nothing to do with the MATLAB Help browser or **Start** button. Because this `info.xml` file is an unrelated file, if the file causes an error, the validation error is irrelevant. In this case, the error is not actually causing any problems, so you can safely ignore it. To prevent the error message from reoccurring, rename the offending `info.xml` file, or ensure that the file is not on the search path or in the current folder.

Invalid Constructs in info.xml File

If the purpose of the `info.xml` file is to add information to the **Start** button or Help browser, correct the reported problem. Use the message in the error to isolate the problem or use any validator. One validator you can use is from the W3C® at <http://www.w3.org/2001/03/webdata/xsv>. For more information about the structure of the `info.xml` file, consult its schema, located at `matlabroot/sys/namespace/info/v1/info.xsd`.

Outdated info.xml File for a MathWorks Product

If you have an `info.xml` file from a different version of MATLAB, that file could contain constructs that are not valid with your version. To identify an `info.xml` file from another version, look at the full path names reported in the error message. The path usually includes a version number, for example, `... \MATLAB\R14\...`. In this situation, the error is not actually causing any problems, so you can safely ignore the error message. To ensure that the error does not reoccur, remove the offending `info.xml` file, or ensure that the file is not on the search path or in the current folder.

Workspace Browser and Variable Editor

- “MATLAB Workspace” on page 6-2
- “Viewing and Editing Workspace Variables with the Variable Editor” on page 6-24

MATLAB Workspace

In this section...

“About the Workspace” on page 6-2

“Opening the Workspace Browser” on page 6-2

“Viewing and Editing Values in the Current Workspace” on page 6-4

“Saving the Current Workspace” on page 6-5

“Viewing and Loading a Saved Workspace and Importing Data” on page 6-7

“Changing and Copying Variable Names” on page 6-8

“Deleting Workspace Variables” on page 6-9

“Viewing Base and Function Workspaces Using the Stack” on page 6-9

“Creating Plots from the Workspace Browser” on page 6-10

“Opening Variables and Objects for Viewing and Editing” on page 6-21

“Setting Workspace Browser Preferences” on page 6-21

About the Workspace

The workspace consists of the set of variables built up during a session of using the MATLAB software and stored in memory. You add variables to the workspace by using functions, running M-files, and loading saved workspaces. For example, if you run these statements,

```
A = magic(4)
R = randn(3,4,5)
```

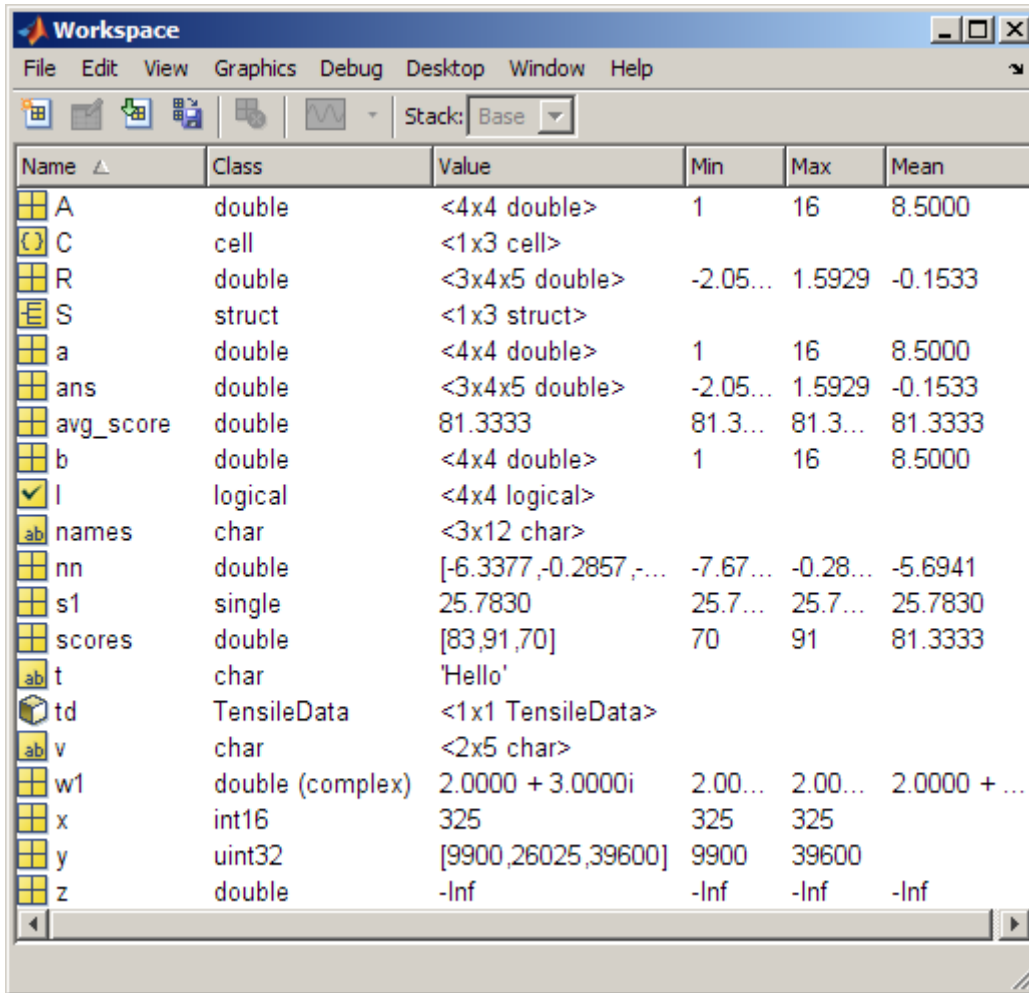
the workspace includes two variables, A and R.

You can perform workspace and related operations using the Workspace browser. When available, equivalent functions are documented with each feature of the Workspace browser.

Opening the Workspace Browser

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

The Workspace browser opens.



You can specify which buttons you want to appear on the toolbar using preferences. Select **File > Preferences > Toolbars** to open the dialog box. Click **Help** for information using the dialog box.

Viewing and Editing Values in the Current Workspace

The Workspace browser shows the name of each variable or object, the class (also represented by the icon), its value, and where relevant, the **Min**, **Max**, and **Mean** calculations. MATLAB performs these calculations using the `min`, `max`, and `mean` functions, and updates the results automatically. These are other features of the Workspace browser:

- You can display additional columns, including size (dimensions), size in bytes, and other common statistical calculations such as mode and standard deviation. To show or hide columns, select **View > Choose Columns**. On Microsoft Windows systems, you can right-click any column header to hide it or to show or hide other columns. To specify the size limit for calculations and how NaNs are considered, use “Setting Workspace Browser Preferences” on page 6-21.
- To resize a column of information, drag the column header border. To reorder columns, drag a column header to a new position.
- You can select the column on which to sort as well as reverse the sort order of any column. Click a column header to sort on that column. Click the column header again to reverse the sort order in that column. For example, to sort on **Name**, click the column header once. To change from ascending to descending, click the header again. You cannot sort by the **Value** column in the Workspace browser.
- You can directly edit variable values in the Workspace browser **Value** column. To edit a value, position the pointer in the **Value** column at the row you want to edit, click, and type the new value.
- To view more of the data for a variable, as well as to more easily edit it, double-click a variable name and it opens in the Variable Editor. For more information, see “Viewing and Editing Workspace Variables with the Variable Editor” on page 6-24.

Function Alternative

Use `who` to list the current workspace variables. Use `whos` to list the variables and information about size and class. For example:

```
>> who
```

```
Your variables are:
```



```

A      S      avg_score  names      scores      v      y
C      a      b          nn          t          w1     z
R      ans     1          s1         td         x

```

```
>> whos
```

Name	Size	Bytes	Class	Attributes
A	4x4	128	double	
C	1x3	348	cell	
R	3x4x5	480	double	
S	1x3	826	struct	
a	4x4	128	double	
ans	3x4x5	480	double	
avg_score	1x1	8	double	
b	4x4	128	double	
l	4x4	16	logical	
names	3x12	72	char	
nn	3x3	72	double	
s1	1x1	4	single	
scores	1x3	24	double	
t	1x5	10	char	
td	1x1	152	TensileData	
v	2x5	20	char	
w1	1x1	16	double	complex
x	1x1	2	int16	
y	1x3	12	uint32	
z	1x1	8	double	

Use `exist` to see if the specified variable is in the workspace.


Saving the Current Workspace

The workspace is not maintained across sessions of MATLAB. When you quit MATLAB, the workspace is cleared. You can save any or all of the variables in the current workspace to a MAT-file, which is a binary file specifically for use in MATLAB. You can then load the MAT-file at a later time during the current or another session to reuse the workspace variables. MAT-files use a `.mat` extension.

Note The .mat extension is also used by Microsoft Access application. You can change the default file association in the Microsoft Windows operating system to associate MAT-files with either MATLAB or the Access application.

Saving All Variables

To save all of the workspace variables using the Workspace browser:

- 1 Select **File > Save Workspace As** from the Workspace browser, or click the Save button  in the Workspace browser toolbar.

The Save to MAT-File dialog box opens.

- 2 Specify the location and **File name**. MATLAB automatically supplies the .mat extension.
- 3 Click **Save**.

The workspace variables are saved under the MAT-file name you specified.

Saving Selected Variables

To save some but not all of the current workspace variables:

- 1 Select the variable in the Workspace browser. To select multiple variables, **Shift**+click or **Ctrl**+click.
- 2 Right-click, and from the context menu, select **Save As**.

The Save to MAT-File dialog box opens.

- 3 Specify the location and **File name**. MATLAB automatically supplies the .mat extension.
- 4 Click **Save**.

The workspace variables are saved under the MAT-file name you specified.

Another way to save selected variables from the Workspace browser to a MAT-file is by dragging the selected variables from the Workspace browser to the Current Folder browser. For more information, see “Creating and Updating MAT-Files with the Current Folder Browser” on page 7-37.

Specifying the Format When Saving MAT-Files

To specify preferences for saving MAT-files that pertain to compression, and compatibility between different versions of MATLAB, see “MAT-Files Preferences” on page 2-131.

Function Alternative

To save workspace variables, use the `save` function followed by the filename you want to save to. For example,

```
save('june10')
```

saves all current workspace variables to the file `june10.mat`.

If you don't specify a filename, the workspace is saved to `matlab.mat` in the current folder. You can specify which variables to save, as well as control the format in which the data is stored, such as ASCII. For these and other forms of the function, see the reference page for `save`. MATLAB provides additional functions for saving information — see “Exporting Data”.

Viewing and Loading a Saved Workspace and Importing Data

You can view saved variables and load the variables and other data into the workspace using the Current Folder browser, the Import Wizard, or the `load` function.


Viewing Variables in MAT-Files and Loading Them into the Workspace

Use the Current Folder browser to view the contents of a MAT-file without loading the file into MATLAB. You can also load selected variables by dragging them from a MAT-file in the Current Folder browser to the Workspace browser. For details, see “Using the Current Folder Browser” on page 7-18.

Function Alternative. Use `whos` with the `-file` option.

Importing Data

MATLAB provides an Import Wizard to load MAT-files and other forms of data into the workspace. This procedure briefly describes how to use the Import Wizard from the Workspace browser to import data from MAT-files.

- 1 Click the Import Data button  on the toolbar in the Workspace browser.

The Import Data dialog box opens.

- 2 Select the MAT-file you want to load and click **Open**.

The variables and their values, as stored in the MAT-file, are loaded into the current workspace. Any variables in the MAT-file that are not in the workspace are added to the workspace. If any variables being loaded have the same names as variables in the current workspace, MATLAB asks if you want to replace the values in the current workspace with the values in the MAT-file, or cancel.

You can also use the Workspace browser to import data you previously copied to the clipboard by selecting **Edit > Paste to workspace**, or use **Ctrl+V**. This imports the clipboard data using the Import Wizard.

For more information about the Import Wizard, see “Tips for Using the Import Wizard”.

Function Alternative for Loading Variables

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

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

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


Changing and Copying Variable Names

To rename a variable in the workspace, right-click the variable in the Workspace browser and select **Rename** from the context menu. Type the new variable name over the existing name and press **Enter**.

To copy variable names to the clipboard, select the workspace variables and select **Edit > Copy**. You can then paste the names, for example, into the Command Window. Multiple variables are comma separated.

Deleting Workspace Variables

You can delete a variable, which removes it from the workspace:

- 1 In the Workspace browser, select the variable, or **Shift**+click or **Ctrl**+click to select multiple variables. To select all variables, choose **Select All** from the **Edit** or context menus.
- 2 Press the **Delete** key on your keyboard or click the Delete button  on the Workspace browser toolbar.
- 3 A confirmation dialog box might appear. If it does, click **OK** to clear the variables.

The confirmation dialog box appears if you selected that preference. For more information, see “Confirmation Dialogs Preferences” on page 2-132.

To delete all variables, select **Edit > Clear Workspace** from any desktop tool.

Function Alternative

Use the `clear` function to clear variables from the workspace. For example,

```
clear A M
```

clears the variables A and M from the workspace.


Use the `clearvars` function with the `-except` option to keep the specified variables and clear all other variables.

Viewing Base and Function Workspaces Using the Stack

When you run M-files, MATLAB assigns each function its own workspace, called the function workspace, which is separate from the base workspace in MATLAB. To access the base and function workspaces when debugging

M-files, use the **Stack** field in the Workspace browser. The **Stack** field is only available in debug mode and otherwise appears dimmed. The **Stack** field is also accessible from the Editor and the Variable Editor. For more information, see “Finding Errors, Debugging, and Correcting MATLAB Files” on page 9-104. See also the `dbstack` and `evalin` functions.

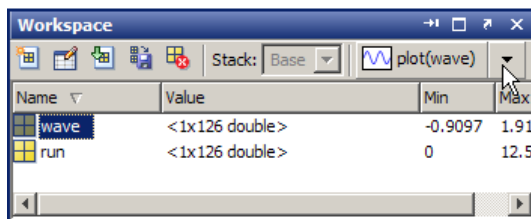
Creating Plots from the Workspace Browser

You can generate a plot of one or more variables with the *Plot Selector* tool  on the Workspace browser toolbar. (The appearance of the tool varies, depending on the variables you select and your history of using it.) The Plot Selector contains menu items identifying plotting functions available to you and executes them when you click the graph icons. The following steps illustrate how the tool works:

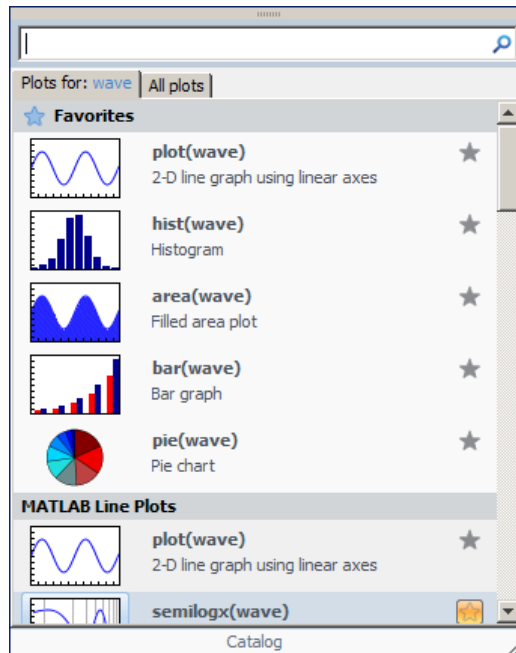
- 1 Create two vector variables:

```
run = 0:.1:4*pi;
wave = (sin(run.^2) + cos(run).^2);
```

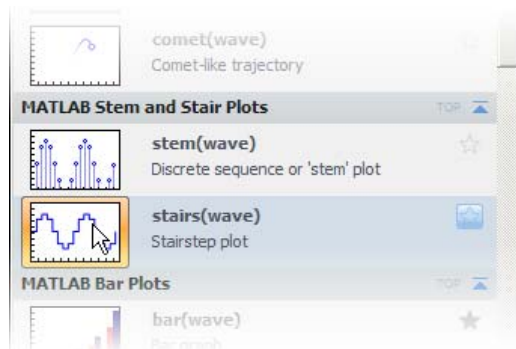
- 2 Select one or both variables in the Workspace Browser. You can **Shift**+click or **Ctrl**+click to select multiple variables to plot as *x*, *y*, or *z* components of a graph.



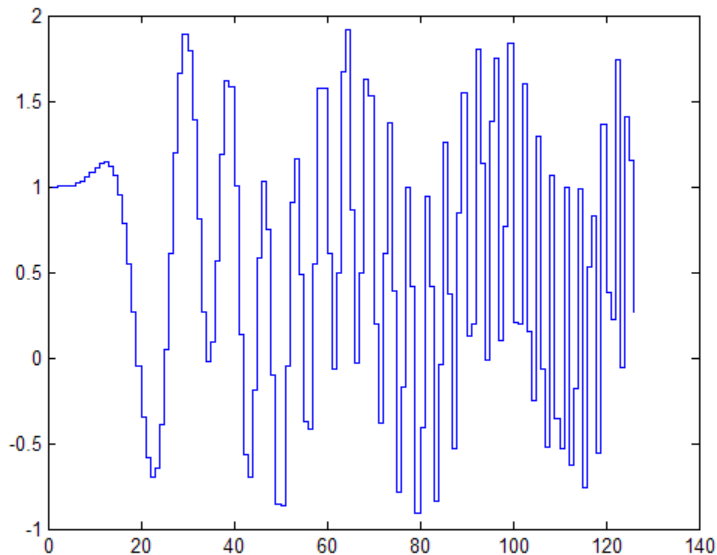
- 3 Click the down arrow icon in the Plot Selector on the toolbar. The Plot Selector menu opens.



- 4** Scroll to the graph type you want to display and click the icon on its left side. The icon highlights when you hover over it.



A figure window opens with a graph of the selected variable or variables, using the plotting function you just chose.


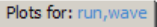





The Plot Selector GUI remains open until you click a desktop component or another window. This enables you to experiment with plot types without reopening the GUI each time.


Working with the Plot Selector GUI

You can interact with the Plot Selector in many ways. Learn about using it by viewing this video demo. Also, read the following table to better understand what you can accomplish with the Plot Selector and how to do it.

What You Can Do	How To Do It
Identify variables to graph.	Select a variable of numeric type in the Workspace Browser. To add variables to your selection, Shift +click or Ctrl +click.
Plot selected variables immediately using the default graph type.	Click the Plot Selector icon; a graph of the type it displays will plot using the selected variables.

What You Can Do	How To Do It
<p>Plot selected variables after choosing a graphing function.</p>	<p>Click the arrow ▼ to the right of the Plot Selector button. The GUI opens for you to choose a graph type. Create plot by clicking the chosen item's picture icon.</p>
<p>Interchange two variables before graphing them.</p>	<p>Click the Reverse Input Variable Order button . The two function arguments the item displays interchange. The button appears only when you select two variables.</p>
<p>Identify the graphing functions you can currently use to create a plot.</p>	<p>With one or more selected variables, select the first tab, Plots for:<variables>  and do either of the following:</p> <ul style="list-style-type: none"> • Scroll through the menu or navigate through it with up/down arrow keys. • Enter a search term in the search bar; only those menu items and categories containing the search string will display.
<p>Identify available graphing functions.</p>	<p>With zero or more variables selected, select the second tab, All plots  and do either of the following:</p> <ul style="list-style-type: none"> • Scroll through the menu or navigate through it with up/down arrow keys • Enter a search term in the search bar; only those menu items and categories containing the search string display

What You Can Do	How To Do It
<p>Add a graph type to Favorites.</p>	<p>In either tab:</p> <ul style="list-style-type: none"> • Click the star  to the right of a menu item and do any of the following: • Right-click a menu item and select Add to Favorites. • Right-click a menu item and select Copy, and then right-click within Favorites and select Paste. • Drag a menu item from its category to Favorites. <p>All of these actions create a copy the item in Favorites without removing it from its category.</p>
<p>Remove a graph type from Favorites.</p>	<p>In the Favorites category of either tab do either of the following:</p> <ul style="list-style-type: none"> • Click the star  at the right side of a menu item • Right-click a menu item and select Remove Favorite <p>You cannot drag a menu item out of Favorites.</p>
<p>Change the order of menu items within a category.</p>	<p>Click a menu item outside the graph icon and drag it to a new position within its category.</p>
<p>Move a menu item to a different category.</p>	<p>Click a menu item outside the graph icon and drag it to a new category. Dragging an item away removes that item from the category it was in. This gesture does not apply to items within Favorites.</p>

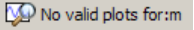
What You Can Do	How To Do It
Get help for a graphing function.	On either tab, hover mouse pointer over a menu item. A syntax description pops up next to the item. To see the complete function reference page, click the <i>More Help</i> link in the description header.
Enable a dimmed menu item.	Select one or more variables appropriate for calling the dimmed function. Then, open the Plot Selector tool again. See “Selecting Appropriate Variables” on page 6-15 and “Determining What Inputs a Graphing Function Needs” on page 6-16.
Manually execute or modify a plotting call.	Drag a menu item (even if it is dimmed) to the Command Window or the Editor; the code for that plot displays and you can edit it. See “Editing a Plot Selector Graphing Command” on page 6-17.
Display the Plot Selector as a window.	Open the Plot Selector, click the grab bar  at the center top, and drag the GUI to where you want it on the screen. That window closes if you click another MATLAB window. The window remains open if you focus on a window of another application.
Access the Plot Catalog.	Click the Catalog link at the bottom of the Plot Selector tool.

Selecting Appropriate Variables

If you select insufficient or inappropriate variables for a graph type, its menu item is dimmed on the **All plots** Plot Selector tab. The dimmed items do not appear at all on the **Plots for:** tab. Dimmed and missing Plot Selector menu items indicate that you cannot use the function for plotting the selected data via the tool. Reasons why a function is not available include:

- You selected a variable having the wrong class for graphing (it is not a numeric type).
- The graphing function requires additional inputs (for example, scatter requires two vectors).

- The graphing function requires fewer inputs than you selected (for example, `plot` cannot handle three vectors)
- Your selected variable has inappropriate type or dimensions (for example, matrices displayed by `feather` must be complex).
- You selected variables in the wrong order (for example, selecting scalar `n` and then matrix `Z` to `contour(n,Z)` instead of `contour(Z,n)`).

If you select variables that no function can display, the Plot Selector button label changes to **No valid plots for:<variable>** . If you click the button, you see the message “No plot available for selection. Change your variable selection or click the **All plots** tab to browse for plots.”

Determining What Inputs a Graphing Function Needs

When a Plot Selector menu item is unavailable (dimmed) in the **All plots** tab, you can learn why by viewing the pop-up help for that function. Suppose you want to make a **semilogy** graph.

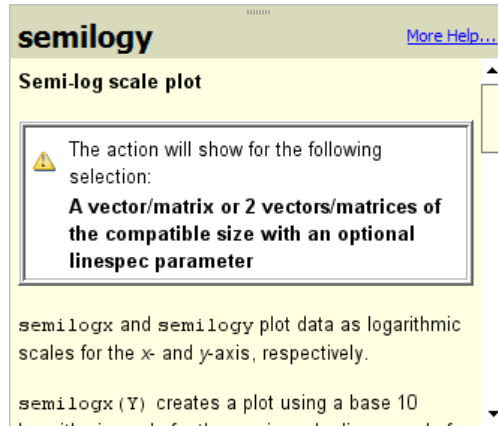
In the **All plots** tab, you can do several different things:

- 1 Create a nonnumeric variable, for example, a string.

```
name = 'abcd';
```

- 2 Select `name` in the Workspace Browser and open the Plot Selector tool by clicking its down arrow.
- 3 Click the **All plots** tab in the Plot Selector window.
- 4 Scroll to the **semilogy** menu item (all items are dimmed) and hover the mouse pointer over the item.

A help message appears that includes a note specifying what inputs the **semilogy** function accepts. The pop-up help window contains a white box that describes the function’s input requirements, as shown in the following figure.



The message helps you to understand what arguments the `semilogy` function supports.

A Help Window opens whenever your mouse pointer lingers over a menu item, but information set off in a white box only appears when the function is dimmed because it cannot generate a graph of the currently selected variables.

Editing a Plot Selector Graphing Command

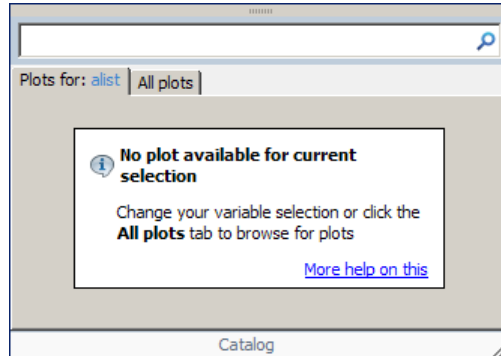
Even when the Plot Selector tool cannot successfully make a graph, it can still give you a starting point for doing so. You can drag any Plot Selector menu item into the Command Window to see the code it generates. Once the plotting command displays in the Command Window, it does not execute until you press **Enter**, enabling you to edit its arguments first. Dragging dimmed items works the same as dragging undimmed items. This enables you to fix problems due to selecting the wrong variables or selecting them in the wrong sequence. You can also add arguments to the plotting command, for example to specify a `linespec` or a `colourspec` for functions that can use them.

The following example illustrates how to drag a dimmed Plot Selector menu item, drop it into the Command Window, and edit the plotting command before executing it.

1 In the Command Window, enter

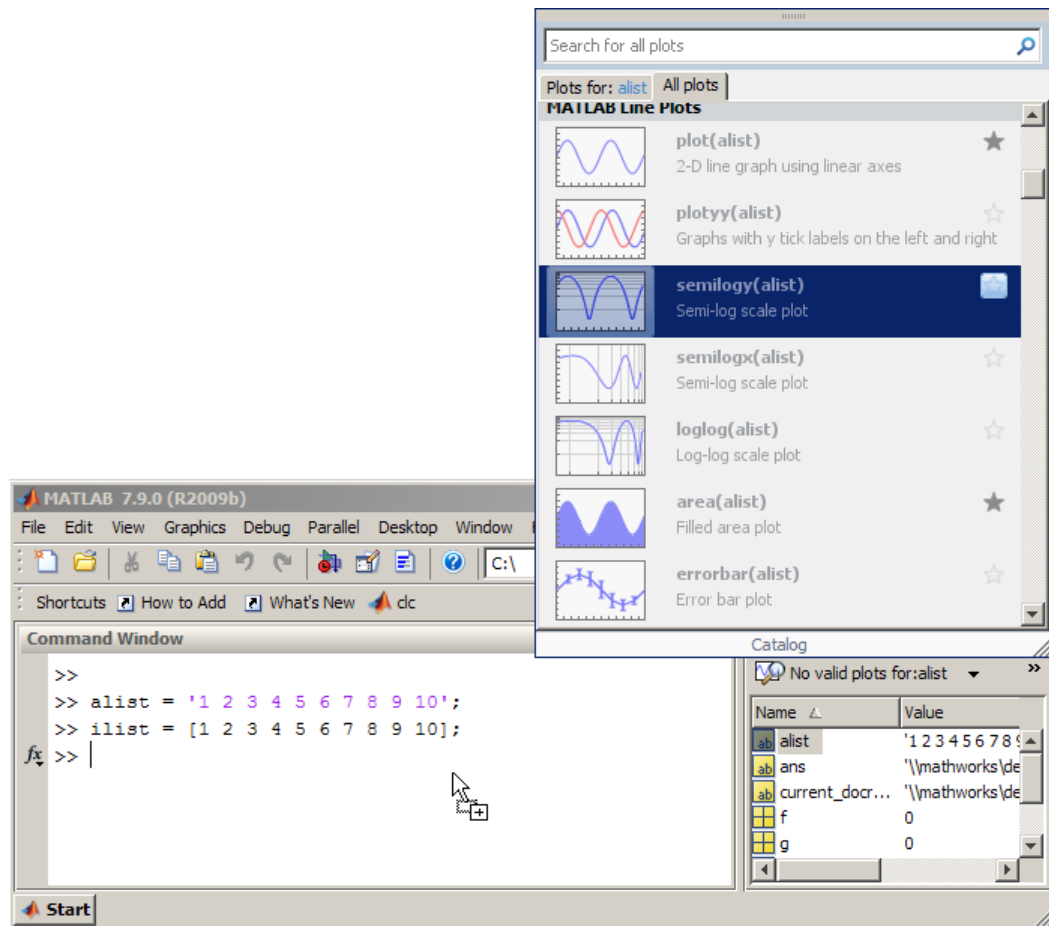
```
alist = '1 2 3 4 5 6 7 8 9 10';
ilist = [1 2 3 4 5 6 7 8 9 10];
```

- 2 Assume that you want to create a graph of `ilist` but instead you select the string variable `alist` in the Workspace Browser. When you click the Plot Selector, it displays this error message.



You can go on to plot `ilist` by continuing as follows.

- 3 Click the **All Plots** tab and scroll to `semilogy`. The menu item is dimmed.
- 4 Press the mouse button over `semilogy` and drag it to the command window, as shown here.



- 5 Release the mouse button in the Command Window. The following line of code appears there:

```
>> semilogy(alist,'DisplayName','alist');figure(gcf)
```

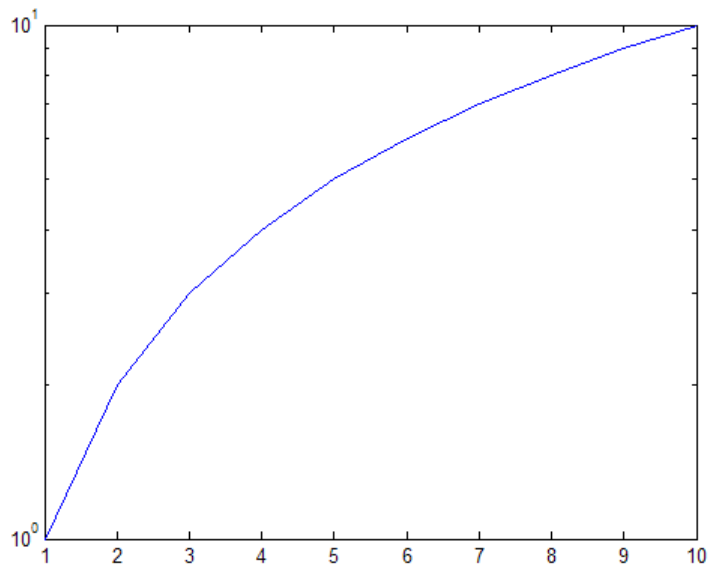
If you press **Enter**, an error displays because `alist` is not a proper calling argument:

```
??? Error using ==> semilogy
Invalid first data argument
```

6 Instead, change the code to the following, substituting `ilist` for `alist`:

```
semilogy(ilist, 'DisplayName', 'ilist');figure(gcf)
```

This call works with the changed arguments and generates the following graph after you press **Enter**.



You can also edit the plotting command to add other arguments, including parameter/value pairs, to customize the graph.

For More Information

If you want to use the Plot Selector to create a graph for a subrange of a vector or a matrix, you can open the variable in the Variable Editor, select the range of data you want to graph, and open the Plot Selector from the Variable Editor toolbar. All data for the graph must all come from one variable and the subrange to plot has to be a contiguous selection.

You can add titles, axis labels, legends and other annotations to graphs that the Plot Selector makes. For more information about graphing data and customizing plots, see the following topics in the *MATLAB Graphics* documentation:

- “Figures, Plots, and Graphs”
- “Plotting Tools — Interactive Plotting”
- “Basic Plotting Commands”
- “Creating Specialized Plots”

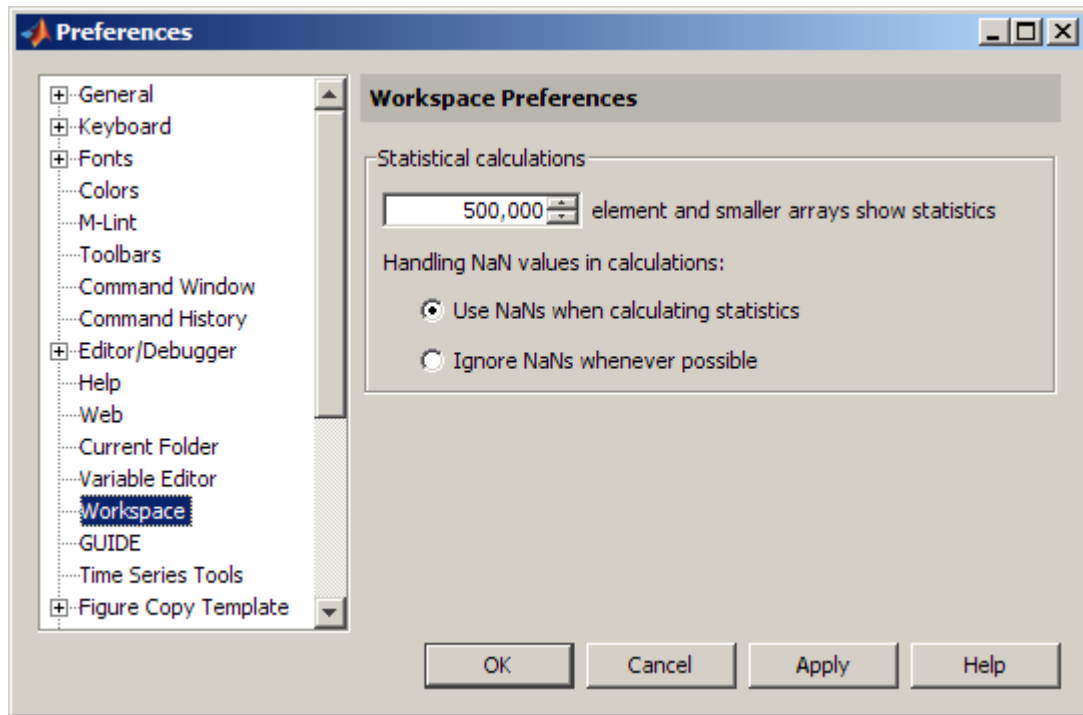
Opening Variables and Objects for Viewing and Editing

In the Workspace browser, double-click a variable and it opens in the Variable Editor where you can view and edit the contents of the variable. See “Viewing and Editing Workspace Variables with the Variable Editor” on page 6-24 for more information.

Some toolboxes allow you to double-click an object in the Workspace browser to open a viewer or other tool appropriate for that object. For details, see the toolbox documentation for that object type.

Setting Workspace Browser Preferences

The Workspace browser displays statistical calculations for variables. Use preferences to restrict the size of arrays on which you perform calculations and to specify if you want those calculations to include or ignore NaNs. Select **File > Preferences > Workspace** to open the dialog box. Make your changes and click **OK**.



Specify Maximum Array Size on Which to Compute Statistics

If you show statistical columns in the Workspace browser, and if you work with very large arrays, you might experience performance issues when the data changes as MATLAB updates the statistical results. In that event, show only the columns of interest to you and hide those you do not need.

Another step you can take is specify via a preference that the Workspace browser *not* perform statistical calculations on the largest arrays. Use the arrows to change the value of the maximum array size for which you want the Workspace browser to perform statistical calculations. The default value is 500,000 elements. Any variable exceeding that size reports <Too many elements> instead of statistical results.

Handling NaN Values in Calculations

If your data includes NaNs, you can specify that the statistical calculations consider the NaNs or ignore the NaNs. For example, if a variable includes a NaN and the preference is set to **Use NaNs when calculating statistics**, the values for **Min**, **Max**, **Var** and some others will appear as NaN, although **Mode**, for example, shows a numeric result. With the preference set to **Ignore NaNs whenever possible**, numeric results appear for most of the statistical columns including **Min** and **Max**; **Var**, however, is still reported as NaN.

For more information about statistical values in the Workspace browser, see “Viewing and Editing Values in the Current Workspace” on page 6-4.

Viewing and Editing Workspace Variables with the Variable Editor

In this section...

“About the Variable Editor” on page 6-24

“Opening the Variable Editor” on page 6-24

“Working with Different Types of Data in the Variable Editor” on page 6-27

“Navigating and Editing Shortcut Keys for the Variable Editor” on page 6-34

“Changing Size, Content, and Format of Variables in the Variable Editor” on page 6-35

“Cut, Copy, Paste, and Clear Contents in the Variable Editor” on page 6-36

“Other Variable Editor Operations” on page 6-40

“Creating Graphs and Variables, and Data Brushing in the Variable Editor” on page 6-41

“Preferences for the Variable Editor” on page 6-46

About the Variable Editor

The Variable Editor is a desktop component that lets you display variables in the current workspace. Use it to view and edit values of one or two-dimensional arrays, character strings, cell arrays, structures, and objects and their properties. You can also view the contents of multidimensional arrays. Edits you make in the Variable Editor immediately update the variable in the workspace. It also supports copying and pasting of data values.


Opening the Variable Editor

To open the Variable Editor from the Workspace browser, perform these steps:

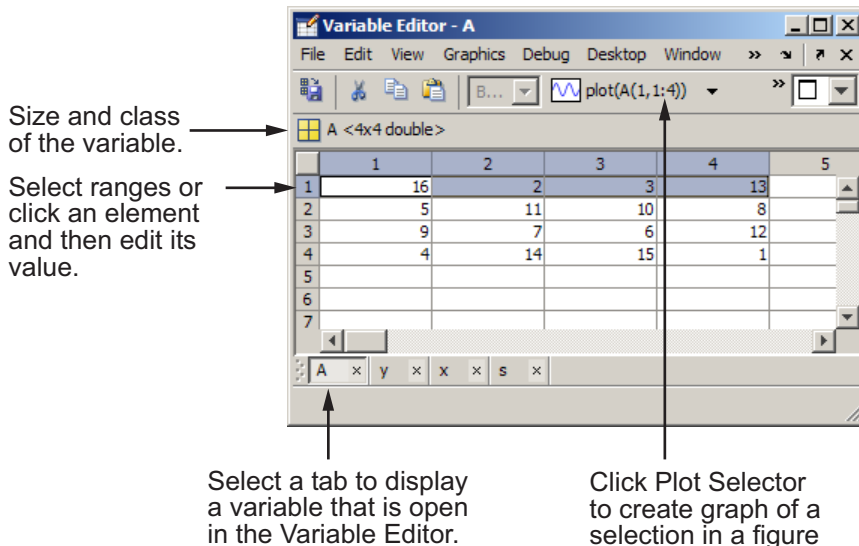
- 1 If you do not have any, create some workspace variables, for example:

```
A = magic(4);  
x = 0:.1:4*pi;  
y = sin(x);
```

```
s = sprintf('This is yext\nwith two lines');
```

- 2 In the Workspace browser, select the variable you want to open. Use **Shift**+click or **Ctrl**+click to select multiple variables, or use **Ctrl+A** to select all variables to open.
- 3 Click the Open Selection button  on the toolbar. For one variable, you can also open it by double-clicking it.

The Variable Editor opens, displaying the values for the selected variable. The class and size of the value appear below the toolbar, and for some classes, include a link to the help for that class.



Repeat the steps to open additional variables in the Variable Editor. Access each variable via its tab at the bottom of the window, or use the **Window** menu.

Changes you make to variables via the Command Window or other operations automatically update the information for those variables in the Variable Editor.

Note MATLAB software does not limit the maximum number of elements in a variable that you can open in the Variable Editor. The limit is based on your operating system or the amount of physical memory installed on your system.

Keyboard Alternatives

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

```
openvar('A')
```

You need to enclose the name of the variable name in single quotes, because the Variable Editor requires strings, not variable references. It needs the name of the variable so MATLAB can notify it when the variable changes value, disappears, or goes out of scope. If you were to type `openvar(A)` instead of `openvar('A')`, the Variable Editor would receive the value of `A` instead of its name. However, `openvar varname` and `openvar 'varname'` both work, as the function assumes string arguments when using *command syntax*. See “Command vs. Function Syntax” in the MATLAB Programming Fundamentals documentation for more information.

MATLAB opens `A` in the Variable Editor.

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

```
A
```


and MATLAB returns

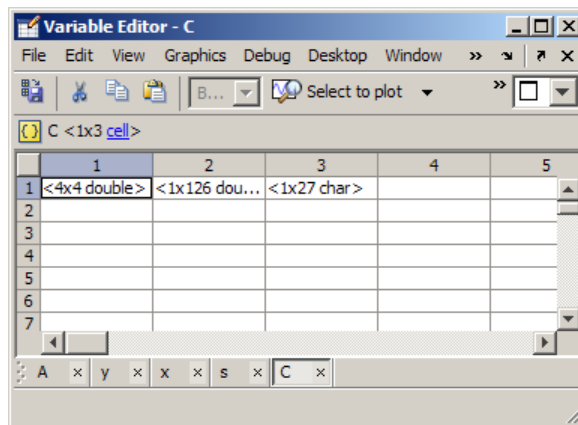
```
A =  
    16     2     3    13  
     5    11    10     8  
     9     7     6    12  
     4    14    15     1
```

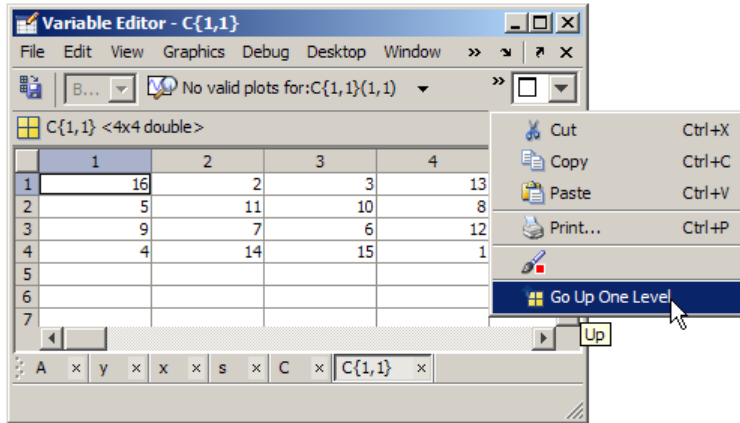
Working with Different Types of Data in the Variable Editor

- “Cell Arrays — Viewing and Editing in the Variable Editor” on page 6-27
- “Structures — Viewing and Editing in the Variable Editor” on page 6-28
- “Objects and Their Properties — Viewing and Editing in the Variable Editor” on page 6-30
- “Multidimensional Arrays — Viewing in the Variable Editor” on page 6-33

Cell Arrays — Viewing and Editing in the Variable Editor

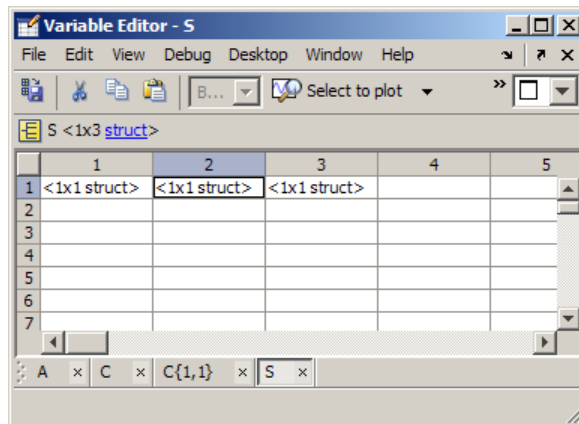
You can view and edit the content of cell arrays in the Variable Editor. In the Variable Editor, double-click an element of a cell array to open it as its own Variable Editor document. You can then view and edit the contents of that element. The following illustrations show a 1-by-3 cell array, `C`, and the contents of `C{1,1}`. When viewing an element in a cell array, for example, `C{1,1}`, use the Up button  to go to its cell array, for this example, `C`.

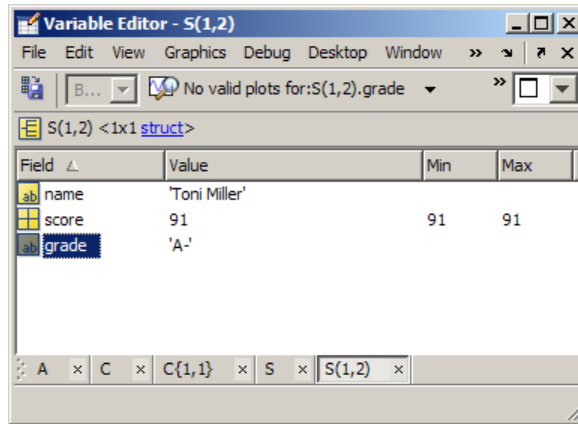





Structures – Viewing and Editing in the Variable Editor

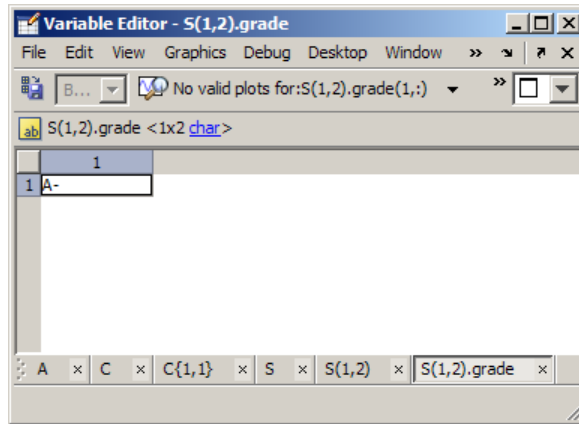
You can view and edit the content of structures in the Variable Editor. In the Variable Editor, double-click an element of a structure to open it as its own Variable Editor document. The following illustrations show a 1-by-3 structure, S and the result of double-clicking S(1,2), which displays the contents in its own new document.





The information shown for the element of the structure is like what the Workspace browser displays: **Field**, **Value**, **Size** and other information. To show or hide columns, select **View > Choose Columns**. On Microsoft Windows systems, you can right-click any column header to hide it or to show or hide other columns. Click a column header to sort by that column, and click again to reverse the sort order. When viewing an element in a structure, for example, $S(1,2)$, use the Up button  to view the structure, for this example, S . The button helps you navigate in the Variable Editor when there are many variables open.

To edit an element, you can click the value and change its value. Or double click the element; a new Variable Editor document containing it opens. Click the value and then change it. The following illustration shows the result of double-clicking the `grade` field for $S(1,2)$, where you can change its value. You can use the Up button go up from the field to view the element. For example, when viewing $S(1,2).grade$, click the Up button to view $S(1,2)$.

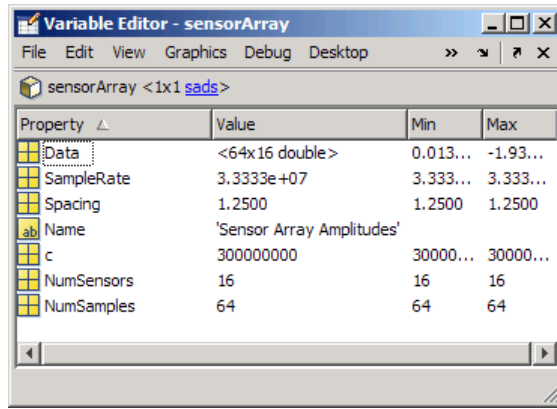


Objects and Their Properties – Viewing and Editing in the Variable Editor

- “Viewing Object Properties in the Variable Editor” on page 6-30
- “Editing Property Values in the Variable Editor” on page 6-31
- “Getting Help for Objects and Properties from the Variable Editor” on page 6-33

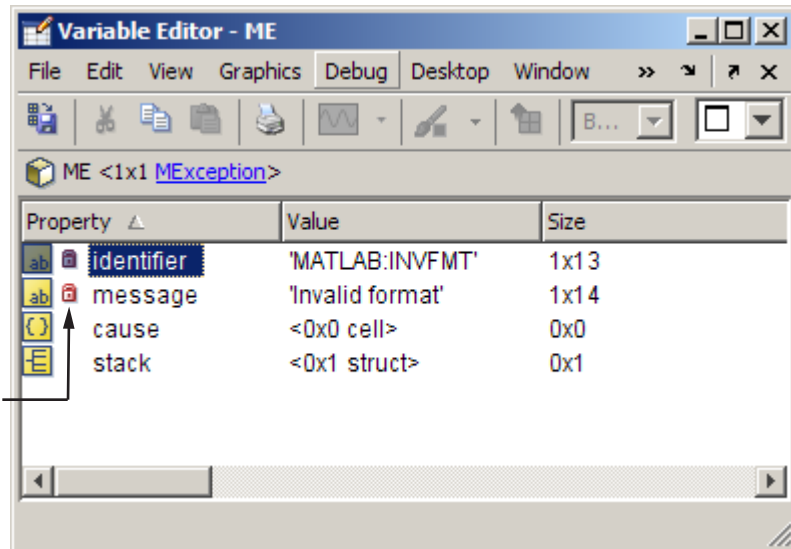
Viewing Object Properties in the Variable Editor. In the Variable Editor, you can view and edit properties of many MATLAB objects you create. When you open an object in the Variable Editor, it displays **Property**, **Value**, **Size**, and other information. To show or hide a column, right-click the column header. To sort by a column, click that column header; to reverse the sort order, click the column header again. You can view help for the class by clicking the class name link. The Variable Editor has special attributes for timeseries objects; for more information, see “Viewing Time Series Objects” in the MATLAB Data Analysis documentation.

The following illustration shows the `sensorArray` object of the `sads` class, in the Variable Editor. For more information about this example, see “Example of Help for a Externally Supplied Class” on page 5-14.

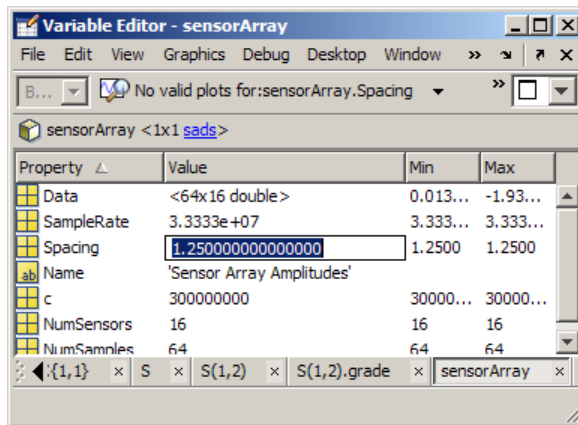



Additional icons, images of locks, denote protected and private properties of an object, indicating you cannot edit the values. The following illustration shows an MException object, ME, with the private properties identifier and message.

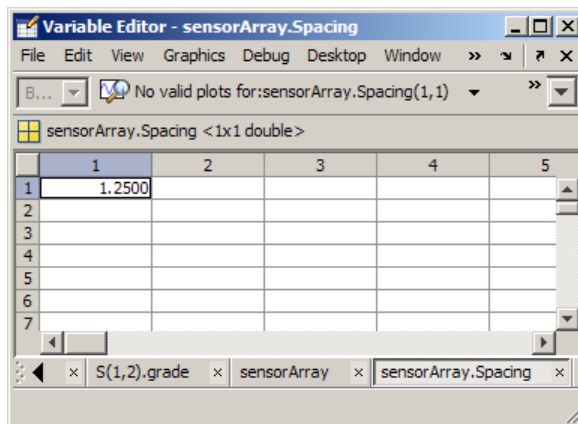
Icon denotes that identifier and message properties are private, and therefore you cannot edit their values.



Editing Property Values in the Variable Editor. To edit a property value while viewing the object, click the value field and edit its contents, as shown in the following illustration.



Alternatively, double-click the value, which displays the value in its own document where you can more easily view and edit it. In the following illustration, the `sensorArray.Spacing` property opens in its own document when you double-click it. When viewing a property, use the **Up** button  to view the object, for this example, `sensorArray`. This button can help you navigate in the Variable Editor when there are many variables open.

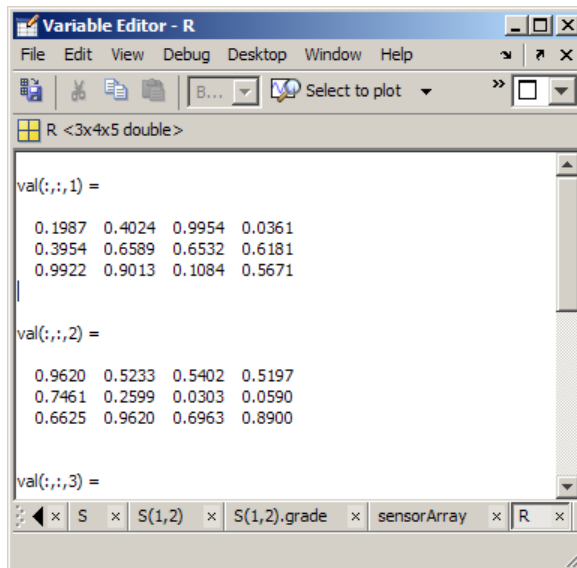


If the Variable Editor window is small, the **Up** button might not be visible. To get to it, click the **>> More Menu** button on the right side of the toolbar and choose **Go Up One Level** from the menu.

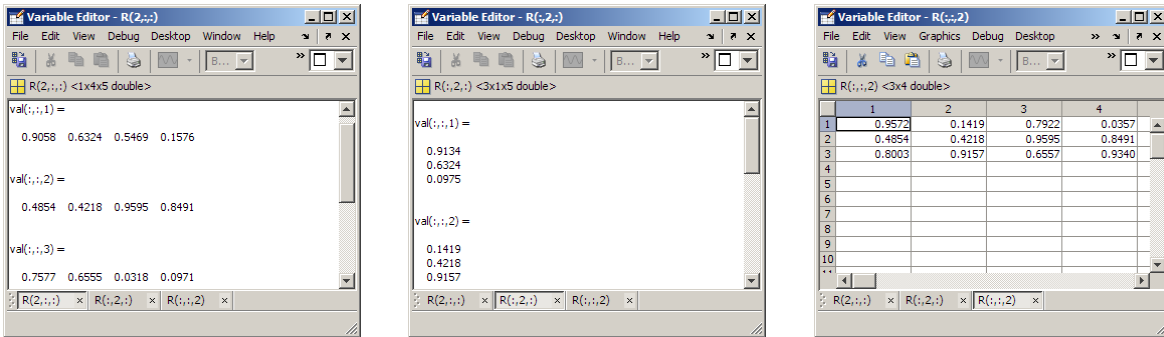
Getting Help for Objects and Properties from the Variable Editor. For most classes supplied by The MathWorks, when you click the link to the class name, for example, `char`, the reference page displays in the Help browser. For user-created classes, help comments supplied in the class definition file display in HTML format in the Help browser. For more information, see “Adding Help for Classes You Create” on page 5-13.

Multidimensional Arrays – Viewing in the Variable Editor

You can view the contents of multidimensional arrays in the Variable Editor. When you open a multidimensional array in the Variable Editor, it does not have usual grid structure, because multidimensional arrays do not fit that format. You cannot double-click an element in a multidimensional array to edit it. The following illustration shows `R = rand(3,4,5)` opened in the Variable Editor.



You can view subsets of multidimensional arrays as long as the indexing expression evaluate to either a 1-D vector or a 2-D matrix. For example, `R(2, :, :)`, `R(:, 2, :)`, and `R(:, :, 2)` display as follows.



You cannot edit subsets of multidimensional matrices. Because you can index into matrices in so many ways, the Variable Editor can incorrectly identify subscripts of variable elements that you might change. To avoid changing the wrong data elements, the Variable Editor prevents you from editing multidimensional matrices.

Navigating and Editing Shortcut Keys for the Variable Editor

Use the following shortcut keys (sometimes called hot keys) to move among elements in the Variable Editor. Navigating in the Variable Editor is much like navigating in the Microsoft Excel application.

Key	Result
Enter	Commit any changes to the element and move to next element. “Preferences for the Variable Editor” on page 6-46 let you specify what the next element is (the default is down)
Tab	Move right Within a selection, also moves from the last column to the first column in the next row
Shift+Enter or Shift+Tab	Move in opposite direction of Enter or Tab

Key	Result
Page Up	Move up m rows, where m is the number of visible rows
Page Down	Move down m rows, where m is the number of visible rows
Home	Move to column 1
Ctrl+Home	Move to row 1, column 1
Shift+Home	Select to column 1
End	Move to last column in current row
F2 (Ctrl+U on Apple Macintosh platforms)	Edit current element, positioning cursor at the end of the element

Changing Size, Content, and Format of Variables in the Variable Editor

To increase the size of an array, scroll to the desired element in the variable and enter a value. The array automatically expands to accommodate the new value. Empty elements fill with zeros if numeric, or empty arrays if a cell array. To decrease the size of an array, select the rows or columns that you want to remove by clicking in the row or column header. Clicking a header selects the entire row or column. Then right-click, and select **Delete** from the context menu. Similarly, you can update arrays in structure and objects.

To change the value of an element in the Variable Editor, click the element and type a new value. Press **Enter**, or click another element to effect the change. You can specify where the cursor moves to after you press **Enter** — see “Preferences for the Variable Editor” on page 6-46.

If you want to change the display format for the Variable Editor, select the **View** menu and choose a format. To change the default format for future use, use the Preferences dialog. For more information, see “Preferences for the Variable Editor” on page 6-46.

If you open an existing MAT-file and change it using the Variable Editor, save that MAT-file if you want the changes to be permanent. For instructions, see “Saving the Current Workspace” on page 6-5.

Cut, Copy, Paste, and Clear Contents in the Variable Editor

You can cut or copy selected elements, rows, and columns in an array and paste them to another position in that or another open array. To select a column or row, click the row or column header (the element that shows the row or column number). Use **Shift**+click to choose contiguous elements, rows, or columns in the array, or **Ctrl+A** to select all elements. For the cut, copy, and paste operations, use the **Edit** menu, the context menu, or the toolbar buttons. You can undo the last operation you performed in the Variable Editor.

When you cut elements, the value of each element you cut becomes 0 (if numeric) or [] (if a cell array). After cutting, select the elements whose value you want to replace with the cut elements and then choose **Edit > Paste**. If the shape of the elements you cut differs from the shape of the elements into which you are pasting, the Variable Editor pastes all the elements. Either it expands the size of the selection you made, or it expands the array size to allow all the elements you are pasting. Pasting copied elements is the same as pasting cut elements, but the elements copied maintain their value rather than becoming 0 or [].

To make the value of elements 0, select elements, rows, or columns and then select **Edit > Clear Contents**. Clearing differs from cutting because the data from the selected elements does not move to the clipboard or modify it.

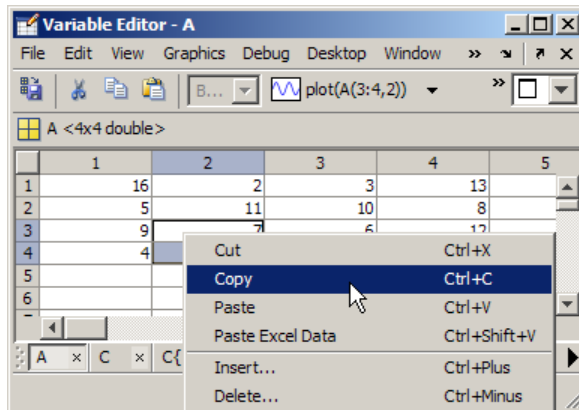
Example: Copying and Pasting Array Elements

In this example, you copy two elements. When you select one element for pasting, it replaces two elements.

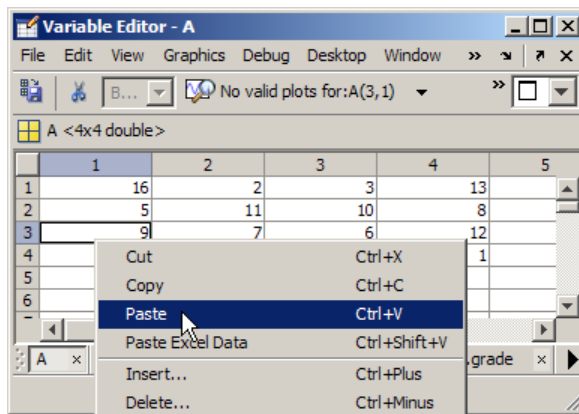
- 1 Create a matrix variable.

```
A = magic(4);
```

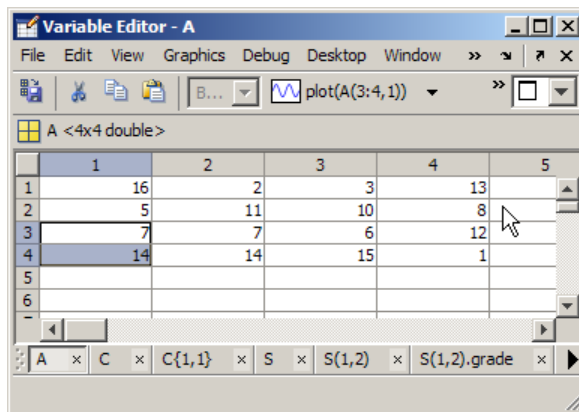
- 2 Select rows 3 and 4 or column 2 by clicking 7 (A(3,2)) and then **Shift**+clicking 14 (A(4,2)). Right-click the selection and select **Copy** from the context menu. You can also press **Ctrl+C** or chose **Copy** from the **Edit** menu to copy the values.



- 3 Select 9 (A(3,1)) and select **Paste** from the context menu or **Edit** menu, or type **Ctrl+V**.



The column vector you copied ($[7; 14]$) replaces the contents of rows 3 and 4 in column 1 (which had been $[9, 4]$), even though you only selected the element containing 9. That is, the shape of the copied elements determines which values get replaced, starting at the upper left element.



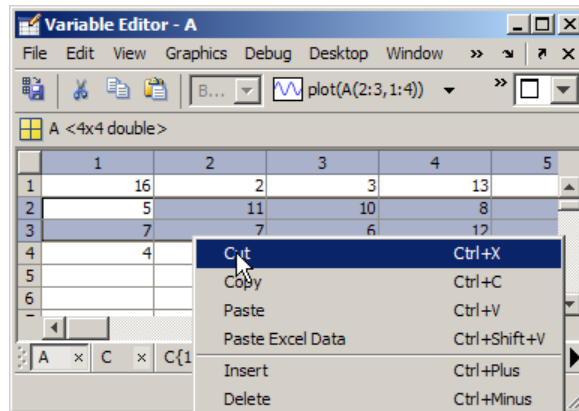
Example: Cutting and Pasting Array Elements

select two rows and cut their contents. Select one row for pasting. The Variable Editor expands the array size, adding a row to accommodate all cut elements. The values of the elements you cut becomes 0.

- 1 Create a matrix variable.

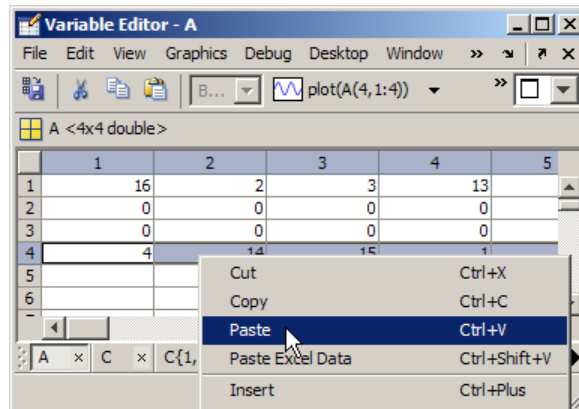
```
A = magic(4);
```

- 2 Select rows 2 by clicking its row number, then **Shift**+click the row number for row 3 to select both rows. Right-click the selection and select **Cut** from the context menu. You can also press **Ctrl+X** or chose **Cut** from the **Edit** menu to cut the values and copy them to the clipboard.

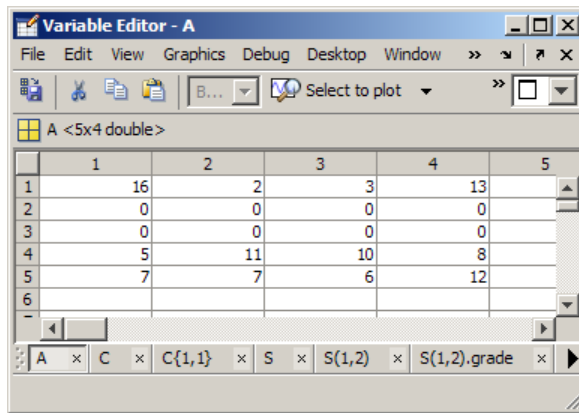


The values in the cut rows all become 0 as a result of the cut operation.

- 3 Select row 4 entirely and select **Paste** from the context menu or **Edit** menu, or type **Ctrl+V**.



The contents of the cut rows replace row 4 and extend the matrix to have an additional row.



Other Variable Editor Operations

Insert and Delete in the Variable Editor

You can insert and delete elements, rows, and columns in arrays in the Variable Editor. When you select **Edit > Insert**, or **Edit > Delete**, a dialog box appears in which you specify rows, columns, or elements. When you delete elements, the Variable Editor prompts you to provide, the direction for shifting existing elements.

Undo and Redo in the Variable Editor

You can undo the last action you performed in the Variable Editor, or redo a change after choosing undo. Select **Edit > Undo** or **Edit > Redo**. The actions supported are the following:

- A change to a value you make by editing it in the Variable Editor
- Cutting
- Pasting
- Inserting
- Deleting
- Clearing contents
- Pasting data from the Microsoft Excel application.

Exchanging Data with the Command Window

You can copy data from an array in the Variable Editor and paste it into the Command Window. You can also copy a value from the Command Window and paste it into an element in the Variable Editor. Be sure that the data types are compatible. For example, you cannot paste text from the Command Window into a numeric array in the Variable Editor.

Creating New Workspace Variables from the Variable Editor

You can also create new variables from a selected element, data range, row, or column in an array in the Variable Editor. Right-click, and from the context menu, select **Create Variable from Selection**, or do the same from the **Edit** menu.

Exchanging Data with the Microsoft Excel Application

You can cut or copy cells from the Microsoft Excel application and paste them into the Variable Editor—use **Edit > Paste from Excel**. You can also cut or copy elements from an array in the Variable Editor and paste them into the Excel® application.

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

Creating Graphs and Variables, and Data Brushing in the Variable Editor

The Variable Editor, like the Workspace Browser, provides several methods for creating graphs without typing plotting commands. Once a graph displays, you can “brush” either the graph or array elements in the Variable Editor to see which observations correspond in the other.

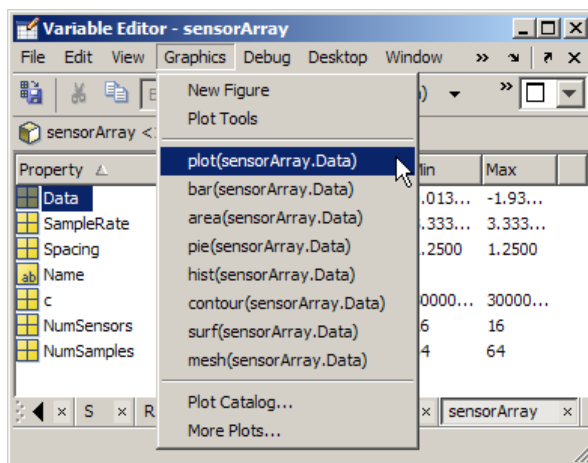
Generating Graphs Automatically

You can create graphs from selected variables in the Variable Editor. To create a graph, select a data range, row, or column in an array, and choose a graph type in one of the ways described in the following bullets. MATLAB examines the selected data and determines which kinds of graphs can display

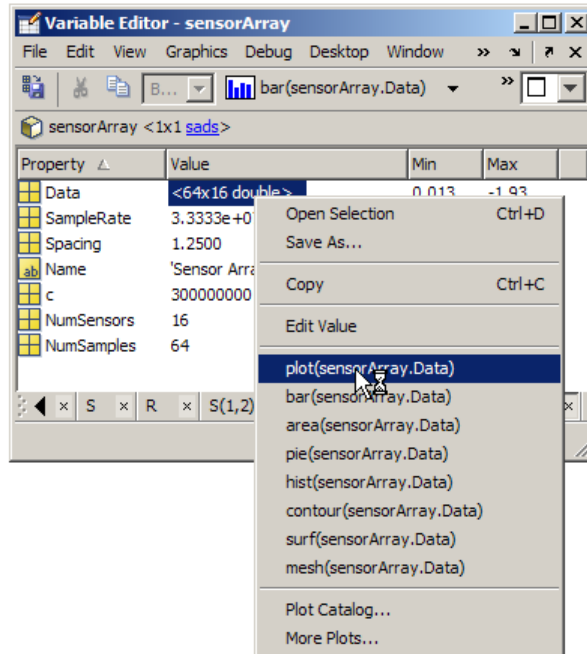
it. In some cases, MATLAB performs data conversion, such as using `cell2mat` to transform cell array data—which cannot be plotted directly—to matrix data. For more information, see “Plotting Process” in the MATLAB Getting Started Guide.

You can graph selections of numeric data and selected objects from the Variable Editor in three ways, illustrated here:

- Select data and choose from a list of graph types from the **Graphics** menu.

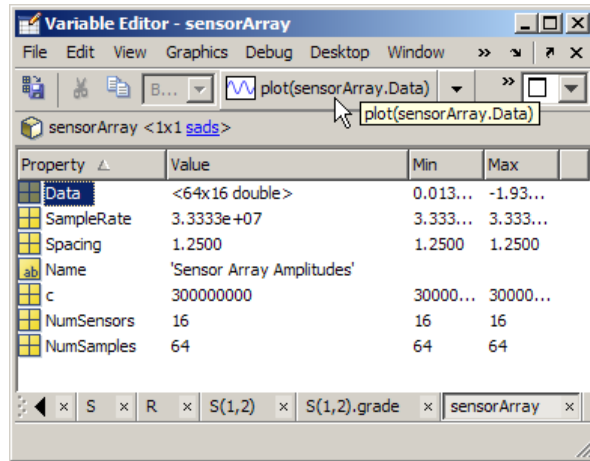


- Select data, right-click, and choose from a list of graph types from the context menu.

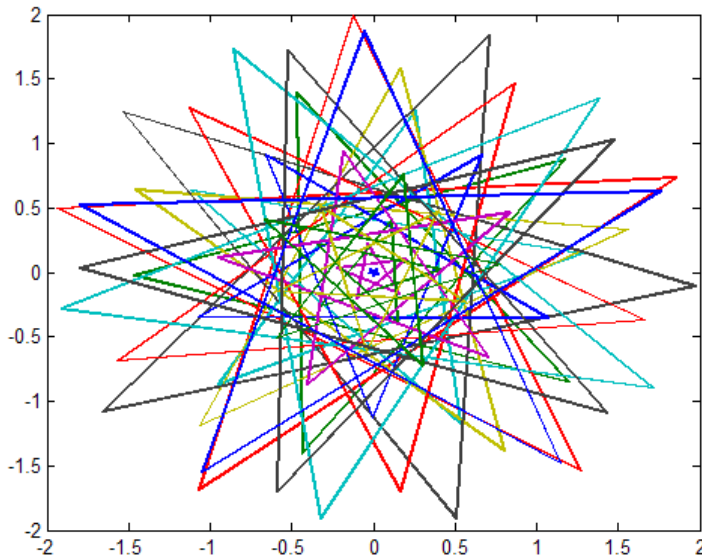


The types of graphs available on the context menu and the **Graphics** menu are the same.

- Select data and click the Plot Selector toolbar icon to generate the type of plot it displays.

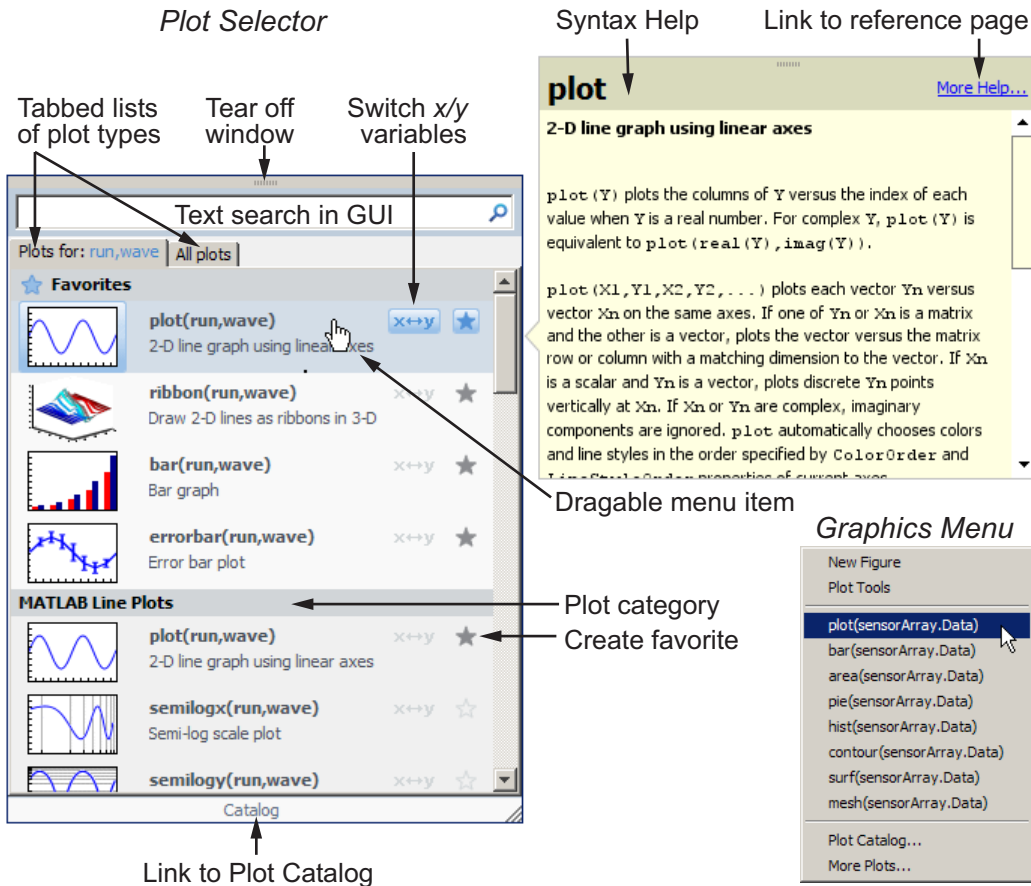


Assuming that you select the same graph type, all three methods generate identical plots of the selected data in the current or a new figure window.



The Plot Selector is the most flexible of the three methods. It lists more graph types you can currently make and, in a separate tab, all graph types available


to you. It also provides function help, and lets you prioritize graph types as a list of favorites. The following illustration compares it to the Graphics menu.




For more information about using the Plot Selector, see “Creating Plots from the Workspace Browser” on page 6-10. The plot selector supports certain toolbox plotting functions in addition to those in MATLAB; see “Plot Selector Supports Additional Toolboxes” for more information.

Brushing Data in Linked Graphs

Data brushing is a technique for exploring where specific data observations fall in a set of graphs and tables. It helps you to visually identify relationships,

outliers, trends, and noise that can be difficult to determine with numerical or statistical methods. Use the Data Brushing Tool  on the Variable Editor and figure toolbars to mark specific observations (or ranges of them) in the Variable Editor and on graphs. You can remove brushed observations or save them to new variables.

If a variable you brush in the Variable Editor is plotted on a graph, selecting the Data Brushing tool and brushing array elements in the Variable Editor highlights those values in the graph displaying the variable you brush. Likewise, brushing observations on a linked plot highlights them in the Variable Editor. For data brush to communicate between the two windows, the figure must be in *Linked Plot*  mode. Linked Plot mode connects a graph's XData, YData and ZData to its data sources in the current workspace. For more information, see “Data Brushing with the Variable Editor” in the MATLAB Data Analysis documentation and the reference pages for brush and linkdata.

Preferences for the Variable Editor

To set preferences for the Variable Editor, select **File > Preferences > Variable Editor**. The Preferences dialog box opens showing **Variable Editor Preferences**.

Format

Specify the default array output format of numeric values displayed in the Variable Editor. This format preference affects only how numbers display, not how MATLAB computes or saves them. For more information, see the reference page for format.

Editing

You can specify where the cursor moves to after you type an element and press **Enter**:

- If you want the cursor to remain at the element where you 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. Choose the **Direction** to specify how

you want the cursor to move. For example, if you want the cursor to move right one element after you press **Enter**, select **Right**.

International Number Handling

You can specify the decimal format of numbers you cut or copy from the Variable Editor when you paste them into text files or other applications. The **Decimal separator for exporting numeric data via system clipboard** edit field is by default "." (period). If you are working in or providing data to a locale that uses a different character to delimit decimals, type that character in this preference and click **OK** or **Apply**. This preference has no effect on numeric data copied from and pasted into MATLAB documents or into the Command Window. Within MATLAB, decimal separators are always periods.

Managing Files in MATLAB

- “Introduction to Managing Files in MATLAB” on page 7-2
- “Understanding File Locations in MATLAB” on page 7-4
- “Working with Files and Folders” on page 7-12
- “Finding Files and Folders” on page 7-27
- “Creating, Opening, Changing, and Deleting Files and Folders” on page 7-36
- “Comparing Files and Folders” on page 7-50
- “Making Files and Folders Accessible to MATLAB” on page 7-66
- “Using the MATLAB Search Path” on page 7-72
- “Related Topics for Managing Files” on page 7-84

Introduction to Managing Files in MATLAB

In this section...
“Ways to Manage MATLAB Files” on page 7-2
“Tools for Managing Files” on page 7-2

Ways to Manage MATLAB Files

MATLAB functions and desktop tools help you:

- Find a file you want to open, edit, load, or run in MATLAB
- Organize your files
- Obtain information about the contents and status of a file
- Ensure that MATLAB can access a file so that you can run or load it.
Typically, any file you access must be in the current folder or in a folder that is on the MATLAB search path.

For a broad introduction, see the “Managing Files in MATLAB” topic in the MATLAB Getting Started guide.

Tools for Managing Files

You can create, open, close, read, write, rename, move, and delete files and folders using MATLAB functions. For a listing of them and links to their reference pages, see “File Operations”.

MATLAB also provides many interactive tools for working with files. Click the name of a tool in the following table to learn more about it.

Desktop Tool	Description
Current Folder Browser	View files, perform file operations such as opening, finding files and viewing file content, and managing and tuning your files.
Find File tools	Locate files by folder, file type, size, text within them, and other criteria.

Desktop Tool	Description
Comparison Tool	View line-by-line differences between two files.
Editor	Create, edit, debug, and analyze files containing MATLAB language statements (functions and scripts), and view and edit other text files.
File Exchange	Access a repository of files, created by users for sharing with other users, at the MathWorks Web site.

Understanding File Locations in MATLAB

In this section...
“Important MATLAB Folders” on page 7-4
“Path Names in MATLAB” on page 7-7

Important MATLAB Folders


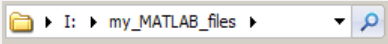
When you work with files and folders, be aware of key locations that MATLAB uses.

The Current Folder

The *current folder* is a reference location that MATLAB uses to find files. This folder is sometimes referred to as the *current directory*, *current working folder*, or *present working directory*. It is *not* the same location as the operating system current folder.

You can always load files and execute scripts and functions that are in the current folder, even if that folder is not currently on the MATLAB search path. Functions in the current folder take precedence over functions with the same file name that reside anywhere on the search path.

Viewing and Changing the Current Folder. You can view and change the current folder using various desktop tools and functions, as described in the following table. To specify the current folder programmatically when MATLAB starts, see “Startup Folder for the MATLAB Program” on page 1-8.

To:	Do this:
Identify the Current Folder	<p>Use one of the following:</p> <ul style="list-style-type: none"> The Current Folder field on the Desktop toolbar.  <ul style="list-style-type: none"> The Current Folder browser address bar — if the full path is not visible, hover over the folder icon.  <ul style="list-style-type: none"> The pwd or cd function.
Change the current folder to one you specify	<p>Do one of the following:</p> <ul style="list-style-type: none"> In the Current Folder field on the Desktop toolbar, type or browse to a different folder. On the Current Folder browser address bar, click an arrow that appears between portions of the path, and then choose a drive or subfolder from the drop-down list. Use the cd function.
Change the current folder to a recently used folder	<p>From the Current Folder field on the Desktop toolbar, click the down arrow, and then select a folder from the history.</p>
Change the current folder to an active document's folder	<p>Right-click the document tab in the Editor, and then select Change Current Folder to folder.</p> <p>Document tabs appear only when more than one document is open in the Editor.</p>
Make a subfolder the current folder	<p>In the Current Folder browser, right-click the subfolder, and then select Open from the context menu.</p>

To:	Do this:
Copy the Current Folder as a string	<ul style="list-style-type: none"> • Select the address in the Desktop toolbar, right-click, and then select Copy. • Click an empty area on the right edge of the Current Folder address bar, right-click, and then select Copy.
Get help using the Current Folder address bar	Right-click an empty area on the address bar, and then from the context menu, select Help Using Address Bar

matlabroot

matlabroot is the folder where you installed MATLAB. The location differs for each installation of MATLAB. Determine its location by running the *matlabroot* function. When you start MATLAB, your current folder can be *matlabroot*, but in practice it is usually a different folder.

The Startup Folder

Each time you start MATLAB, your current folder is always the same. This location is called the *startup folder*. The operating system commands that runs MATLAB specifies the location of the startup folder. You can configure MATLAB to make your initial current folder a different location. For more information, see “Startup Folder for the MATLAB Program” on page 1-8.

Locations of MathWorks Products

Files and folders for products provided by MathWorks are in *matlabroot/toolbox*. The files and folders under *matlabroot* are important to your installation. In particular:

- Do *not* store your personal files and folders in *matlabroot/toolbox*.
- Do *not* change files, folders, and subfolders in *matlabroot/toolbox*. The exception is the *pathdef.m* file, which you can update and save in its default location, *matlabroot/toolbox/local*.

To improve performance, at the beginning of each session, MATLAB loads and caches in memory the locations of files in *matlabroot/toolbox*. If you

make changes to files and folders in *matlabroot*/toolbox, running functions can produce unexpected results or generate warnings, that are related to the toolbox cache. See “Toolbox Path Caching in the MATLAB Program” on page 1-19.

To see a list of all toolbox folder names supplied with MathWorks products, run:

```
dir(fullfile(matlabroot, '/toolbox'))
```

Locations for Storing Your Files

For your convenience, MATLAB provides a folder called MATLAB to store your files. At startup, MATLAB adds the folder to the search path, allowing MATLAB to access the files stored there.

The location of the *userpath* MATLAB folder varies by platform and system configuration. To determine the location, run the *userpath* function.

On Microsoft Windows platforms, MATLAB sets the current folder to *userpath* at startup. On other platforms, you instruct MATLAB differently to set the current folder to *userpath* at startup. For more information, see “Startup Folder for the MATLAB Program” on page 1-8.

If you create subfolders within the MATLAB folder, make the new subfolders accessible to MATLAB.

If you store files in locations other than the MATLAB folder:

- Make the files accessible to MATLAB by adding their folders to the search path.
- Do not store the files in the folders provided for MathWorks products.

Path Names in MATLAB

A path name specifies file locations, for example, C:\work\my_data (on Microsoft Windows platforms) or /usr/work/my_data (on UNIX or Macintosh platforms). Path name specifications differ, depending on the platform on which you are running MATLAB. When you work with files and folders, be

aware of how MATLAB uses path names and the restrictions it places on them.

Specifying Path Names on Macintosh Systems

When you specify path names on Macintosh, do not use accent characters. If path names include such characters, for instance umlauts or circumflexes, the Current Folder browser and MATLAB cannot recognize the path. In addition, attempts to save a file to such a path results in unpredictable behavior.

**Specifying File Separator Characters, / and **

The file separator character is the symbol that distinguishes one folder level from another in a path name.

A forward slash (/) is a valid separator on any platform. A backward slash (\) is valid only on Microsoft Windows platforms.

In the full path to a folder, the final slash is optional.

Type `filesep` in the Command Window to determine the correct file separator character to use when working with files programmatically.

Specifying Absolute and Relative Path Names

MATLAB always accepts *absolute* path names (also called *full* path names), such as `I:/Documents/My_Files`. An absolute path name can start with any of the following:

- UNC path '\\\ ' string
- Drive letter, on Microsoft Windows platforms, such as `C:\`.
- `'/'` character on UNIX¹⁰ platforms

Some MATLAB functions also support relative path names. The reference page for a function specifies the valid types of path name. Unless otherwise noted, the path name is relative to the current folder. For example:

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

- `/myfolder` refers to the `myfolder` folder in the current folder and `myfile.m` refers to the `myfile.m` file in the current folder.
- `../myfolder/myfile.m` refers to the `myfile.m` file in the `myfolder` folder, where `myfolder` is at same level as the current folder. Each repetition of `../` at the beginning moves up an additional folder level.

Tip If multiple documents are open and docked in the Editor, you can copy the absolute path of any of these documents to the clipboard. This is useful if you need to specify the absolute path in another MATLAB tool or an external application. Right-click the document tab, and then select **Copy Full Path to Clipboard**

Maximum Length of Path Names in MATLAB

The maximum length allowed for a path name depends on your platform.

For example, on Microsoft Windows platforms:

- The maximum length is known as `MAX_PATH`.
- You cannot use an absolute path name that exceeds 260 characters.
- For a relative path name, you might need to use fewer than 260 characters. When the Windows operating system processes a relative path name, it can produce a longer absolute path name, possibly exceeding the maximum length.

If you get unexpected results when working with long path names, use absolute instead of relative path names. Alternatively, use shorter names for folders and files.

Constructing Path Names on Different Platforms

Use `fullfile` to construct path names in statements that work on any platform. This function is particularly useful when you provide code to someone using it on a platform different from your own. The `ismac`, `ispcc`, and `isunix` functions identify the platform you are currently using.

Including Spaces in Path Names

When a function argument is a file or path name, and the name includes spaces, use the function syntax. For example:

```
delete('temp file.m') % Function syntax works for a file name containing a space
```

The command syntax does not work. For example:

```
delete temp file.m % Command syntax does NOT work for a file name containing a space
```

Partial Path Names in MATLAB

A partial path name is the last portion of a full path name for a location on the MATLAB search path.

Some functions accept partial path names. The reference page for a function typically specifies the valid types of path names.

Examples of partial path names are: `matfun/trace`, `private/cancel`, and `demos/clown.mat`.

Use a partial path name to:

- Specify a location more conveniently than by using the full path name.
- Specify a location independent of where MATLAB is installed.
- Locate a function in a specific toolbox when multiple toolboxes contain functions with that name. For example: to get help for the `set` function in the Database Toolbox™ product, type:

```
help database/set
```

For help for the `plot` method for time series objects, type:

```
help timeseries/plot
```

- Locate private and method files, which sometimes are hidden.

Be sure to specify enough of the path name to make the partial path name unique. Specifying the `@` in method folder names is optional.

See Also

- “Slash and Backslash — / \”
- “Naming Functions”
- `ismac`, `ispc`, and `isunix` functions, for MATLAB statements that require different path names for different platforms
- “Private Functions” in the MATLAB Programming Fundamentals documentation.

Working with Files and Folders

In this section...
“Viewing Folder Contents” on page 7-12
“Using the Current Folder Browser” on page 7-18

Viewing Folder Contents

- “Opening the Current Folder Browser” on page 7-13
- “Preferences for the Current Folder Browser” on page 7-14
- “Refreshing the List of Files” on page 7-15
- “Viewing Hidden Files and Folders” on page 7-16
- “Controlling the Appearance of Files Inaccessible to MATLAB” on page 7-16
- “Using Functions to Get Details About Files and Folders” on page 7-18

You can view information about folders from the MATLAB Desktop like you can from operating system windows. You can also type commands to describe files as you can from a command shell. For a summary of MATLAB functions you can use, see “Using Functions to Get Details About Files and Folders” on page 7-18 .

The principal Desktop tool for working with files and folders is the *Current Folder browser*. Like other Desktop components, you can dock the Current Folder browser or open it as a separate window. The Current Folder browser displays details about files in your current folder and within the hierarchy of the folders it contains. You can modify the kinds of information it displays to suit your needs, for example by reordering or deleting specific columns of information.

The Current Folder browser:

- Always displays your current folder, as well as its subfolders.
- Lets you access operating system file management features from within MATLAB.

- Is similar to file browsers provided with operating systems, but also includes features unique to MATLAB. For example, you can add folders to the search path from the Current Folder browser.

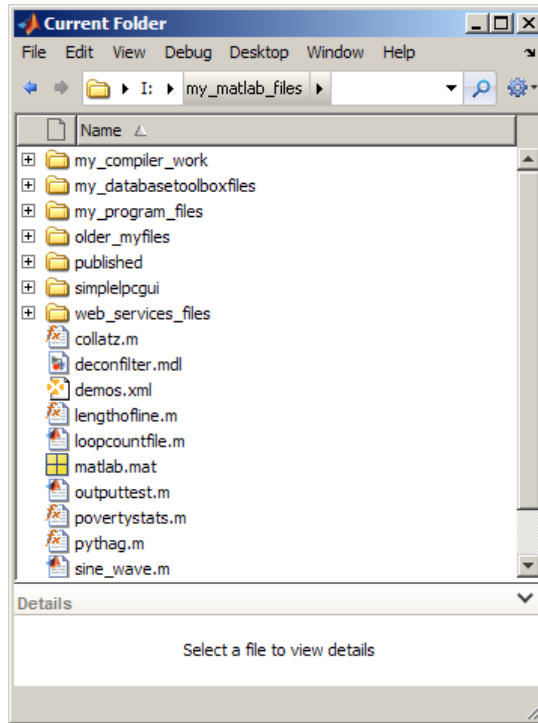
The following sections explain what you can do with the Current Folder browser and how to use it.

Opening the Current Folder Browser

Open the Current Folder browser by selecting **Desktop > Current Folder** from the MATLAB desktop. By default (when you start MATLAB for the first time) the Current Folder browser is docked on the left side of the desktop.

The Current Folder browser shows the full path to the current folder in the navigation bar, and shows the contents of the current folder in a pane underneath the bar.

Change the size, location, or other characteristics of the Current Folder browser as you would any tool on the MATLAB desktop. See Chapter 2, “Desktop”. You also can undock it from the desktop.



View the contents of subfolders and zip files by clicking the + (expand) button. Double-clicking a subfolder displays its contents, and makes that folder the current folder. Right-clicking a zip file and then selecting **Extract** extracts the files from it.

MATLAB might not be able to run files in the subfolders. See “Making Files and Folders Accessible to MATLAB” on page 7-66.

Preferences for the Current Folder Browser

You can set preferences for aspects of the Current Folder browser. Access these preferences by selecting **File > Preferences > Current Folder**. The preferences are:

- History — The number of recently used folders maintained in the Current Folder browser drop-down list.

- **Refresh** — How frequently the Current Folder browser updates to reflect changes to files made outside of MATLAB.
- **Path indication**— Controls the appearance of folders and files that are inaccessible to MATLAB, and whether to display tooltips describing their status.
- **Toolbar** — Provides a link to the Toolbars preferences. Those preferences enable you to adjust the toolbar layout and controls for Desktop tools, including the Current Folder browser.
- **Hidden files** — Controls whether the Current Folder browser displays hidden files and folders.

This preference not on available Microsoft Windows platforms.

Tip For information on changing the date format in the Current Folder browser, see “Customizing the Column Display” on page 7-19

Refreshing the List of Files

When files and folders are created, deleted, or changed outside of MATLAB, the Current Folder browser automatically reflects the changes. When you access files on a network, frequent refreshing of the Current Folder browser can slow performance in MATLAB. If this seems to be a problem, try improving the performance by changing how frequently refreshing occurs using the Current Folder **Refresh** preference:

1 Select **File > Preferences > Current Folder**.

By default, the **Auto-refresh view from file system** option is on, with an update time of 3 seconds. Every 3 seconds, the Current Folder browser checks for and reflects changes made from programs and tools other than MATLAB.

2 Try to improve responsiveness by either:

- Increasing the **Number of seconds between auto-refresh**.
- Clearing the **Auto-refresh view from file system** check box to turn off the feature.

3 Click **OK**.

To manually refresh the view at any time:

- 1 Right-click in the list area of the Current Folder browser.
- 2 Select **Refresh** from the context menu.

Viewing Hidden Files and Folders

The operating system, by default, hides certain files and folders from system file browsers and file-listing commands. The Current Folder browser can display hidden files and folders. You control this in different ways on different operating systems.

On Microsoft Windows platforms, the Current Folder browser follows the Windows preference for showing hidden files. To set or change the Windows preference:

- 1 Open Windows Explorer.
- 2 Select **Tools > Folder Options**.
- 3 Click the **View** tab.
- 4 Under **Advanced** settings, select **Show hidden files and folders**.

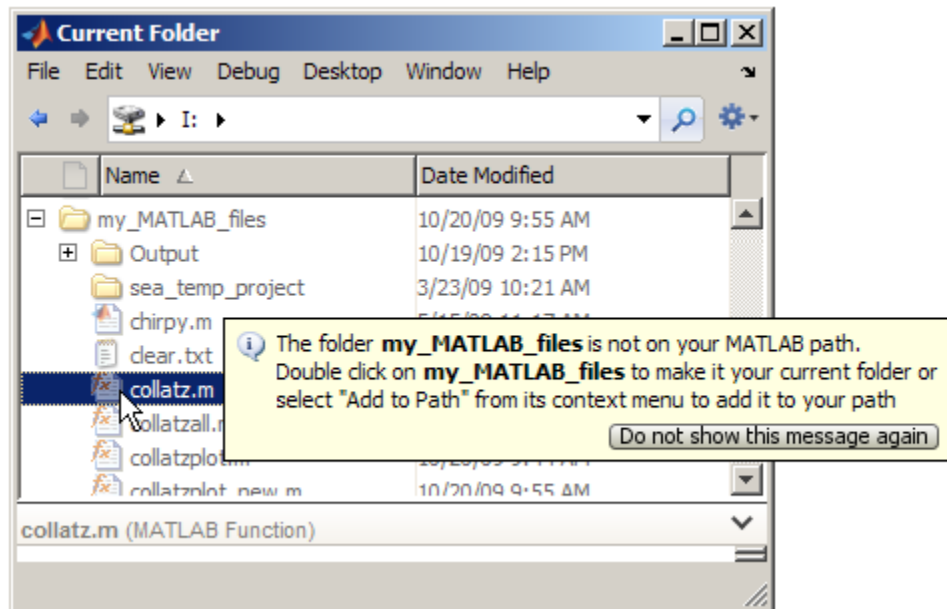
On other platforms, specify the behavior using Current Folder preferences:

- 1 Select **File > Preferences > Current Folder**.
- 2 Specify the setting for **Hidden files and folders**.

Controlling the Appearance of Files Inaccessible to MATLAB

MATLAB cannot access files if they are not on the search path or, in some cases, if they are in a private folder. By default, the Current Folder Browser dims the display of files and folders inaccessible to MATLAB. Furthermore, if you hover over a dimmed file, a tooltip provides information on why that file is inaccessible. If you disable this feature, the Current Folder browser

displays all files and folders as undimmed and provides no tooltips regarding their availability to MATLAB.



To customize this feature:

- 1** Select **File > Preferences > Current Folder**.
- 2** Select the **Indicate inaccessible files** check box to enable this feature; deselect it and skip to step 5 to disable this feature.
- 3** Move the **Text and icon transparency** slider to adjust the level of dimming.

View the region below the slider to preview how your choice will affect the appearance of files in the Current Folder browser.
- 4** Select **Show tooltip explaining why files are inaccessible** to enable tooltips; deselect it to disable them.
- 5** Click **OK**.

For more information, see “Private Folders” and “What Is the Search Path?” on page 7-72.

Using Functions to Get Details About Files and Folders

In the Command Window, you can list files, move to another folder, create folders, and delete files and folders. Most of these commands work like they do in an operating system command shell, but have different options. For example, some of the commands can return an argument in the form of a structure containing file information. The following table lists some of the actions you can perform with commands.

To...	Use This Function
Change your current folder	<code>cd</code>
Get the name, date, and size for a file or folder	<code>dir</code> or <code>ls</code>
Get and set the read-write status and other attributes for a file or folder	<code>fileattrib</code>
Separate a path name into folder and file name	<code>fileparts</code>
Build a file or folder name from parts	<code>fullfile</code>
Determine if a string corresponds to a folder	<code>isdir</code>
Move a file to a different folder	<code>movefile</code>
Create a folder	<code>mkdir</code>
Delete a file	<code>delete</code>
Delete a folder	<code>rmdir</code>
List files specific to MATLAB	<code>what</code>

Using the Current Folder Browser

- “Customizing the Column Display” on page 7-19

- “Viewing File Descriptions” on page 7-20
- “Viewing File Details Without Opening Files” on page 7-21
- “Viewing Help for a MATLAB Program File” on page 7-24
- “Sorting and Grouping Files and Folders” on page 7-24

The Current Folder browser lists details about files and folders in columns, beneath file and folder names, and in the details panel. Any file that is modified in the Editor, but not yet saved has an asterisk (*) next to it in the Current Folder browser. The browser displays columns for **Size**, **Date Modified**, **Type**, and **Description**. You can modify the information it displays. You also use this tool to perform operations on files and folders, such as moving, compressing, renaming, creating, and deleting them.

Note Do not use accented characters, such as à, é, ñ, or ü, in folder names. The Current Folder browser cannot locate folders containing such characters or save files to them.

Customizing the Column Display

You can show and hide columns, change their order, and adjust the date format in the Current Folder browser.

To Specify the Columns to Display.

- 1 Select **View > Show** or right-click on any column header.
- 2 Select the columns to show. Clear the columns to hide.

In addition, consider:

- Hiding the **Type** column if the icon column provides enough information about the type.
- Sorting or grouping by a column without showing the column.

Select **View > Group By** or **View > Sort By**. Then, choose the method by which you want to group or sort columns.

To Modify Columns.

- To change the order, drag a column header to a new position.
- To change the width, drag the edge of the column header.

To Change the Date Format. MATLAB uses your operating system's short date format to display dates in the Current Folder browser and the Command History window. To change the date format, for instance from MM/DD/YYYY to DD/MM/YYYY, (where MM is the numerical value for the month, DD is the numerical value for the day, and YYYY is the numerical value for the year):

- 1 Change the short date format for your operating system. For instructions, see your operating system documentation.
- 2 Refresh the date display by either restarting MATLAB or doing the following:
 - In the Current Folder browser, right-click, and then choose **Refresh** from the context menu.

Dates refresh and use the new format.
 - In the Command History window, right click, and then choose **Clear Command History** from the context menu.

The window clears. MATLAB specifies new dates in the window using the new format.

Viewing File Descriptions

Show or hide descriptions in the Current Folder browser by selecting **View > ShowDescription**.

Descriptions appear in gray text beneath the name of the file or folder. When the Current Folder browser window is wide enough, descriptions display on the same line as file names. The Current Folder browser shows descriptions only for files and folders that are relevant to products from MathWorks. How the Current Folder browser gets the description depends on the type of item:

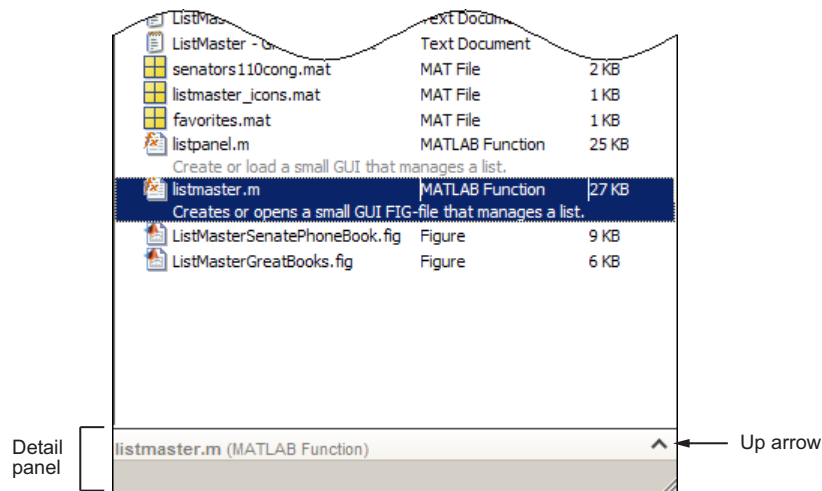
- **MATLAB program files** — The description is the first line of the help comments, known as the H1 line.

- **Simulink Models** — The description is from the Description pane of the Model Properties dialog box. Use the Current Folder browser to view model descriptions without starting the Simulink software.
- **Folders** — The description is the first comment line of the Contents.m file for the folder.

To provide descriptions for your own files and folders, see “Providing Your Own Help and Demos” on page 5-8.

Viewing File Details Without Opening Files

Display file details without opening a file by selecting the file, and then clicking the up arrow button on the lower right corner of the Current Folder browser. The details panel expands.



File and Folder Details. When you select a file or folder, the details panel displays more information about that file or folder, if possible. For example, if you select a MATLAB code file, the details panel shows the functions or subfunctions which that the file contains.

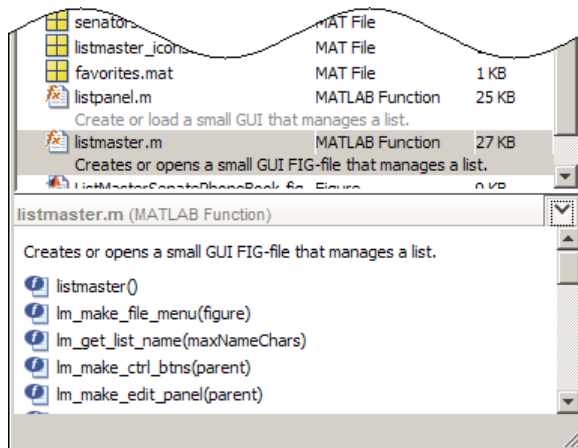
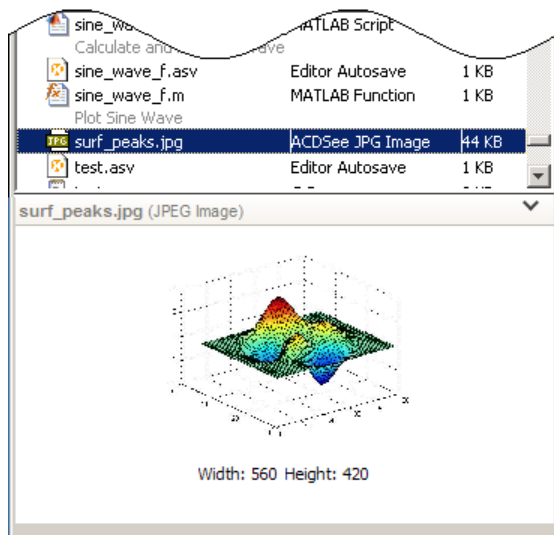
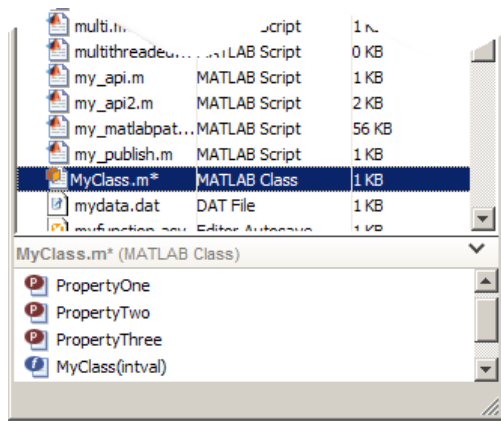


Image File Details. When you select a JPEG, JPG, BMP, WBMP, PNG, or GIF image, the details panel displays a thumbnail of the image and lists its width and height in pixels. To open the Import Wizard, double-click the thumbnail.







Viewing Unsaved File Changes. When you select a file that is currently open in the Editor and that contains unsaved changes, an asterisk (*) appears after that file name. The Current Folder browser columns reflect the content of the unsaved file. For instance, if you open a file, change it from a script to a function, and modify the H1 line, then the icon, type name, and description update in the Current Folder browser.

The preview in the details panel also reflects the unsaved file content, not the content on disk. For instance, in the following example, `PropertyTwo` exists in the modified `MyClass.m` file, but not the `MyClass.m` file on disk.



Viewing and Going to Elements within a MATLAB Program File. The details panel lists these elements when you selected a file with a `.m` extension:

- Subfunctions 
- Cells 
- Properties 
- Methods 

To open the file in the Editor, scroll to the start of the element in the details panel and double-click the element.

Viewing and Loading MAT-File Variables. Use the details panel to view the name, class, and value of all variables in the selected MAT-file. To load a variable into the workspace, select it in the details panel and drag it to the Workspace browser. The folder containing the MAT-file does not need to be on the search path for you to load it in this way.

Viewing Help for a MATLAB Program File

From the Current Folder browser, you can view help for a file that has a .m extension and is in the current folder or in a folder on the search path:

- 1 Right-click the file.
- 2 Select **View Help** from the context menu.

The reference page, if it exists, opens in the Help browser. Otherwise, help comments from the beginning of the file, if any exist, display in the Help browser.

Sorting and Grouping Files and Folders

Organize, find, and manage the files and folders you use with MATLAB by sorting and grouping items.

By default, sorting is by **Name** and grouping is off.

Regardless of the sorting and grouping options selected, the Current Folder browser lists folders and files separately.

Sorting Items. To change the order of items listed, sort by column:

- 1 Select **View > Sort By**.
- 2 Select the name of the column to sort by.

Alternatively, click the column header by which you want to sort. Click it again to reverse the direction of sorting.

Grouping Items. To see related items listed together, group them:

- 1 Select **View > Group By**.

2 Select **Type**, **Size**, or **Date Modified**.

Each group has a label. To hide the items in a group, click the collapse button (–) next to the label.

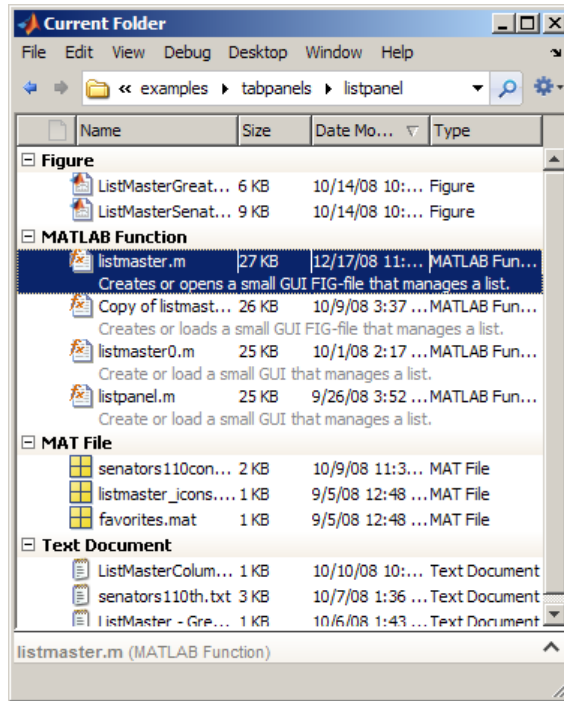
To turn off grouping, select **View > Group By > Stop Grouping**.

Using Sorting and Grouping Together. You can sort and then group, or group and then sort.

After grouping items, sort using different criteria. The sort applies to the groups and to items within each group.

The following figure illustrates grouping by type, with some groups collapsed. The sort order is descending by date

- Because the most recently modified item was a MATLAB figure, the group of figures appears at the top.
- Within the **MATLAB Function** group, the most recently modified file, `listmaster.m`, appears at the top.



Viewing Only One Type of File. To view *only* files of a certain type (for example, files having a .m extension) use a simple search. See “Simple Search for File and Folder Names in the Current Folder Browser” on page 7-27.

Finding Files and Folders

In this section...

“Finding Files and Folders by Name in the Current Folder” on page 7-27

“Simple Search for File and Folder Names in the Current Folder Browser” on page 7-27

“Advanced Search for Files — Find Files Tool” on page 7-30

“Locating a File or Folder in the Operating System Browser” on page 7-34

“Finding Files and Folders Using Functions” on page 7-35

“Additional Ways to Find Files” on page 7-35

Finding Files and Folders by Name in the Current Folder

In the Current Folder browser, use the typeahead feature to find a file or folder by name in the current folder:

- 1 Position the pointer in the list of files and folders in the current folder.
- 2 Type the first characters of the name you want to find.

As you type, the Current Folder browser searches downward from the top of the window, looking through all expanded folders. It selects the first entry in the current folder whose name begins with the characters you typed.

Typeahead and *find as you type* are other names for this feature.


Simple Search for File and Folder Names in the Current Folder Browser

Find names in the current folder and subfolders that contain a specified series of characters by using the search field in the Current Folder browser. *Instant search* and *filtering* are other names for this feature.

Steps for Using the Search Field

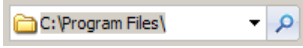
- 1 Change the current folder to the folder you want to look within.

See “Viewing and Changing the Current Folder” on page 7-4.

- 2 Perform a search by clicking within the address bar or click the search button  in the address bar.

If you clicked the search button, the path in the address bar becomes a field where you enter text, displaying the message `Type search text (ex: *.m)`. Typing in the field replaces this message with your typed characters.

If you clicked in the address bar, the current folder path is highlighted.

 , Type over or after the path and press **Enter** to change to a new folder.

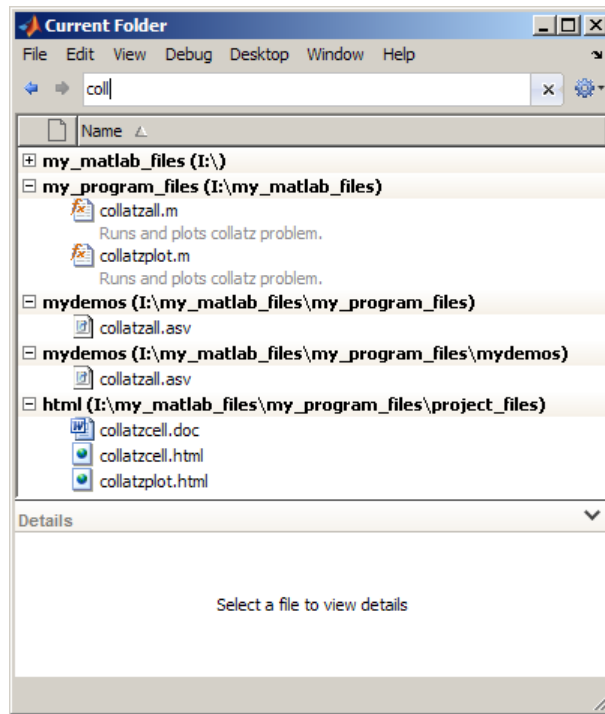
(If the address bar is not in the Current Folder browser toolbar, see “Using Toolbar Features” on page 2-110.)

You can either type a full path name or begin typing a file name. On Microsoft Windows platforms, full path names begin with two backslashes (\\) or with a drive letter followed by a colon (for example, C:). On other systems, full path names begin with a slash (/). If you type a partial path name, such as `matlab\toolbox`, it is regarded as a file name.


- 3 Ignore irrelevant characters in the string by using * (an asterisk) as the wildcard character.

As you type, the Current Folder browser lists only the names of files and folders that include the string you typed.

The following is an example of the results when you search for `coll`. The example shows results arranged by location, with the full path to the location in parenthesis.

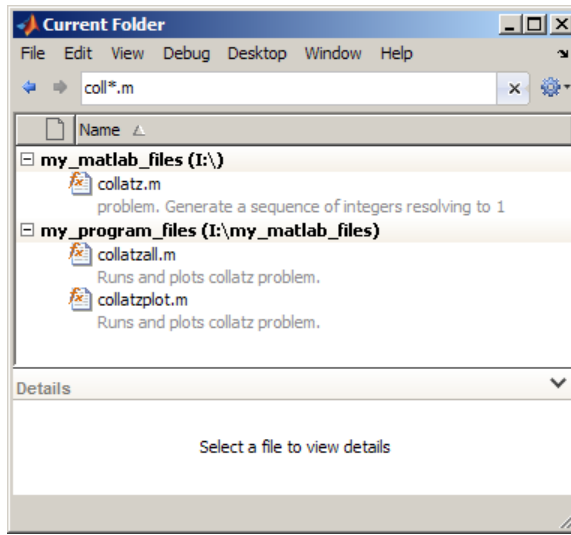



4 Press **Enter**:

- When you press **Enter** after typing a path name, the path that you typed becomes the current folder. If the folder you specified cannot be found, you receive an error dialog; after you click **OK** to dismiss it, your current folder remains unchanged.
- When you press **Enter** after typing a file name, all files within the current folder and its subfolders that match the name are shown. If no matches exist, the Current Folder browser is empty. Click the **X** button  to the right of the search field to redisplay the complete contents of the current folder.

5 Further filter the list by typing additional characters or removing characters you already typed.

Continuing the example, append *.m to show only file names that begin with coll and have a .m extension.



- 6 Customize the way search results appear by using the **View** menu options: **Show**, **Sort**, and **Group**. See “Viewing Folder Contents” on page 7-12.
- 7 Clear the filter results and show all items in the current folder by clicking the Close box  in the filter field. Alternatively, press the **Esc** key.

Advanced Search for Files — Find Files Tool

To look for a specified string in file names and within files located in multiple folders, select **File > Find Files**, which opens the Find Files tool. The following sections provide details on using the tool.

- “Steps for Using the Find Files Tool” on page 7-31
- “Opening Files from the Results List” on page 7-32
- “Accessing Previous Results” on page 7-33
- “Skipping File Types” on page 7-33

Steps for Using the Find Files Tool

- 1 Open the Find Files tool by selecting **Edit > Find Files**.
- 2 Search for file names containing a specified string by typing the string in the **Find files named** field.

Ignore irrelevant characters in the string by using an asterisk (*) as the wildcard character. For example, type `coll*` to search for file names that start with `coll`.

- 3 Search for a specified string in the content of files by typing the string in the **Find files containing text** field.

For example, search for `plot`. Alternatively, select text in the Command Window or Editor and that text appears in the field.

- For partial word searching in file contents, select **Contains text** under the **More options Search type**.
- Find an exact full-string match by selecting **Matches whole word**.

- 4 Specify file types to search for by selecting one of the options listed in the table.

One type	For Include only file type(s) , select the file type you are looking for. For example, select <code>*.m</code> to limit the search to MATLAB program files.
All types	<ol style="list-style-type: none"> a For Include only file type(s), select All files (*). b Clear the Skip file type(s) check box, under More options.
Other variations	<ol style="list-style-type: none"> a For Include only file type(s), select All files (*). b Select the Skip file type(s) check box, under More options.

- Select **Edit** to specify the file types.
See “Skipping File Types” on page 7-33.

- 5 Specify the folders to search, using one of the **Look in** options:
 - Select an option listed.
 - Enter the full path for one or more folders. Separate each path by a semicolon (;).
 - Include subfolders by selecting the **Include subdirectories** check box.
- 6 Further restrict the search using **More options**. For example, use the **Skip files over** option. It ignores large files that could take a long time to look through.
- 7 Perform the search by clicking **Find**.

The Find Files tool presents the search results in the right pane of the dialog box, with a summary at the bottom. For text searches, results include the line number and line of code.

- 8 Customize the display of results:
 - To see file locations, select **Show full pathnames**.
 - To sort results by a column, click the column heading. For example, click **Line** to sort results by line number.

Opening Files from the Results List

- 1 Select the file to open. To select multiple files:
 - Click to the left of an icon and drag up or down to select contiguous items
 - **Shift**+click to select contiguous items
 - **Ctrl**+click to select non-contiguous items
- 2 Right-click and select one of the **Open** options from the context menu.

For details about the **Open** options, see “Opening and Running Files” on page 7-45.

Accessing Previous Results

View the results of a previous search by selecting its tab at the bottom of the results pane. The Find Files tool shows up to 10 tabs for previous search results while the tool is open. File Files does not maintain the results after you close the tool.

Skipping File Types

Use the Find Files tool to look in all file types *except* file types you specify:

- 1** For **Include only file type(s)**, select **All files (*)**.
- 2** Specify the file types you want the search to ignore:
 - a** Select the **Skip file type(s)** check box.
 - b** Click **Edit**.
- 3** In the resulting Edit Skipped File Extension dialog box, specify which file types to look in and which to ignore:
 - Ignore a file type by selecting its **State** check box.
 - Look for a file type by clearing its **State** check box.
- 4** Add any file types not listed that you want to skip or look for:
 - a** Enter the file extension in the field at the top of the dialog box.
 - b** Click **Add**.

The file type appears in the list.

- c** Verify that the **State** check box has the setting you want.

The example at the end of this procedure shows the `scc` file type added.

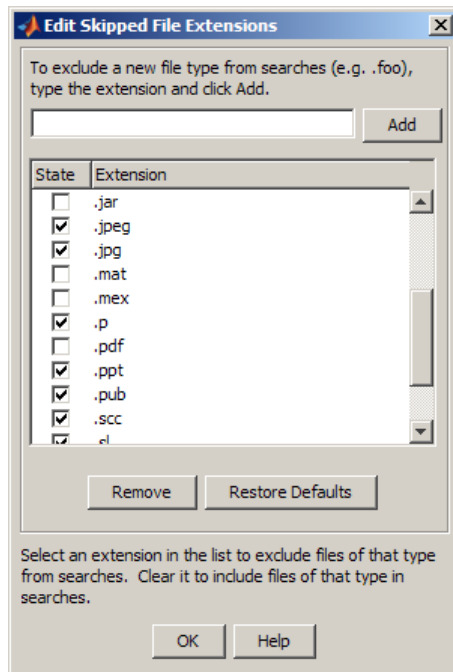
- 5** Reduce the size of the list by removing any file extensions irrelevant to your search:
 - a** Select the name of the extension.

b Click **Remove**.

6 Click **OK** to accept your changes.

The **Edit Skipped File Extensions** dialog box closes.

When you use the Find Files tool, search ignores the selected file types after making the changes.



Locating a File or Folder in the Operating System Browser

To go to a file or folder location in the Windows Explorer or the Apple Macintosh Finder, do one of the following:

- In the Current Folder browser, right-click the file or folder, and then select **Locate on Disk**.

- In the Editor, right-click a document tab, and then select **Locate on Disk**.

Document tabs appear in the Editor only when multiple documents are open and docked in the Editor.

The Windows Explorer or Macintosh Finder opens to the folder containing the selected item.

Finding Files and Folders Using Functions

To...	Use This Function
List files and folders in the current folder or in subfolders on the search path	<code>dir</code>
Determine if a variable, function, or folder exists.	<code>exist</code>
Search for the specified string in the first line of help in a MATLAB program file	<code>lookfor</code>
See files and folders that are relevant to MATLAB	<code>what</code>
See the full path to a file	<code>which</code>

Additional Ways to Find Files

- “Tools for Managing Files” on page 7-2
- “Finding Functions Using the Function Browser” on page 3-40
- “Searching the Documentation” on page 4-14
- “Finding Files in File Exchange — Searching and Using Tags” on page 8-13

Creating, Opening, Changing, and Deleting Files and Folders

In this section...

“Creating New Files and Folders” on page 7-36

“Copying, Renaming, and Deleting Files and Folders” on page 7-42

“Opening and Running Files” on page 7-45

Creating New Files and Folders

You can add files and subfolders to your current folder with the Current Folder browser or by typing commands.

- “Creating Files and Folders with the Current Folder Browser” on page 7-36
- “Creating and Updating MAT-Files with the Current Folder Browser” on page 7-37
- “Creating and Managing Zip File Archives” on page 7-38
- “Creating Files and Folders Using Functions” on page 7-41

Creating Files and Folders with the Current Folder Browser

- 1 Right-click at the location for the new file or folder. See “Locations for Storing Your Files” on page 7-7.
- 2 Select one of the following from the context menu:
 - **New Folder.**
MATLAB creates and selects a folder named `New Folder`.
 - **New File > *file-type*,**
You can select Script, Function, Class, Enumeration, or (if Simulink is installed) Model. Function, class, and enumeration files that you create this way contain template information representing the fundamental elements for the file (such as function arguments).

MATLAB creates and selects a new file named untitled with the appropriate extension.

- 3 Replace the selected name by typing a new name.

For file naming conventions, see “Function Name” and “Naming Functions”.

- 4 Press **Enter**.

Creating and Updating MAT-Files with the Current Folder Browser

To create or update a MAT-file using variables in the workspace:

- 1 In the Current Folder browser, change the current folder to the folder where you want to save the variables. See “Locations for Storing Your Files” on page 7-7.
- 2 In the Workspace browser, select a variable to save. Hold down the **Ctrl** key and click any other variable names you want to include in the MAT-file.
- 3 Drag the selected variables from the Workspace browser to the Current Folder browser.
- 4 Drop the variables in the Current Folder browser:
 - Create a MAT-file by dropping the variables onto any empty location in the Current Folder browser. Then name the file.
 - Update an existing MAT-file by dropping the variables onto the file name.

MATLAB warns you when the MAT-file contains variables of the same name. To update the existing variables, click **Continue**. Otherwise, click **Cancel**.

To suppress the warning, select

File > Preferences > General > Confirmation Dialogs, and clear the preference, **Confirm when overwriting variables in MAT-files**.

See also “Opening Files and Importing Data Using the Current Folder Browser” on page 7-45.

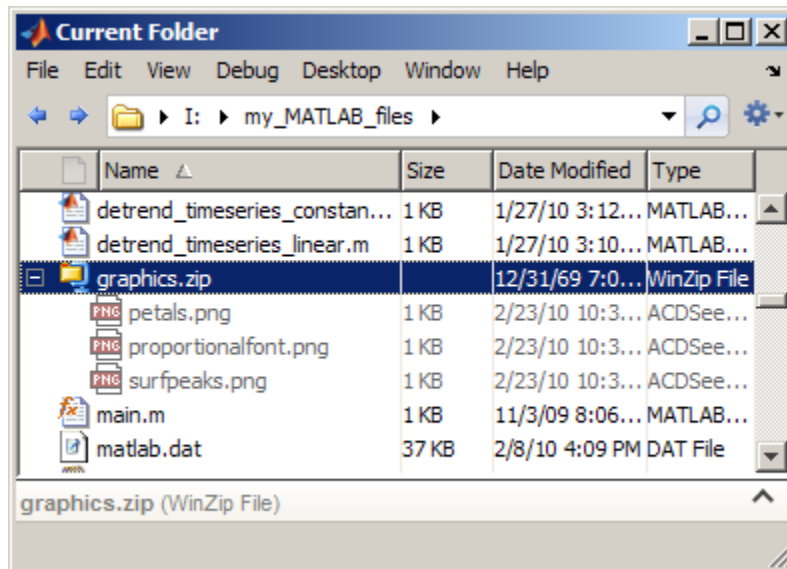
Creating and Managing Zip File Archives

To back up files, conserve file storage space, or to forward collections of files to other people, create archives using zip files. You can create, view, and adjust the contents of a zip file from within the Current Folder browser, as described in the sections that follow.

Viewing the Contents of Zip Files. To view the contents of a zip file without extracting any files it contains, click the associated + (expand) button in the Current Folder browser. This feature is helpful when you want to:

- Confirm the contents of a newly created zip file
- View the contents of a zip file before extracting files
- Selectively open certain items from a zip file

The following image shows `graphics.zip` expanded within the Current Folder browser. By default, files within a zip file appear dimmed to indicate that they are not on the MATLAB path.



Note Archives created outside of MATLAB can be encrypted or password-protected. You cannot add files to, or extract files from, protected archives from within MATLAB.

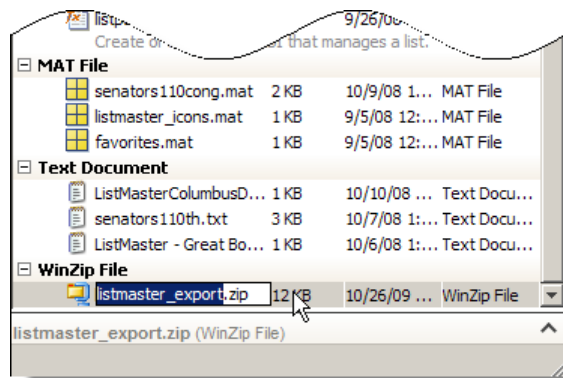
Creating Zip Archives.

1 In the Current Folder browser, select a folder or one or more files, and then right-click on any selected item.

2 From the context menu, select **Create Zip File**.

MATLAB creates an archive of the selected folder, file or files, and gives the archive a default name of `Untitledn.zip`, where n is an integer.

3 Type over the default file name to specify a descriptive name, for example `listmaster_export.zip`, as shown here.



Extracting Files from Zip Files. To extract a single file from within a zip file in the Current Folder browser, do one of the following:

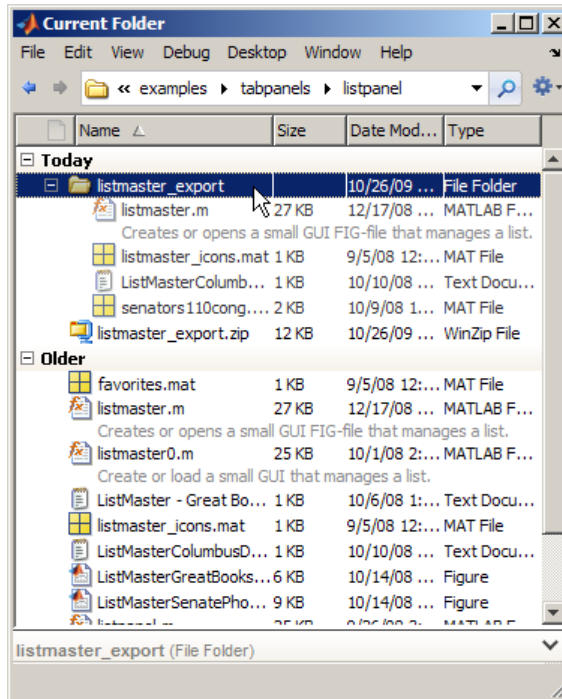
- Copy a file name and paste it into a folder in the Current Folder browser.
- Drag the file into a folder in the Current Folder browser.

MATLAB extracts the file and saves it to the folder where you dragged or pasted it.

To extract all the files from a zip file, do one of the following:

- Double-click the zip file in the Current Folder browser.
- Right-click the zip file, and then select **Extract**.

MATLAB extracts the entire contents of the zip file into a folder having the same name as the zip file, as shown here.



Because MATLAB creates a folder when extracting files, none of the extracted files can overwrite files that have the same name. If you attempt to overwrite a folder with the same name when extracting, MATLAB prompts you to determine what you want to do.

Adding Files to a Zip Archive. To add files and folders to a zip file archive in the Current Folder browser, do one of the following:

- Select, and then drag the file that you want to add onto the archive.

- Copy the file that you want to add to the archive. Then, select the archive to which you want to add the file and paste the file into the archive.

If the archive contains a file or folder with the same name as the one you are adding, a MATLAB dialog box opens. The dialog box asks if you want to replace the existing file in the archive.

Comparing the Contents of a Zip Archive to Unzipped Files and Folders. To determine differences between archived and unarchived files, use the Comparison Tool from within the Current Folder browser as you would for any other files and folders.

For instance:

- Right-click a zip archive, and then from the context menu select **Compare Against** and specify the folder to which you want to compare the contents of the zip archive.
- Expand a zip archive, right-click a file within it, and then from the context menu select **Compare Against**. Specify the file to which you want to compare the archived file.

For details, see “Comparing Files and Folders” on page 7-50.

Creating Files and Folders Using Functions

As an alternative to using the Current Folder browser to create files and folders, you can run functions in the Command Window or from a script.

To...	Use This Function
Create a folder	mkdir
Create a text file, such as a MATLAB program file	edit
Create a MAT-file	save
Create archive of files	zip, gzip, tar
Extract files from archive	unzip, gunzip, untar

See also “Locations for Storing Your Files” on page 7-7.

Copying, Renaming, and Deleting Files and Folders

- “Renaming Files Using the Current Folder Browser” on page 7-42
- “Renaming Files and Folders Using Functions” on page 7-42
- “Deleting Files and Folders Using the Current Folder Browser” on page 7-42
- “Deleting Files and Folders Using Functions” on page 7-44
- “Copying and Moving Files and Folders” on page 7-45
- “Changing Properties of Files and Folders” on page 7-45

Renaming Files Using the Current Folder Browser

- 1 Select the item to rename.
- 2 Right-click and select **Rename** from the context menu.
- 3 Type over the existing name with the new name. Warnings appear when:
 - The new name is invalid. Change the name to make it valid. See “Naming Functions”.
 - The folder is on the search path. See “Handling Errors and Unexpected Behavior When Updating Folders” on page 7-83.
- 4 Press **Enter**.

Renaming Files and Folders Using Functions

Use the `movefile` function.

Deleting Files and Folders Using the Current Folder Browser

To remove items:

- 1 Select the item to delete. To select multiple items:
 - Click to the left of an icon and drag up or down to select contiguous items
 - **Shift**+click to select contiguous items
 - **Ctrl**+click to select non-contiguous items

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

Note You cannot delete a folder while it is on the search path. See “Handling Errors and Unexpected Behavior When Updating Folders” on page 7-83.

When you delete a file or folder using the Current Folder browser, MATLAB permanently removes it or moves it to another location, based on your platform.

Platform	Behavior Deleting Files and Folders Using the Current Folder Browser
Microsoft Windows platforms	<p>Follows the Windows system preference for sending files to the Recycle Bin. Some systems only allow recycling of local files and not files accessed on a network.</p> <p>To delete a selection permanently when the system preference is set to recycle, press Shift+Delete.</p>
Linux platforms	<p>Specify the behavior:</p> <ol style="list-style-type: none"> 1 Select File > Preferences > General. 2 Set the Deleting files option you want. <p>To move files to a temporary folder, determine the location by running <code>tempdir</code>.</p> <p>To delete a selection permanently when the preference is set to recycle, press Shift+Delete.</p>
Apple Macintosh platforms	<p>Follows your Macintosh system preference for sending files to the Trash.</p>

Deleting Files and Folders Using Functions

To...	Use This Function
Delete a file	<code>delete</code>
Delete a folder	<code>rmdir</code>

You cannot recover folders deleted using `rmdir`.

By default, the `delete` function permanently deletes files. To move them to a different location instead, use the **Deleting files** preference:

- 1** From any desktop tool, select **File > Preferences > General**.
- 2** Set the **Deleting files** option you want.

Setting the preference to delete files permanently makes `delete` run faster.

To override the preference when using the `delete` function, use the `recycle` function.

The location for deleted files varies by platform, as the following table indicates.

Platform	Location for Files Not Permanently Deleted Using the <code>delete</code> Function
Microsoft Windows platforms	Recycle Bin. Some systems only allow recycling of local files and not files accessed on a network.
Linux platforms	MATLAB_Files_<day>-<mo>-<yr>_<hr>_<min>_<sec> folder in the location returned by the <code>tempdir</code> function. For example, when <code>tempdir</code> returns <code>/tmp</code> , files deleted at 2:09:28 in the afternoon of November 9, 2009 move to <code>/tmp/MATLAB_Files_09-Nov-2009_14_09_28</code> .
Apple Macintosh platforms	Trash

Deleted files remain in these locations until you remove them. To remove deleted files, use operating system features, such as **Empty Recycle Bin** on Windows platforms.

Copying and Moving Files and Folders

Copy and move files and folders using the Current Folder browser using standard GUI practices. For example, click and drag a file from one folder to another or to another application, such as Windows Explorer.

Note You cannot move a folder that is on the search path using the Current Folder browser. See “Handling Errors and Unexpected Behavior When Updating Folders” on page 7-83

To copy and move files and folders using functions, use `copyfile` and `movefile`.

Changing Properties of Files and Folders

To change some properties of files and folders, such as read/write permissions, use the `fileattrib` function.

Opening and Running Files

- “Opening Files and Importing Data Using the Current Folder Browser” on page 7-45
- “Opening Files Using the Current Folder Browser” on page 7-46
- “Opening Files Using Functions” on page 7-47
- “Running MATLAB Program Files from the Current Folder Browser” on page 7-48

Opening Files and Importing Data Using the Current Folder Browser

1 In the Current Folder browser, right-click the file you want to open or load.

2 From the context menu, select an option for opening or importing the file:

- **Open** — Opens the file using the appropriate MATLAB tool for the file type. For example, this option loads a MAT-file into the Workspace browser.
- **Open in GUIDE** — Opens a FIG-file in GUIDE instead of a figure window.

For more information, see “Opening GUIDE”.

- **Open as Text** — Opens the file in the Editor as a text file, even if the file type is associated with another application or tool.

This is useful, for example, if you have imported a tab-delimited data file (.dat) into the workspace and you find you want to add a data point. Open the file as text in the Editor, make your addition, and then save the file.

- **Open Outside MATLAB** — Opens the file using the application or tool that the operating system associates with the file type.

For example, .mat is the extension for MATLAB data files and Microsoft Access files. Whereas **Open** loads the file into the MATLAB workspace, **Open Outside MATLAB** opens the file into Microsoft Access. See .

- **Load** — Loads data from the file into the workspace using the Import Wizard. For more information, click **Help** in the wizard.

For information on how to view information about a file without opening it, see “Viewing File Details Without Opening Files” on page 7-21.

Opening Files Using the Current Folder Browser

1 In the Current Folder browser, right-click the file you want to open or load.

2 From the context menu, select an option for opening or importing the file:

- **Open** — Opens the file using the appropriate MATLAB tool for the file type. For example, this option loads a MAT-file into the Workspace browser.
- **Open in GUIDE** — Opens a FIG-file in GUIDE instead of a figure window.

For more information, see “Opening GUIDE”.

- **Open as Text** — Opens the file in the Editor as a text file, even if the file type is associated with another application or tool.

This is useful, for example, if you have imported a tab-delimited data file (.dat) into the workspace and you find you want to add a data point. Open the file as text in the Editor, make your addition, and then save the file.

- **Open Outside MATLAB** — Opens the file using the application or tool that the operating system associates with the file type.

For example, .mat is the extension for MATLAB data files and Microsoft Access files. Whereas **Open** loads the file into the MATLAB workspace, **Open Outside MATLAB** opens the file into Microsoft Access. See .

- **Load** — Loads data from the file into the workspace using the Import Wizard. For more information, click **Help** in the wizard.

For information on how to view information about a file without opening it, see “Viewing File Details Without Opening Files” on page 7-21.

Opening Files Using Functions

To...	Use This Function
Open a file or open a variable in the Variable Editor	open
Add variables from a MAT-file to the workspace	load
Import data files	importdata
Import data file using the Import Wizard	uiimport
Access the system clipboard	clipboard

See Also.


- “Importing Data”

- fileformats

Running MATLAB Program Files from the Current Folder Browser

For convenience, you can run MATLAB scripts and functions from the Current Folder browser. Script files do not accept input arguments or return values and can be run directly. If the program is a function which requires input arguments or returns output arguments, you can define a *run configuration* for it that defines arguments. See “Using Run Configurations for Functions” on page 7-48. Run any program file in the following way:

- 1 In the Current Folder browser, change the current folder to the folder containing the file to run.
- 2 Right-click the file name to open the context menu.
- 3 (Optional) If you have defined a run configuration for the file you want to use, select it from the **Run Configurations** on the context menu. Select **Edit Configurations** to edit or create one.
- 4 From the context menu, select **Run**.

If you have customized the Current Folder Browser toolbar with a **Run** button , you can select the file and then click the **Run** button. That button has a dropdown list of function run configurations you have defined. For details about customizing toolbars, see “Setting Toolbars Preferences for Desktop Tools” on page 2-156

Using Run Configurations for Functions. If you run a function that requires input arguments, executing it from the **Run** context menu or toolbar button item might not work properly. Specify default input arguments for functions that require them by defining run configurations for them. To create a run configuration:

- 1 Right-click the name of a function in the Current Folder browser and select **Run Configurations > Edit Configurations**.
- 2 Give your run configuration a name.

- 3** Type in the expressions for running the function in the MATLAB Expression panel.
- 4** Click **Run** if you want to test the configuration.
- 5** Click **Close** to save the configuration and exit the dialog box.

Executing a function with a run configuration sets up function arguments as the configuration specifies. You can create multiple configurations for a function. Your configurations are saved with your preferences. To use a run configuration:

- 1** Right-click on the function name and select **Run Configurations > *configuration name***.
- 2** The function executes according to the configuration you select and that configuration is the selected one the next time you use this context menu.

For more information about using run configurations, see “Running MATLAB Files in the Editor” on page 9-87.

Comparing Files and Folders

In this section...

- “Comparing Files and Folders” on page 7-50
- “Comparing Text Files” on page 7-52
- “Comparing Files with Autosave Version or Version on Disk” on page 7-56
- “Comparing MAT-Files” on page 7-57
- “Comparing Binary Files” on page 7-59
- “Comparing Folders and ZIP Files” on page 7-60
- “Using Features of the Comparison Tool” on page 7-63
- “Function Alternative for Comparing Files and Folders” on page 7-65

Comparing Files and Folders

- “Select Files or Folders to Compare” on page 7-50
- “Choose Comparison Type” on page 7-51
- “Explore Comparison Report” on page 7-52

Select Files or Folders to Compare

The Comparison Tool determines and displays the differences between your selected pair of files or folders. You can compare files and folders using any of these methods:

- From the Current Folder browser:
 - Select a file or folder, right-click and select **Compare Against**.
 - For two files or subfolders in the same folder, select the files or folders, right-click and select **Compare Selected Files/Folders**.
- If you have a file open in the Editor, select **Tools > Compare Against**. You can use the Editor options of **Choose**, **Autosave Version**, or **Compare Against Version on Disk**.

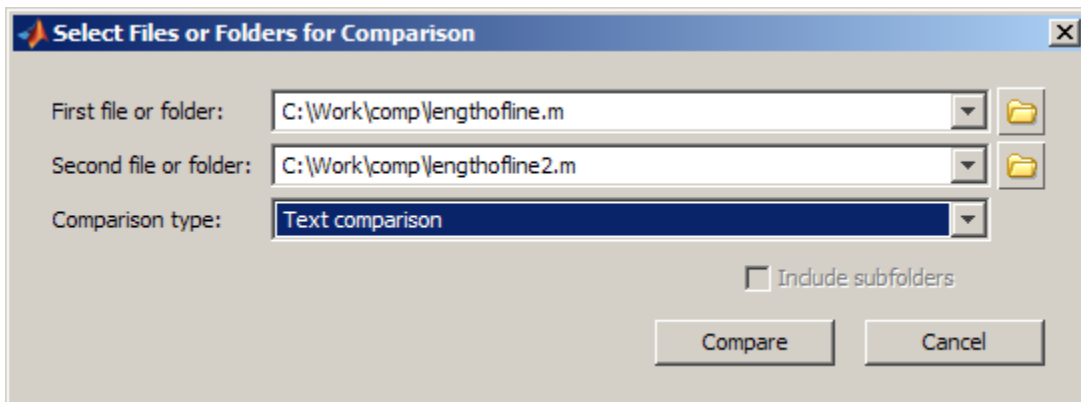
- From the MATLAB desktop, select **Desktop > Comparison Tool**. Then, select the files or folders to compare.
- At the command line, you can use the `visdiff` function.

If you use **Compare Against**, select the second item to compare in the Select Files or Folders for Comparison dialog box. Optionally, change the comparison type if multiple types are available for your selections.

Choose Comparison Type

If you specify two files or folders to compare using either the Current Folder Browser or the `visdiff` function, then the Comparison Tool automatically performs the default comparison type.

If there are multiple comparison types available for your selections, you can change what type of comparison to run, for example, text, binary, file list, or XML comparison. To change comparison type, create a new comparison from the Comparison Tool. You can change comparison type in the Select Files or Folders for Comparison dialog box.



For example, from the Current Folder browser, if you select two MAT files to compare, you get the default comparison type showing information about the variables. To change the comparison type to binary, create a new comparison from the Comparison Tool. See “Selecting Files or Folders to Compare from the Comparison Tool” on page 7-63.

Explore Comparison Report

The Comparison Tool report features depend on your comparison type. You can use the tool to:

- Compare lines in two text files (some other applications refer to this as a *file diff* operation). See “Comparing Text Files” on page 7-52.
- Compare variables in two MAT-files. See “Comparing MAT-Files” on page 7-57.
- Determine whether the contents of two binary files are the same. See “Comparing Binary Files” on page 7-59.
- Compare any combination of folders, ZIP files, or Simulink manifests:
 - To determine which file and folder names are unique to each list
 - To determine if files and folders with the same name in each list have the same contentSee “Comparing Folders and ZIP Files” on page 7-60.
- Compare XML files:
 - If you select XML files to compare and you have MATLAB® Report Generator™ software, the tool runs a hierarchical matching algorithm. You then see a report showing a hierarchical view of the portions of the two XML files that differ.
 - If you have Simulink® Report Generator™ software, you can select a pair of Simulink models (.mdl files) to compare XML files generated from them.

Comparing Text Files

- “Highlighting of Differences” on page 7-53
- “Stepping Through Differences” on page 7-55
- “Viewing a Summary of Differences” on page 7-55
- “Hide Whitespace Differences in Text Comparisons” on page 7-56
- “Increasing or Decreasing Line Lengths Shown for Text Files” on page 7-56
- “Save HTML Report” on page 7-56

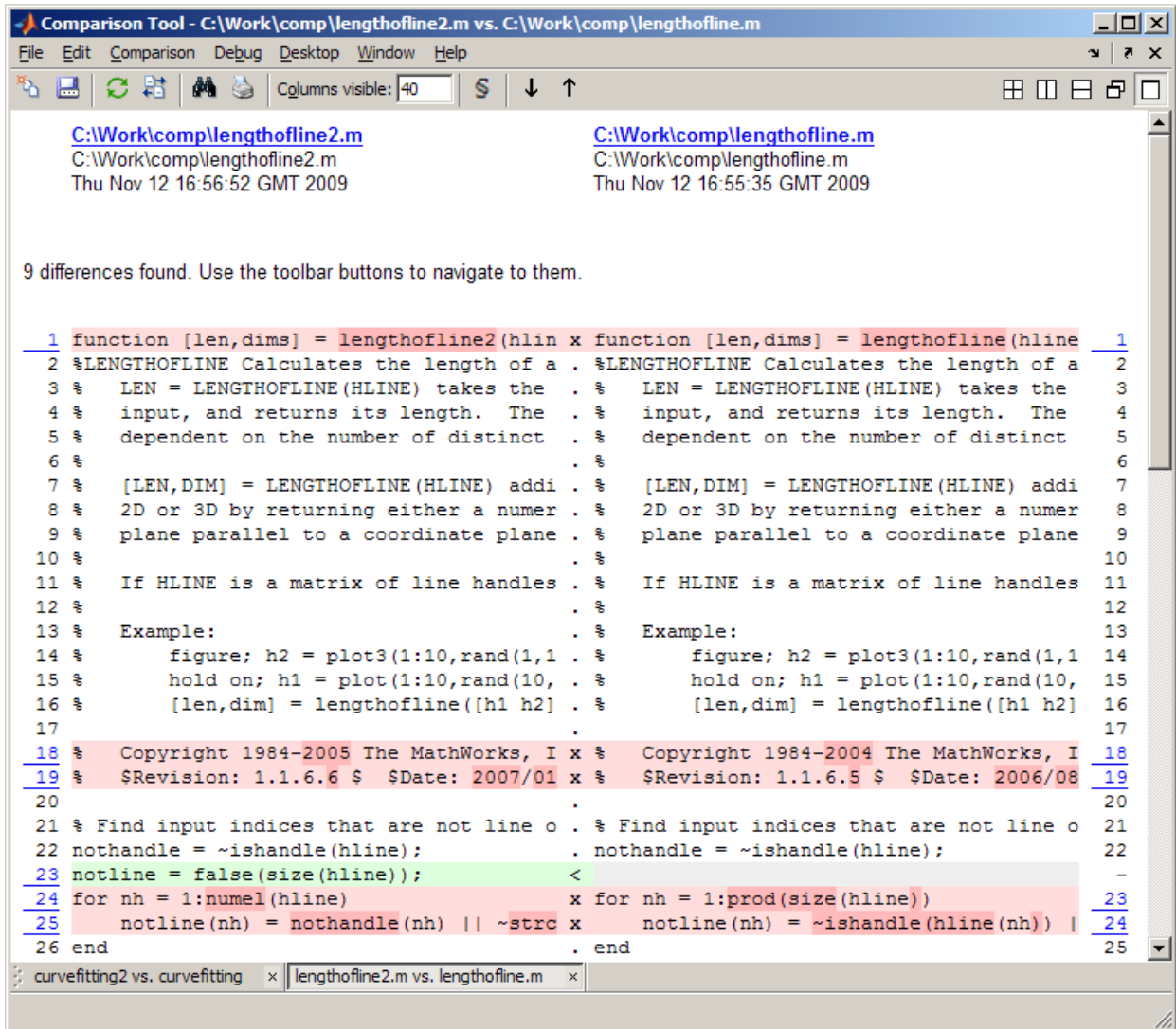
Highlighting of Differences

To select files to compare, see “Select Files or Folders to Compare” on page 7-50.

When you use the Comparison Tool to compare two text files, a window opens and presents the two files side by side. Symbols indicate how you can adjust the files to make them match. This feature can be useful when you want to compare the latest version of a text file to an autosave version.

The Comparison Tool report displays the files side by side and highlights lines that do not match, as follows:

- Dark pink highlighting indicates changed characters within lines.
- Pink highlighting and an **x** between the two sides indicate that the content of the lines differs between the two files.
- Green highlighting and a right (>) or left (<) angle bracket between the two sides indicate a line that exists on one side only.



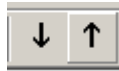
The Comparison Tool attempts to match lines and detects local text that is added, deleted, or changed. It does not do a simple line-by-line comparison. In the previous image, for example, the tool determines that `lengthofline2.m` has a line of code that does not exist in `lengthofline.m` and highlights it (line 23) in green. Also, notice that the tool takes the additional line into

account and determines that the line containing the end statement in each file matches, even though the end statement does not occur on the same line number.

If the files you are comparing are extremely long, the tool could run out of memory while attempting to perform the file comparison. In which case, the message, **Maximum file length exceeded. Defaulting to line-by-line comparison.** appears. In a line-by-line comparison, the tool highlights the lines containing the end statement because in performing this operation, it finds that the last line in one file does not match the last line in the other file.

Stepping Through Differences

Because text files can be lengthy, the Comparison Tool provides toolbar buttons to help you step through the results from one difference to the next.



To navigate through comparison results:

- Click the down arrow toolbar button to go to the next set of lines that differ.
If no additional sets of lines differ, the down arrow takes you to the end of the file.
- Click the up arrow toolbar button to go to a previous set of lines that differ.
If no previous set of lines differ, the up arrow takes you to the beginning of the file.

Alternatively, use the menu items **Comparison > Next** or **Previous**.

Viewing a Summary of Differences

To see a summary of differences between two text files, scroll to the bottom of the Comparison Tool and view the list, which contains information such as:

- Number of matching lines: 52
- Number of unmatched lines in left-hand file: 12
- Number of unmatched lines in right-hand file: 15

Hide Whitespace Differences in Text Comparisons

You may want to hide whitespace differences to help you distinguish between functional changes and changes to indentation.



Use the toolbar button to toggle the display of differences only involving whitespace characters, or select **Comparison > Ignore Whitespace**.

Increasing or Decreasing Line Lengths Shown for Text Files

To change the display width of a text comparison, edit the number in the **Columns visible** field. Resize the window, if necessary.

For details on other comparison tool features, see “Using Features of the Comparison Tool” on page 7-63.

Save HTML Report

Click Save As on the toolbar to save a copy of the report as an HTML file. The tool creates a corresponding folder containing the style sheet and JavaScript files that control the report highlighting. To preserve the highlighting, keep the folder with the HTML file.

Note Report links for opening files in MATLAB only work in the MATLAB Web browser.

Comparing Files with Autosave Version or Version on Disk

From the Editor you can compare one open text file with another, or you can choose to compare the latest version of a file in the Editor to an autosave version or a saved version. For an example, follow these steps:

- 1 Open one of the text files you want to compare in the Editor.

To open the example file provided, `lengthofline.m`, run the following command in the Command Window:

```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...  
'examples','lengthofline.m'))
```

- 2 Select **Tools > Compare Against > Browse**, or **Save and Compare Against** if your file is modified.

Navigate to the file you want to compare against, select the file, and click **Open**. To open the example file provided, select `lengthofline2.m` from the folder where you found `lengthofline.m`. Other options available are the following:

- To compare the open file to the Editor's automatic copy (*filename.asv*), select **Tools > Compare Against > Autosave Version**. For more information, see “Autosaving Files” on page 9-85.
- To compare an open file that has been changed, but not saved, to the saved version, select **Tools > Compare Against Version on Disk**

Comparing MAT-Files

To select files to compare, see “Select Files or Folders to Compare” on page 7-50.

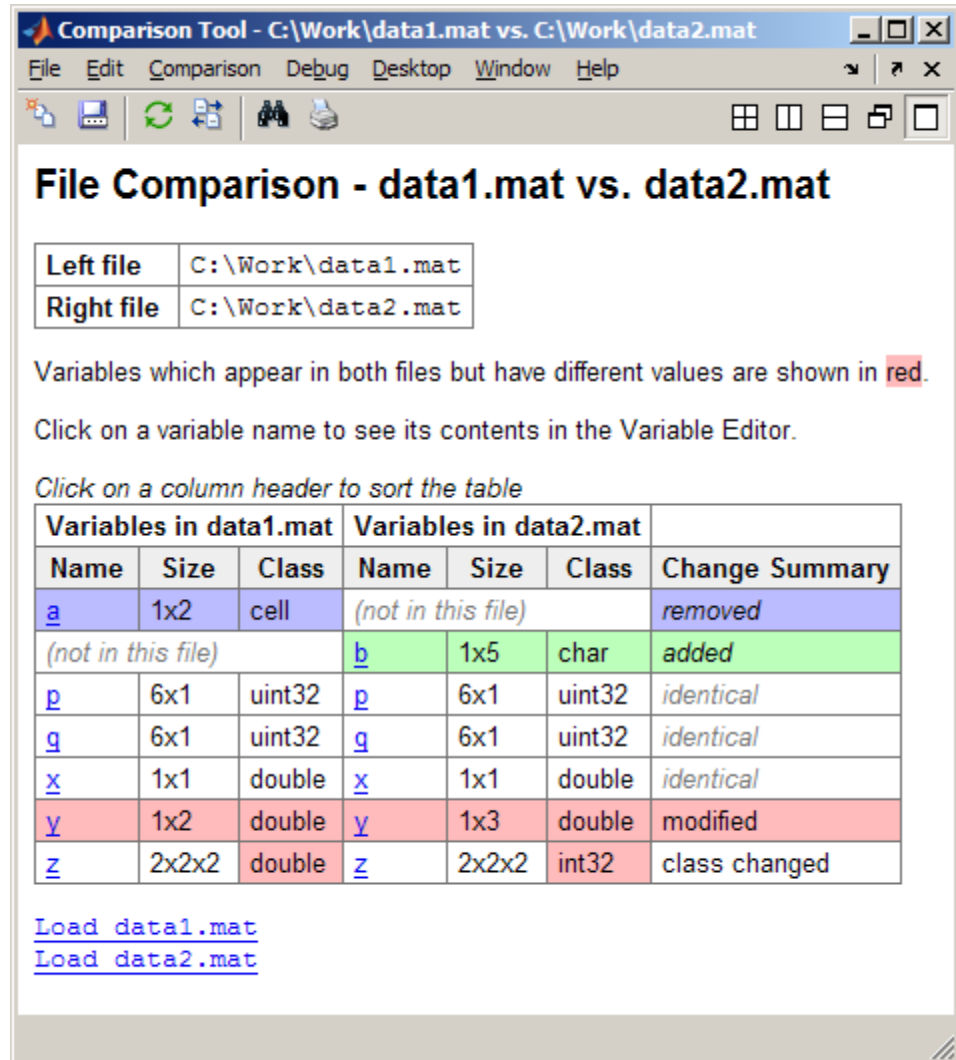
You can use the Comparison Tool to compare two MAT-files. The tool presents the variables in the two files side by side, which enables you to:

- View and sort by the name, size, class, and change summary of all variables.
- See which variables are common to each file and which are unique.
- Load the contents of the variables into the Variable Editor by clicking the name of that variable.
- Load the MAT-files into the workspace by clicking a **Load** link.
- Save a copy of the report as an HTML file. Click **Save As** on the toolbar.

The Comparison Tool highlights changes in variables as follows.

Change Summary	Highlighting	Notes
Contents changed	Pink	Values of the variable differ between the two files.
Added	Green	Variable only exists in right file.
Removed	Purple	Variable only exists in right file.
Identical	None	Variable identical in both files.
Class changed	Pink (only in Class columns)	Variable data class changed.

The following image shows the results when you compare two files `data1.mat` and `data2.mat`.



Comparing Binary Files

To select files to compare, see “Select Files or Folders to Compare” on page 7-50.

You can use the Comparison Tool to compare two binary files such as DLL files or MEX-files. Also, you can select the Binary comparison type for any pair of files with a choice of comparison types.

- If the files are the same, the tool displays the message: The files are **identical**.
- If the files differ, the tool displays the message: The files are **different**. MATLAB cannot display the differences between files of these types.

Comparing Folders and ZIP Files

- “Folder Comparison Report” on page 7-60
- “Highlighting of Differences” on page 7-61
- “Next Steps Using the Report” on page 7-62

Folder Comparison Report

To select items to compare, see “Select Files or Folders to Compare” on page 7-50. You can perform *file list comparisons* for any combinations of folders, ZIP files, and Simulink Manifests.

When you use the Comparison Tool to compare two folders (sometimes referred to as *directories*) or any file list comparison (for example, folder versus ZIP file), a window opens and presents the contents side by side. The tool enables you to:

- Determine the files that the comparison lists have in common.
- Determine if files with identical names that are common to both comparison lists also have identical content.
- Open a new comparison of two files or folders that are common to both comparison lists, but have different content.
- Open a file for viewing in the Editor.

For list comparisons, if you want to expand the list to see all files in subfolders in one report, select the **Include subfolders** check box when selecting items

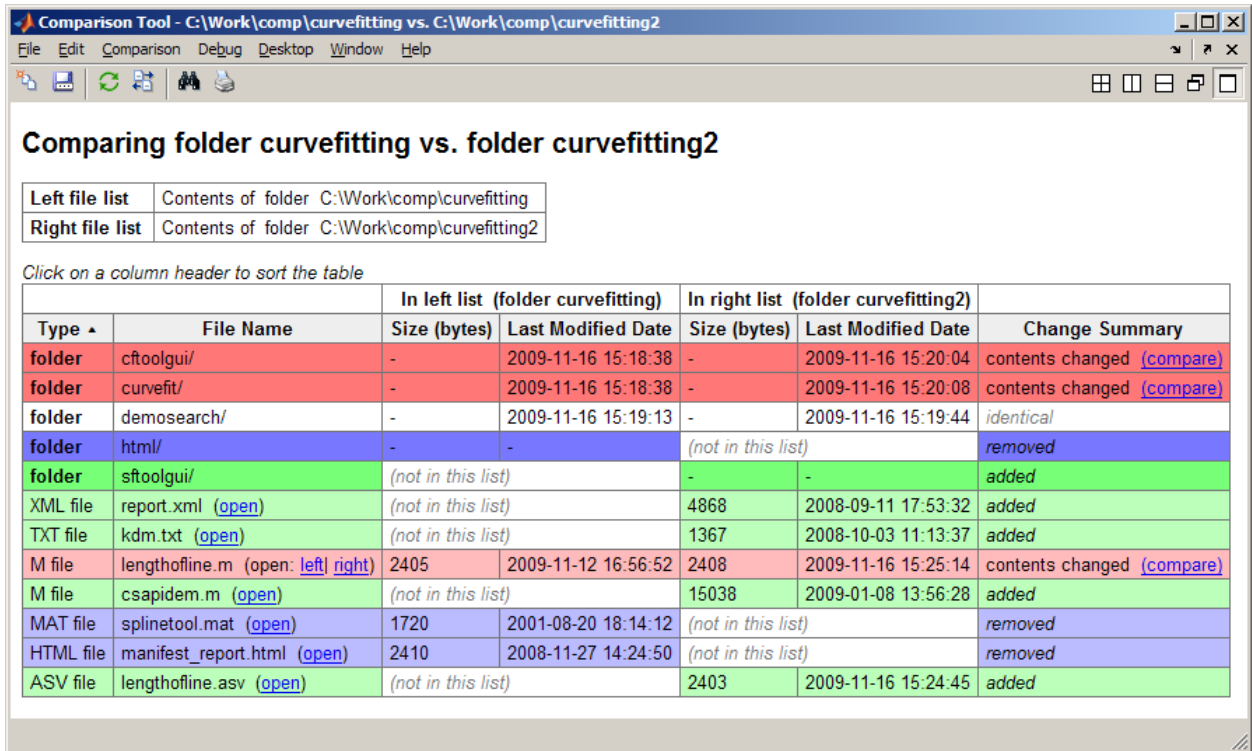
to compare. If you do not include subfolders, you can click **compare** links in the report to open a new comparison of two folders with changed content.

Highlighting of Differences

The Comparison Tool displays the contents of the lists side by side and highlights files and subfolders that do not match. The following table describes how the tool highlights each type of change. The status message (such as **identical** or **contents changed**) appears in the **Change Summary** column.

Change Summary	Highlighting for Folders	Highlighting for Files	Notes
Contents changed	Dark pink	Pink	The contents of the files or folders differ. Click the compare link to investigate.
Added	Dark green	Green	File or folder only exists in the right list.
Removed	Dark purple	Purple	File or folder only exists in the left list.
Identical	None	None	

The following image shows an example of the Comparison Tool when two folders are compared. The results are sorted by **Type**.



Next Steps Using the Report

To explore the report you can use the following tools:

- You can sort the results by name, type, size, or last modified timestamp by clicking the column headers. For example, click the **Type** column header to sort by folder and file type, as shown in the preceding figure.
- To open a new comparison of two files or folders with changed contents, click the **compare** link next to file or folder names highlighted in pink.
- To open a file in the Editor, click the **open** link next to a file name.

If the file is present in both folders, you can click links to open the **left** or **right** file.

- If subfolders are very large and contain many files, analysis continues in the background. The tool displays the number of items still to be compared at the top of the report, as shown in the next figure. You can click the links to **Skip Current** item or **Cancel All** to stop further analysis.



Items still to be compared: 12 [Skip Current](#) [Cancel All](#)

- For details on other comparison tool features, see “Using Features of the Comparison Tool” on page 7-63.

Using Features of the Comparison Tool

You can use the Comparison Tool for the following tasks:


- “Selecting Files or Folders to Compare from the Comparison Tool” on page 7-63
- “Exchanging the Left and Right Sides of the Report” on page 7-64
- “Refreshing the Report to Show Updated Files” on page 7-64
- “Finding Text” on page 7-64
- “Viewing New Comparisons” on page 7-64
- “Viewing Previous Comparisons” on page 7-65

Selecting Files or Folders to Compare from the Comparison Tool


To compare two files or folders from the Comparison Tool, follow these steps:

- 1** Select **Desktop > Comparison Tool** to open the tool.

The Comparison Tool opens a window that is empty, except for the title bar, a menu bar, and a toolbar.

- 2** Compare files or folders by clicking the New comparison button  or by selecting **File > New Comparison**.

The dialog box Select Files or Folders for Comparison appears.


- 3 In the dialog box, select two files or folders to compare. Use the drop-down lists or the **Browse** buttons  to locate and select the items that you want to compare.

You also can drag and drop a file or folder from Windows Explorer to the left and right file and folder fields.


- 4 Optionally, choose the comparison type you want to use. Either use the default **Comparison type** value, or if multiple comparison types are available, select a different one from the list. For example, for text files you could select text or binary comparison types.

- 5 Click **Compare**.


Exchanging the Left and Right Sides of the Report

To move the file or folder on the left side to the right side and vice versa, select **File > Swap Sides**, or click the Swap sides button .


Refreshing the Report to Show Updated Files

After making changes to and saving the files in the Editor, update the results in the Comparison Tool by selecting **File > Refresh** or clicking the Refresh button .

Finding Text

To find a phrase in the current display, select **Edit > Find**, or click the Find text button . The resulting Find dialog box is the same as the one you use in the Command Window. For more information, see “Finding Text Currently Displayed in the Command Window” on page 3-52.

Viewing New Comparisons

You can perform another file comparison by selecting **File > New Comparison**, or by clicking the toolbar button .

New comparisons open additional tabs in the Comparison Tool.

Viewing Previous Comparisons

You can see the results of previous comparisons in the current session by selecting that comparison's tab entry on the document bar at the bottom of the window. If you close the Comparison Tool, the current and previous comparisons are lost.

Function Alternative for Comparing Files and Folders

Use the `visdiff` function to open the Comparison Tool from the Command Window.

```
visdiff(fileorfoldername1, fileorfoldername2)
```

For example, type:

```
visdiff('lengthofline.m', 'lengthofline2.m')
```

Making Files and Folders Accessible to MATLAB

In this section...

“Files and Folders That MATLAB Can Access” on page 7-66

“How to Make Files Accessible” on page 7-66

“Determining if MATLAB Can Access a File” on page 7-68

“Ensuring MATLAB Uses the File You Want” on page 7-70

Files and Folders That MATLAB Can Access

For performance reasons, MATLAB limits where it looks for files. To run or get help for a MATLAB program, or to load a MAT-file, the file must be in one of these locations:

- The current folder in MATLAB
- A folder that is on the search path. See “What Is on the Search Path?” on page 7-72

Make the following accessible to MATLAB:

- Folders containing files that you and others create.
- Folders containing files that are *called by* files you run.
- Subfolders containing files that you run. Making a folder accessible does not make its subfolders accessible.

For files in @ (class) and + (package) folders, make the parent folder accessible. For details, see “Organizing Classes in Folders”.

How to Make Files Accessible

For files that you and others create, see “Basic Options for Making Files Accessible” on page 7-67.

To understand the differences in the basic options, and for other approaches, see “All Options for Making Files Accessible” on page 7-67.

Basic Options for Making Files Accessible

- Store the files that you and others create in the MATLAB folder, which is on the search path. See “Locations for Storing Your Files” on page 7-7.
- Change the current folder to the folder that contains the files.
- Add the folders that contain the files to the search path.

All Options for Making Files Accessible

Usage	Recommendation
You <i>seldom</i> run the file.	Change the current folder to the folder that contains the file. See “Tools for Managing Files” on page 7-2.
The file is a <i>script</i> (takes no input or output arguments).	Use the run function.
The files are in <i>one</i> folder.	Put the files in the <i>userpath</i> folder. See “Locations for Storing Your Files” on page 7-7.
The files are in <i>multiple</i> folders.	Add the folders to the search path. See “Adding Folders to the Search Path” on page 7-75. If you regularly use the files, save the changes. See “Saving Changes to the Search Path” on page 7-79.
The files call other files that are in multiple folders.	1 Determine the location of all the called files. See “Displaying Dependencies Among MATLAB Code Files” on page 10-15. 2 Add the folders to the search path. See “Adding Folders to the Search Path” on page 7-75.
Some files in multiple folders have the same name.	See “Detecting and Addressing Name Conflicts” on page 7-70.

Usage	Recommendation
You use files in different versions of MATLAB or on different platforms.	Modify the search path in a <code>startup.m</code> file. See “Using the Search Path with Different MATLAB Installations” on page 7-80.
You work with the search path content programmatically.	See the functions in the Search Path category.

Determining if MATLAB Can Access a File

The following table lists ways to determine if MATLAB has access to a file.

Option	When MATLAB Can Access the File	When MATLAB Cannot Access the File
Use the file.	Works successfully.	Produces an error. Typical error notifications include: <ul style="list-style-type: none"> • Dialog box • Message: ??? Undefined function or method 'fileName' • Message: Cannot find function 'fileName'
View the file in the Current Folder browser.	Either or both of the following are true: <ul style="list-style-type: none"> • File is in the current folder. • File does not appear dimmed in Current Folder browser, assuming the Indicate 	Either or both of the following are true: <ul style="list-style-type: none"> • File is in a subfolder of the current folder, and the subfolder is

Option	When MATLAB Can Access the File	When MATLAB Cannot Access the File
	<p>inaccessible files option is selected in “Preferences for the Current Folder Browser” on page 7-14.</p>	<p>not on the search path.</p> <ul style="list-style-type: none"> • File appears dimmed in the Current Folder browser, assuming the Indicate inaccessible files option is selected in “Preferences for the Current Folder Browser” on page 7-14.
<p>Select File > Set Path.</p>	<p>Set Path dialog box list includes the file location.</p>	<p>The list in the Set Path dialog box does not include the file location.</p>
<p>Run <code>dir</code> with no arguments.</p>	<p>The result includes the file, indicating the file is in current folder.</p>	<p>The result does not include the file.</p>
<p>Run <code>path</code>.</p>	<p>The result includes the file location, indicating the file is in a folder on search path.</p>	<p>The result does not include the file location.</p>
<p>Run <code>which filename</code>.</p>	<p>The result is the full path to the file.</p>	<p>The result is an error or a file with the same name in another location.</p>

Ensuring MATLAB Uses the File You Want

About Name Conflicts and Shadowed Files

When MATLAB has access to multiple files with the same name, these precedence rules determine the file MATLAB uses:

- MATLAB uses the file in the current folder instead of a file in a folder on the search path.
- MATLAB uses the file whose folder is closest to the top of the search path instead of a file further down.

The file that MATLAB does *not* use is called a *shadowed* file. In some cases, MATLAB warns you that a shadowed file exists.

Other name conflicts include the following:

- A file has the same name as a variable in the base workspace.
- A file has the same name as a built-in function for a MathWorks product.

When there are name conflicts, MATLAB follows these precedence rules:

- “Precedence Rules” and “File Precedence” in the MATLAB Programming Tips documentation
- “Class Precedence and MATLAB Path”

Detecting and Addressing Name Conflicts

MATLAB might not be accessing the file that you want it to when:

- You use a file and get a warning about a potential name conflict.
- You get unexpected results.

To identify a name conflict, try using the `which` function.

To address a name conflict, try one of the following:

- Change the current folder.

- Move or remove folders on the search path.
- Rename or move files.
- Specify the full path or partial path to the file that you want.
- Maintain a single version of a file instead of multiple versions.

Name conflicts can arise from using files that you create. Conflicts also can arise from using:

- Files that others create, such as from File Exchange
- A different system that has additional MathWorks products installed
- A different version of MATLAB, which could include new functions that have the same names as your existing files

See Also

- “Built-In Functions” and “Overloaded MATLAB Functions”
- rehash and “Toolbox Path Caching in the MATLAB Program” on page 1-19

Using the MATLAB Search Path

In this section...

“What Is the Search Path?” on page 7-72

“Viewing Files and Folders on the Search Path” on page 7-74

“Changing the Search Path” on page 7-75

“Using the Search Path with Different MATLAB Installations” on page 7-80

“Recovering from Problems with the Search Path” on page 7-81

“Handling Errors and Unexpected Behavior When Updating Folders” on page 7-83

What Is the Search Path?

The search path, or *path* is a subset of all the folders in the file system. MATLAB software uses the search path to locate files used with MathWorks products efficiently. MATLAB can access all files in the folders on the search path.

What Is on the Search Path?

- By default, folders provided with MATLAB and other MathWorks products. These folders are under *matlabroot/toolbox*, where *matlabroot* is the folder displayed when you type `matlabroot` in the Command Window.
- By default, the MATLAB *userpath*. See “Locations for Storing Your Files” on page 7-7.
- Folders you explicitly add to the search path for the files you and others create.

Adding folders to the search path is like performing an include or import operation in other applications.

Class, package, and `private` folders are *not* on the search path. See “Files and Folders That MATLAB Can Access” on page 7-66.

Order of Folders on the Search Path

The *order* of folders on the search path is important when two files with the same name are in folders on the search path. MATLAB uses the file nearest to the top of the search path. To customize the order of files on the search path, see “Ensuring MATLAB Uses the File You Want” on page 7-70.

Relationship Between the Search Path and the System Path

The search path is *not* the same as the system path. Furthermore, there is no explicit relationship between the MATLAB search path and the system path. However, both paths help in locating files, as follows:

- MATLAB uses the search path to locate MATLAB files efficiently.
- The operating system uses a system path to locate operating system files efficiently.

Therefore, you can issue MATLAB commands that result in the use of both the MATLAB search path and the system path. For example, if you type `dos('tasklist &')` in the MATLAB Command Window, then:

- 1 MATLAB uses the search path to locate and run `dos.m`.
- 2 The `dos` function passes `'tasklist &'` to the Microsoft Windows operating system.
- 3 Microsoft Windows uses the system path to locate and run `tasklist.exe`.

Similar behavior results when you use the MATLAB `unix` and `system` functions or the shell escape (`!`). For details on using the shell escape with MATLAB, see “Running External Programs” on page 3-8.

How MATLAB Stores the Search Path

MATLAB saves the search path information in the `pathdef.m` file. The `pathdef.m` file is a series of full path names, one for each folder on the search path, separated by a semicolon (`;`).

By default, `pathdef.m` is in `matlabroot/toolbox/local`.

When you change the search path, MATLAB uses it in the current session. To use it in future sessions, save the changes as described in “Saving Changes to the Search Path” on page 7-79.

Viewing Files and Folders on the Search Path

MATLAB provides various ways for you to view the search path, as described in the following sections:

- “Using the Current Folder Browser” on page 7-74
- “Using the Set Path Dialog Box” on page 7-74
- Using MATLAB Search Path functions

Using the Current Folder Browser

To determine if files or folders in the Current Folder browser are on the search path:

- 1 In the Current Folder browser, right-click any file or folder, and ensure there is a check mark next to **Indicate Files Not on Path**.

If there is no check mark, select **Indicate Files Not on Path**. A check mark appears.

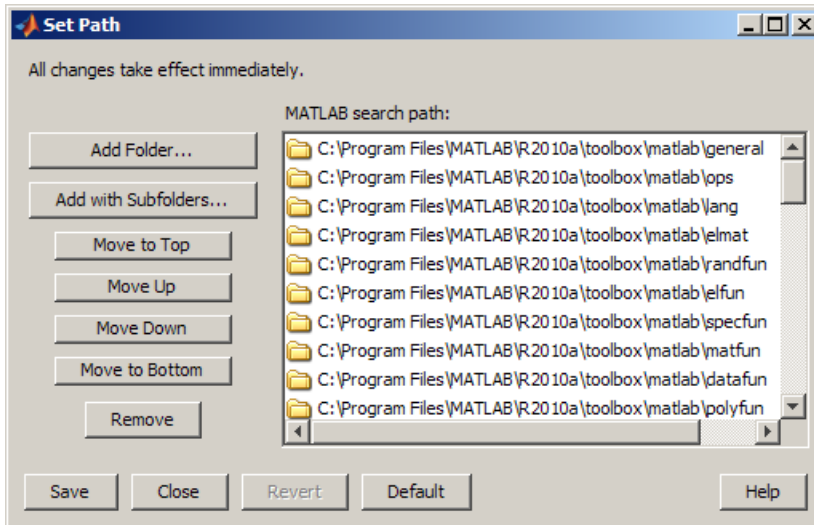
- 2 Hover the pointer over any dimmed file or folder in the Current Folder browser to find out why it is dimmed.

A tooltip opens with an explanation. Frequently, the tooltip indicates that the file or folder is not on the MATLAB path.

Using the Set Path Dialog Box

To view the entire MATLAB search path, select **File > Set Path**.

The Set Path dialog box opens, listing all folders on the search path.



Changing the Search Path

MATLAB provides various ways for you to change the search path, as described in the following sections:

- “Adding Folders to the Search Path” on page 7-75
- “Removing Folders from the Search Path” on page 7-77
- “Changing the Order of Folders on the Search Path” on page 7-78
- “Saving Changes to the Search Path” on page 7-79
- “Specifying Startup Options in the MATLABStartup File” on page 1-15
- Using MATLAB Search Path functions

Adding Folders to the Search Path

You can add folders to the search path for just the current session, or for both the current and future sessions.

Current Session Only. To add folders to the top of the search path for the duration of the current session use the following method:

- 1 From the Current Folder browser, select, and then right-click the folder or folders to add.
- 2 From the context menu, select **Add to Path**, and then select an option:
 - **Selected Folders**
 - **Selected Folders and Subfolders**

To add the folder that contains an Editor document to the top of the search path:

- 1 In the Editor, right-click the document tab.

Document tabs appear in the Editor only when multiple documents are open and docked in the Editor.

- 2 Select **Add *folder-name* to Search Path**.

To change the ordering in the search path, follow the instructions in “Changing the Order of Folders on the Search Path” on page 7-78.

Current and Future Sessions. To add folders to the search path for the current session and future sessions, use the `addpath` function, or follow these steps:

- 1 From the Current Folder browser, select **File > Set Path**.

The Set Path dialog box appear.

- 2 Click one of these buttons:
 - **Add Folder**
 - **Add with Subfolders**
- 3 In the Browse For Folder dialog box, select the folder to add to the search path, and then click **OK**.

MATLAB adds the specified folder to the top of the search path.

If you do not want the folder at the top of the search path, see “Changing the Order of Folders on the Search Path” on page 7-78.

4 Keep or cancel the search path changes:

- To use the newly modified search path only in the current session, click **Close**.
- To reuse the newly modified search path in the current session and future sessions, click **Save**, and then click **Close**.

For details on where to save the file, see “Saving Changes to the Search Path” on page 7-79.

- To undo your changes, click **Revert**, and then click **Close**.
- To restore the default search path, click **Default**, and then click **Close**. See “Restoring the Default Search Path” on page 7-80.

Removing Folders from the Search Path

You can remove folders from the search path for just the current session, or for both the current and future sessions.

For the Current Session Only. To remove one or more folders from the search path:

- 1** Select and right-click the folder or folders to remove.
- 2** From the context menu, select **Remove from Path**, and then select an option:
 - **Selected Folders**
 - **Selected Folders and Subfolders**

To remove the folder that contains an Editor document from the search path:

- 1** In the Editor, right-click the document tab.

Document tabs appear in the Editor only when multiple documents are open and docked in the Editor.

- 2** Select **Remove *folder-name* from Search Path**.

For Current and Future Sessions. To remove folders from the search path for the current and future sessions, use the `rmpath` function, or follow these steps:

- 1 Open the Set Path dialog box by selecting **File > Set Path**.
- 2 Select the folders to remove from the search path.
- 3 Click **Remove**.
- 4 Apply the changes:
 - To use the newly modified search path only in the current session, click **Close**.
 - To reuse the newly modified search path in the current session and future sessions, click **Save**.

For details on where to save the file, see “Saving Changes to the Search Path” on page 7-79.

 - To restore the default search path, click **Default**. See “Restoring the Default Search Path” on page 7-80.
- 5 Click **Close**.

Changing the Order of Folders on the Search Path

Change the order of folders in the search path when files with the same name appear in multiple folders on the search path. When you specify such a file, MATLAB uses the one found in the folder nearest to the top of the search path.

Moving Folders to Various Positions on the Search Path. To change the order of folders on the search path:

- 1 Open the Set Path dialog box by selecting **File > Set Path**.
- 2 Select the folders to move on the search path.
- 3 Click one of the Move buttons, such as **Move to Top**. The order of the folders changes.
- 4 To use the modified search path in future sessions, click **Save**.

If you do not save the changes, the newly modified search path remains in effect until you end the current MATLAB session.

5 Click Close.

Note The MATLAB (*userpath*) folder automatically moves to the top of the search path the next time you start MATLAB. See “Locations for Storing Your Files” on page 7-7.

Moving a Folder to the Top or Bottom of the Search Path. To move a folder to the top or bottom of the search path, use the path function.

Saving Changes to the Search Path

Changes you make to the search path always remain in effect during the current MATLAB session. For MATLAB to use the changed search path in future sessions, save the search path, which updates the `pathdef.m` file.

Note The MATLAB (*userpath*) folder automatically moves to the top of the search path the next time you start MATLAB. See “Locations for Storing Your Files” on page 7-7.

Ways to Save Changes. To save changes to the search path, do one of the following:

- Click **Save** in the Set Path dialog box. See “Using the Set Path Dialog Box” on page 7-74.
- Use the `savepath` function.

Where to Save the Search Path File. Save the search path to the *default* location, `matlabroot/toolbox/local`, so MATLAB can locate it.

If you do not have write access to the default location, MATLAB prompts you for a different location. Choose the MATLAB startup folder.

Restoring the Default Search Path. The default search path contains only folders provided by MathWorks.

To restore the default search path, do one of the following:

- Click **Default** in the Set Path dialog box. See “Using the Set Path Dialog Box” on page 7-74. This method also adds the *userpath* folder. See “Locations for Storing Your Files” on page 7-7.
- Use the `restoredefaultpath` function.

See also “Recovering from Problems with the Search Path” on page 7-81.

Using the Search Path with Different MATLAB Installations

Using the Search Path with Different Versions

The default search path changes for each MATLAB version because the default folders that come with the products change. Different MATLAB versions cannot use the same `pathdef.m` file.

To use your files with a new MATLAB version or with multiple versions, do one of the following:

- For each version, add the folders containing your files to the search path. Save the search path where that version of MATLAB can access it.
- Instead of changing the `pathdef.m` file, include `addpath` statements in the `startup.m` file. Use the same `startup.m` file with the multiple versions of MATLAB.

Using the Search Path with Different Platforms

To use your files with MATLAB on different platforms, include `addpath` statements in the `startup.m` file. For more information, see “Specifying Startup Options in the MATLABStartup File” on page 1-15.

Recovering from Problems with the Search Path

When there is a problem with the search path, you cannot use MATLAB successfully.

You could experience search path problems when:

- You save the search path on a Windows platform, and then try to use the same `pathdef.m` file on a Linux platform.
- The `pathdef.m` file becomes corrupt, invalid, renamed, or deleted.
- MATLAB cannot locate the `pathdef.m` file.

For example, when you start MATLAB, if a message like the following appears, it indicates a problem with the search path:

```
Warning: MATLAB did not appear to successfully set the search path...
```

To recover from problems with the search path, try the following steps. Proceed from one step to the next only as necessary.

1 Ensure MATLAB is using the `pathdef.m` file you expect:

a Run

```
which pathdef
```

b If you want MATLAB to use the `pathdef.m` file at another location, make corrections. For example, delete the incorrect `pathdef.m` file and ensure the correct `pathdef.m` file is in a location that MATLAB can access. See “Where to Save the Search Path File” on page 7-79.

2 Look for and correct problems with the `pathdef.m` and `startup.m` files:

a Open `pathdef.m` and `startup.m` in a text editor. Depending on the problem, you might not be able to open the `pathdef.m` file.

b Look for obvious problems, such as invalid characters or path names.

c Make corrections and save the files.

d Restart MATLAB to ensure that the problem does not recur.

- 3** Try to correct the problem using the Set Path dialog box:
 - a** Restore the default search path and save it. See “Using the Set Path Dialog Box” on page 7-74. Depending on the problem, you might not be able to open the dialog box.
 - b** Restart MATLAB to ensure that the problem does not recur.
- 4** Restore the default search path using functions:
 - a** Run `restoredefaultpath`, which sets the search path to the default and stores it in `matlabroot/toolbox/local`.
 - b** If `restoredefaultpath` seems to correct the problem, run `savepath`.
 - c** Restart MATLAB to ensure that the problem does not recur.

Depending on the problem, a message such as the following could appear:

```
The path may be bad. Please save your work (if desired), and quit.
```

- 5** Correct the search path problems encountered during startup:
 - a** Run

```
restoredefaultpath; matlabrc
```

Wait a few minutes until it completes.
 - b** If there is a `pathdef.m` file in the startup folder, it caused the problem. Either remove the bad `pathdef.m` file or replace it with a good `pathdef.m` file. For example, run:

```
savepath('path_to_your_startup_folder/pathdef.m')
```

See “Startup Folder for the MATLAB Program” on page 1-8.

- c** Restart MATLAB to ensure that the problem does not recur.

After correcting problems with the search path, make any changes to run your files. For example, add the `userpath` folder or other folders to the search path.

Handling Errors and Unexpected Behavior When Updating Folders

You can encounter errors or unexpected behavior when you try to delete, rename, or move folders that:

- Are on the search path
- Contain subfolders that are on the search path

The behavior varies by platform because it depends on the behavior of similar features in the operating system.

If your task fails and the error message indicates it is because the folder is on the search path, then:

- 1** Remove the folder from the search path.
- 2** Delete, rename, or move the folder.
- 3** Add the folder to the search path.

Related Topics for Managing Files

- “Comparing Files and Folders” on page 7-50
- Chapter 10, “Tuning and Managing MATLAB Code Files”
- Chapter 13, “Source Control Interface”
- Chapter 8, “File Exchange — Finding and Getting Files Created by Other Users”

File Exchange — Finding and Getting Files Created by Other Users

- “Before Using File Exchange” on page 8-2
- “How To Use the File Exchange Desktop Tool” on page 8-5
- “Finding Files in File Exchange — Searching and Using Tags” on page 8-13
- “Viewing and Sorting the List of Files in File Exchange” on page 8-28
- “Viewing Details About a File” on page 8-30
- “Downloading Files from the File Exchange Repository” on page 8-32
- “Best Practices for Using Files Provided by Other Users” on page 8-37
- “Contributing to the File Exchange Repository” on page 8-39
- “Frequently Asked Questions About File Exchange” on page 8-42

Before Using File Exchange

In this section...
“What Is File Exchange?” on page 8-2
“What You Need to Use File Exchange” on page 8-2
“Ways to Access the File Exchange Repository” on page 8-3

What Is File Exchange?

File Exchange lets you use files that were created by other users. Users have submitted thousands of files to a repository located at the MathWorks Web site, that include:

- MATLAB program files
- Simulink models
- Video demos

Use File Exchange to find a file you want and download it for use in MATLAB. Using the files saves you time, provides new ideas for your own work, and extends the set of features provided with MathWorks products.

What You Need to Use File Exchange

To access the repository, you need:

- An Internet connection

If your network uses a proxy server to access the Internet, specify the proxy server settings. For more information, see “Specifying Proxy Server Settings” on page 2-104.

- A MathWorks Account. If you do not have an account, create one when you open the File Exchange desktop tool. Use the **Create an account** link in the login window.

There is no cost to create an account or to use files in the File Exchange repository.

Ways to Access the File Exchange Repository

There are two ways to access the repository:

- File Exchange tool in the MATLAB desktop. See “How To Use the File Exchange Desktop Tool” on page 8-5.
- Web interface. Select **Help > Web Resources > MATLAB File Exchange**, or go to <http://www.mathworks.com/matlabcentral/fileexchange/>

Both the desktop tool and Web interface provide similar functionality. Use either to find, view details for, download, and provide feedback about files in the repository. There are some differences.

When to Use the Desktop Tool

Use the desktop tool when you want to:

- Work within the MATLAB desktop, as a natural part of your workflow
- Use multiple tags to find files
- Use search words and tags together to find files

When you use the File Exchange desktop tool, you can access only those files that are licensed under the BSD license. As a result:

- The desktop tool could report fewer matches than the Web interface reports.
- You might not find a file using the desktop tool that you can find using the Web interface.

When to Use the Web Interface

Use the Web interface when you want to:

- Submit files to the repository.
- Use File Exchange in a Web browser, without starting MATLAB.
- View a list of all authors, monitor changes to files with a watch list, and use other related features.
- View the complete list of files that match your criteria.

For performance reasons, the desktop tool does not list all matches at once. The desktop tool lists a maximum of 50 matches at once. You can view the other files in the desktop tool by changing your criteria.

How To Use the File Exchange Desktop Tool

In this section...

“Steps for Using File Exchange” on page 8-5

“Example — Finding and Downloading a File in File Exchange” on page 8-6

Steps for Using File Exchange

- 1 Open the tool by selecting **Desktop > File Exchange**.
- 2 In the login window, provide the e-mail address and password for your MathWorks Account.

For more information, see “What You Need to Use File Exchange” on page 8-2.

After logging in, File Exchange displays:

- The most popular tags for all files in the repository. *Tags* are keywords that users associate with files. The more frequently users apply a tag, the more popular it is.
- The 50 most recently submitted files

- 3 Find files. In general, the most efficient way to begin is by entering search words.

For more information, see “Finding Files in File Exchange — Searching and Using Tags” on page 8-13.

- 4 Refine results by selecting relevant tags, which you can see in cloud or list view. In cloud view, the font size of the tag indicates its popularity.

For more information, see “Using Tags to Find Files in File Exchange” on page 8-14.

- 5 Look for files you want to use. If you do not see any files you want, see other results by changing the sort order, the search words, or the selected tags. File Exchange lists up to 50 different files.

For more information, see “Viewing and Sorting the List of Files in File Exchange” on page 8-28.

- 6 View more details about a file by clicking the file name in the list of files. The file details page opens.

For more information, see “Viewing Details About a File” on page 8-30.

- 7 Get a file you want to use by clicking the **Download** button at the top of the file details page.

For more information, see “Downloading Files from the File Exchange Repository” on page 8-32


- 8 Use a downloaded file with MATLAB software.

If you have problems with the file, see “Best Practices for Using Files Provided by Other Users” on page 8-37.

- 9 Provide your rating and comments, and add tags to the file using the **Submit** area at the bottom of the file details page.

For more information, see “Contributing to the File Exchange Repository” on page 8-39.

- 10 When you finish using the tool, close it or log out.

If you have questions while you work, see the Frequently Asked Questions (FAQ) by clicking the Help button .

Example — Finding and Downloading a File in File Exchange

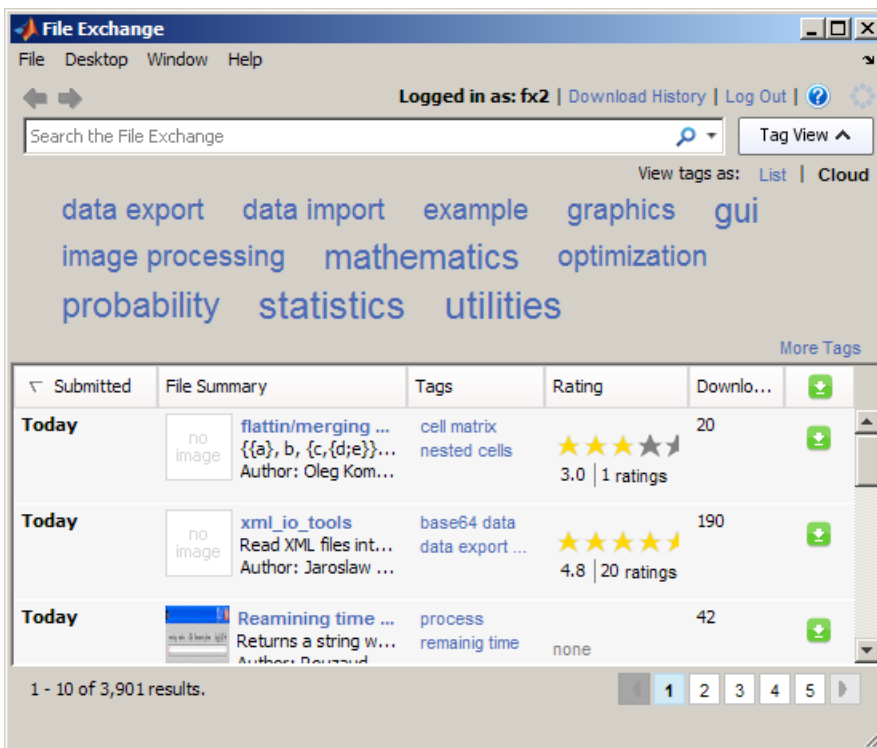
This example looks for files to help you analyze voice signals.

Note The File Exchange repository changes daily. Your results could differ from this example.

- 1 Select **Desktop > File Exchange**, and log in to your MathWorks Account.

File Exchange:

- Shows the most popular tags for *all* files in the repository. For the example, the most popular tags are data export, data import, etc.
- Lists the *50 most recently submitted* files, with 10 files per page. For the example, the most recently submitted file is flattin/merging nested cells.
- Reports the total number of files in the repository accessible to File Exchange in the desktop (have a BSD license). For the example, there are 3,901 files.



- 2 Type signal in the search field.

File Exchange:

- Reports that 277 files contain `signal` or variations of it in the title, tag, or description.
- Lists 50 of the 277 matches, with the most recently submitted, `Find and Replace`, at the top of page 1.
- Shows the most popular tags associated with the 50 files listed: `aerospace`, `automotive`, etc.

For more information, see “Using Search to Find Files in File Exchange” on page 8-13.

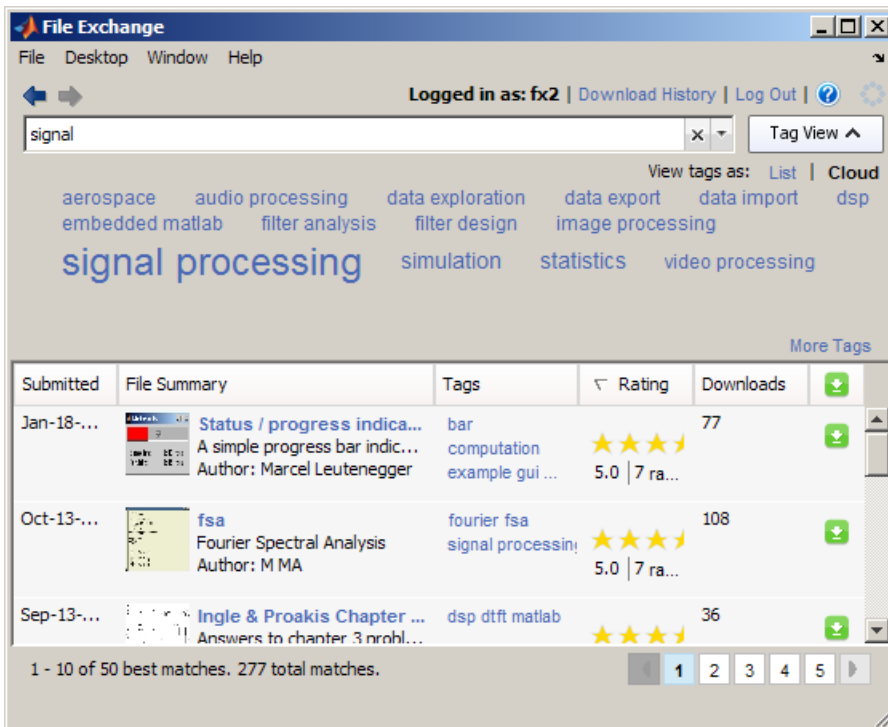


- 3 Because there are no obvious results or tags of interest, change the criteria. Change the sort order to show results that have the highest ratings first by clicking the **Rating** column header.

File Exchange:

- Lists the 50 highest rated files among the 277 matches. The most highly rated result is Status / progress indicator.
- Shows the most popular tags associated with the 50 files listed.
 - The audio processing tag is now among the most popular.
 - The automotive tag is no longer among the most popular tags for the 50 files listed.

For more information, see “Sorting the List of Files in File Exchange” on page 8-29.

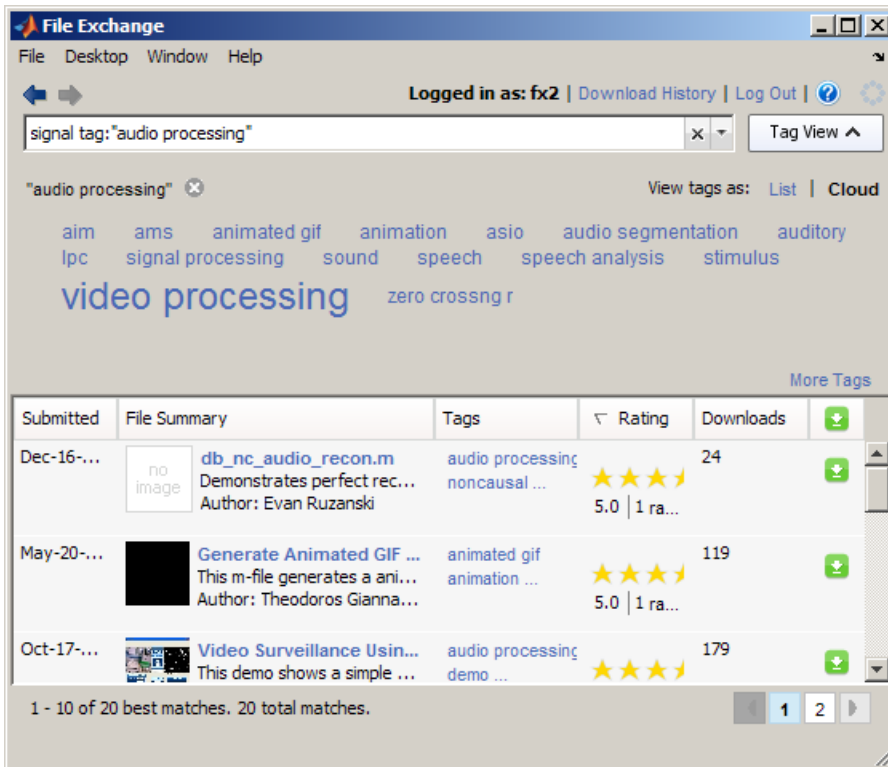


- 4 Narrow the results by clicking the audio processing tag.

File Exchange:

- Reports that 20 files are associated with the audio processing tag or variations of it, and have signal in the title, tag, or description.
- Shows the popular tags associated with the 20 files listed.

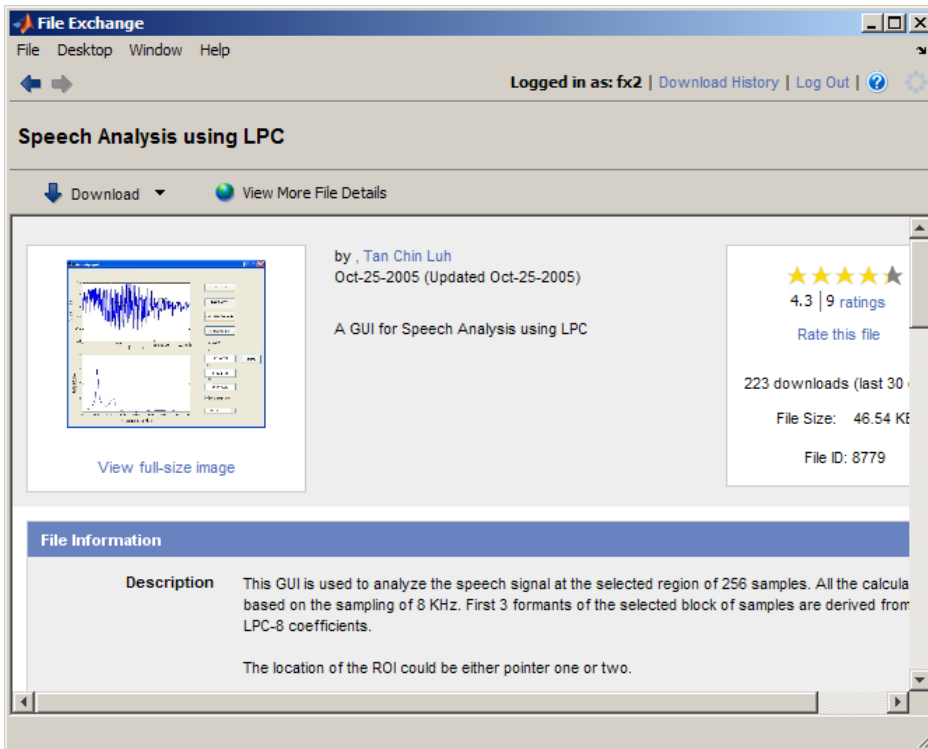
For more information, see “Finding Files Using Tags” on page 8-16.



- 5 Scroll through the list of files on the first page. **Speech Analysis using LPC** looks like it could be useful. Select the file name in the list of files.

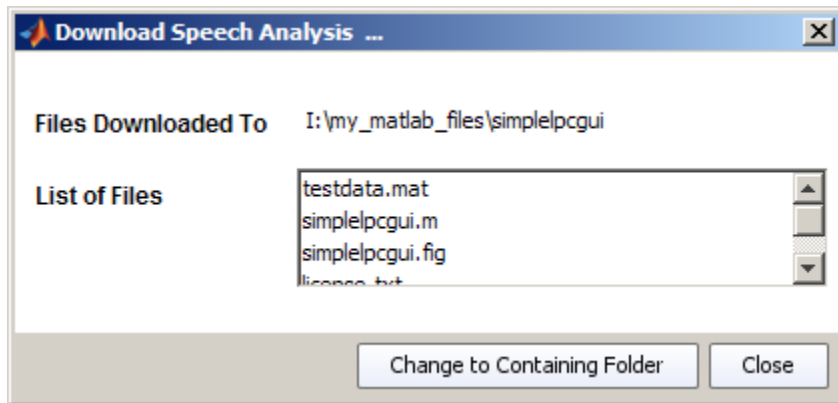
File Exchange replaces the list of files with the details page for **Speech Analysis using LPC**. On the details page, view additional information to decide if you want to use the file. For example, the details page reports that the file requires the **Data Acquisition Toolbox™** and **Signal Processing**

Toolbox products. For more information, see “Viewing Details About a File” on page 8-30.



- 6 Download the file to the current folder:
 - a In the details page, open the menu on the **Download** button.
 - b From the button menu, select **Download to Current Folder**.

For more information, see “Downloading Files from the File Exchange Repository” on page 8-32.
- 7 In the resulting confirmation dialog box, click **Download**.
- 8 When the download completes, File Exchange reports the list of files and download location. Click **Change Current Folder to Download Location** to access the files.



- 9 Go to the Current Folder browser. The current folder is `simplelpcgui` and contains the files you downloaded. Open the `simplelpcgui.m` file to review it. You can run the file. For more information, see “Best Practices for Using Files Provided by Other Users” on page 8-37.

For more options to find files, see “Example — Using Tags to Find Files in File Exchange” on page 8-18.

Finding Files in File Exchange — Searching and Using Tags

In this section...

“About Finding Files in File Exchange” on page 8-13

“Using Search to Find Files in File Exchange” on page 8-13

“Finding Files by Product, Author, and Other Attributes in File Exchange” on page 8-14

“Using Tags to Find Files in File Exchange” on page 8-14

“Clearing Your Criteria” on page 8-26

“Getting Better Results Using Search and Tags” on page 8-26

About Finding Files in File Exchange

- Find files by using search words, searching for attributes, selecting tags, and sorting.
- You can use all the methods at the same time.
- It is more efficient to search first, and then refine the search results by selecting tags and changing the sort order.
- Because the repository changes daily, criteria you use to find files now could show different results when you use the same criteria later.

Using Search to Find Files in File Exchange

- 1 Go to the list of files.
- 2 Type search words in the search field and press **Enter**.

File Exchange finds files in the repository whose titles, descriptions, or tags contain the search words you entered.

For an example, see “Example — Finding and Downloading a File in File Exchange” on page 8-6 .

Syntax for Search Words

To view guidelines for entering search words:

- 1 Click the arrow in the search field.
- 2 From the menu, select **Help Searching for Files**.
- 3 View the relevant help topic.

Finding Files by Product, Author, and Other Attributes in File Exchange

You can search for files by their attributes, for example, files whose author is Jones. For more information, click the arrow in the search field and select **Help Searching for Files**.

Using Tags to Find Files in File Exchange

- “What Are Tags?” on page 8-14
- “Ways to View Tags” on page 8-15
- “Finding Files Using Tags” on page 8-16
- “Example — Using Tags to Find Files in File Exchange” on page 8-18
- “Applying a Tag to a File” on page 8-25
- “Adding a New Tag to a File” on page 8-25

What Are Tags?

In File Exchange:

- Tags are keywords associated with a file.
- Users create tags and apply tags to files.
- Typically, a file has more than one tag applied to it.
- There is no relationship among the tags applied to a file.

Use tags to:

- Find files about a topic.
- Label files that pertain to a topic.

Ways to View Tags

- “Viewing Popular Tags for a List of Files” on page 8-15
- “Viewing More Tags for a List of Files” on page 8-15
- “To See Different Tags” on page 8-16
- “Viewing Tags for a File” on page 8-16

Viewing Popular Tags for a List of Files. Popular tags are those tags users applied most often. Multiple users can apply a tag to a file. The popularity of a tag reflects the number of times the tag was applied by all users.

File Exchange shows popular tags above the list of files:

- When the search field is empty, File Exchange shows popular tags for the entire repository.
- After you select a tag or perform a search, File Exchange shows the popular tags associated with the resulting list of up to 50 files.

You can change the way that popular tags display:

- To show or hide popular tags, click **Tag View**.
- To see additional popular tags, make the File Exchange tool wider.
- To change the view of tags, click **Cloud** or **List**:
 - **Cloud** view — The font size of a tag indicates its popularity.
 - **List** view — All tags have the same font size. You can see the popularity of a tag by the number in parentheses.

Viewing More Tags for a List of Files. When you want to see more than just the popular tags for a list of files, click **More Tags**. The resulting window:

- Allows you to choose the view, as a cloud or a list

- Shows the 250 most popular tags in the entire repository when the search field is empty
- Shows all tags associated with the list of files when the search field is not empty
- Does not show tags you already selected
- Automatically closes when you click anywhere outside of it
- Remains open if you click its top edge or drag the window by its top edge to another location

To See Different Tags. If you do not see tags of interest in popular tags or **More Tags**:

- Change the search words, remove tags, or select different tags.
- When the search field is not empty, and there are more than 50 results, you can change the sort order to see different tags.

Viewing Tags for a File. You can view tags for a file:

- In the list of files, in the **Tags** column
- On the file details page, in **Everyone's Tags**

For more information about **Everyone's Tags** and **Your Tags**, click the information button  in **Tags for This File**.

Finding Files Using Tags

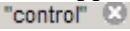
- “Selecting Tags to Find Files” on page 8-16
- “Removing Tags You Already Selected to Expand Results” on page 8-17
- “Directly Entering a Tag Name” on page 8-17

Selecting Tags to Find Files. You can select tags before or after entering search words. In general, it is more efficient to search first, and then refine the search results by selecting tags.

Select one or more tags from any of the following locations:


- Popular tags shown above the list of files
- The window that opens when you click **More Tags**
- The **Tags** column in the list of files
- **Tags for This File** on the file details page

After you select a tag:

- The tag name appears below the search field. Example of control tag selected: 
- tag: "*tagname*" appears in the search field.
- File Exchange looks for files that have variations of the selected tag associated with them.
- File Exchange reports the files that best match all your criteria.

For example, when you select the tag `control`, File Exchange finds files that have the tag `controls` `design`, `pid controller`, and other variations. The reverse is not true: if you select the tag `pid controller`, File Exchange does not find files with the tag `control`.

Removing Tags You Already Selected to Expand Results. When selecting a tag produces too few results, remove it to see more results. To remove a selected tag, do one of the following:

- Click the close button  for the tag.
- Delete tag: "*tagname*" from the search field.

Directly Entering a Tag Name. When you do not see a tag but want to use it to find files, you can directly enter the tag in the search field.

To enter the tag directly, type tag: "*partial_tagname*", and press **Enter**.

For example, if you do not see the tag `control` in popular tags or **More Tags**, enter tag: "`control`" in the search field.

For more information, click the arrow in the search field and select **Help Searching for Files**.

Example — Using Tags to Find Files in File Exchange

The example illustrates advanced aspects of using tags to find files.

You want to find files to help you compare your data, which has small sample sizes, to known distributions.

Note The File Exchange repository changes daily, so your results could differ from this example.

- 1 Open File Exchange and change the tag view by selecting **List** from **View tags as**. For more information, see “Ways to View Tags” on page 8-15.

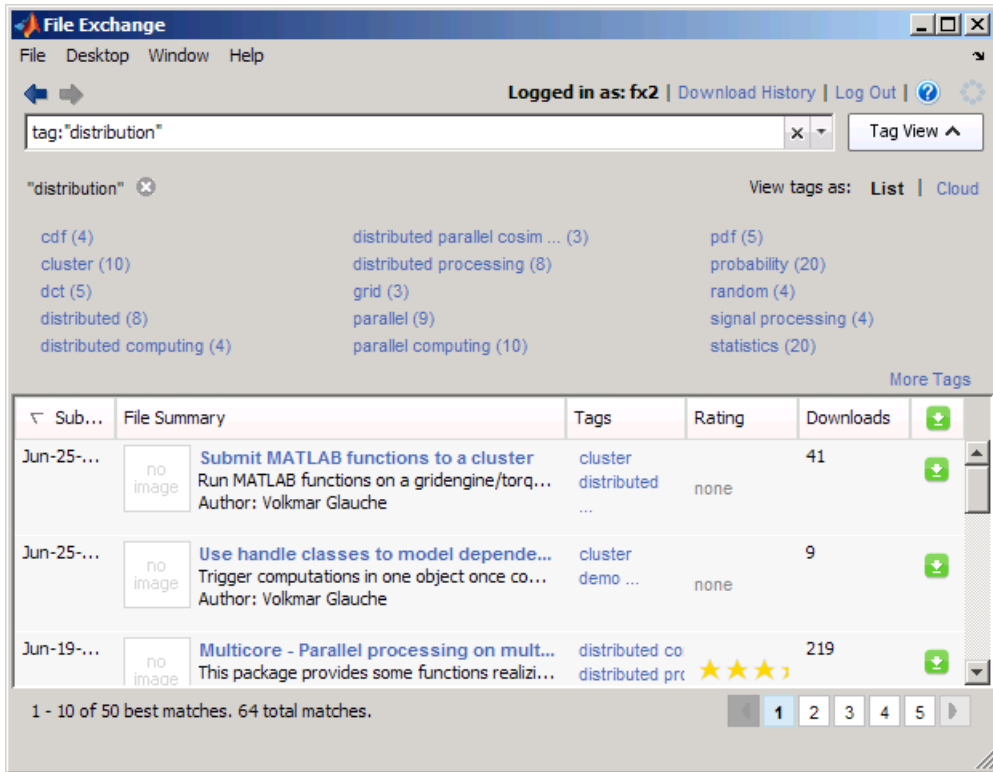
Among the list of tags, the `statistics` tag looks relevant. But because it was applied 712 times, it could be too general. You think a distribution tag could help, but do not see it.



- 2 Type tag: "distribution" in the search field, and press **Enter**. For more information, see “Directly Entering a Tag Name” on page 8-17.

File Exchange reports 64 files that have the **distribution** tag or a variation of it applied to them.

The **statistics** tag was applied 20 times among the 50 files listed. The tag could be applied additional times among the 14 matches not listed.



- 3 Narrow your results by selecting the `statistics` tag. For more information, see “Selecting Tags to Find Files” on page 8-16.

File Exchange reports 31 files that have both the `distribution` and `statistics` tags applied to them. The number of times the `statistics` tag was applied increased from 20 to 31. Before you selected the `statistics` tag, it was applied only 20 times among the 50 files listed, as described in step 2. But there were 64 results in step 2, so the `statistics` tag was also applied 11 times among the 14 results that were not listed then.

File Exchange

File Desktop Window Help

Logged in as: fx2 | Download History | Log Out | ?

tag: "distribution" tag: "statistics"

"distribution" "statistics"

View tags as: List | Cloud

cdf (3) multinomial distribution (4) rand (2)
 density (2) multivariate normal distri ... (2) random (3)
 hypothesis testing (2) normal distribution (2) sample size (2)
 mathematics (2) pdf (4) select (2)
 multinomial (3) probability (30) signal processing (4)

More Tags

Sub...	File Summary	Tags	Rating	Downloads
Jun-19-...	 probability distribution function (normal... This function calculates the probability unde... Author: Sherif Omran	gaussian disti normal distri ...	none	183
Mar-25-...	 RANDP Random integers with given probabilities (... Author: Jos	cdf density distribution ...	★ ★ ★ 5.0 3 ra...	173
Mar-11-...	 power Student Approximate power estimation of a perform... Author: Antonio Trujillo-Ortiz	hypothesis te probability ...	★ ★ ★ 3.0 4 ra...	169
Feb-19-...	 FISHERTEST Fisher Exact test for 2-x-2 contingency tables	chance distribution	none	151

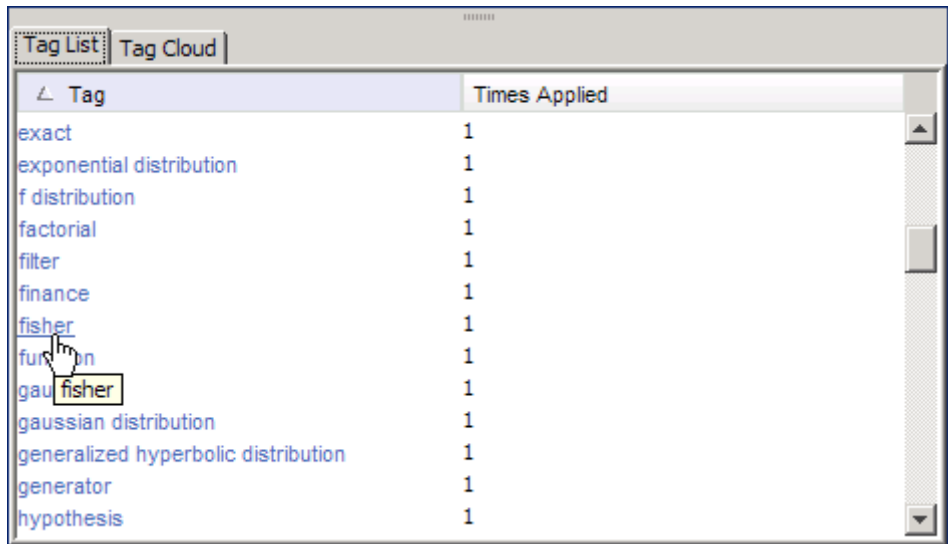
1 - 10 of 31 best matches. 31 total matches.

- 4 None of the popular tags are of interest. The FISHERTEST file is close to what you want. But the description shows it as being for a 2-by-2 contingency table and you have a 2-by-3 table.

Look for a `fisher` tag to see if there are other files for the Fisher test:

- a Select **More Tags**.
- b From the resulting window, select the `fisher` tag to add it to your criteria.

For more information, see “Viewing More Tags for a List of Files” on page 8-15.



The screenshot shows a window titled 'Tag List' with a 'Tag Cloud' tab. The window contains a table with two columns: 'Tag' and 'Times Applied'. The table lists various mathematical and statistical terms, each with a count of 1. A mouse cursor is hovering over the 'fisher' tag, and a small box highlights the word 'fisher' in the 'Tag' column.

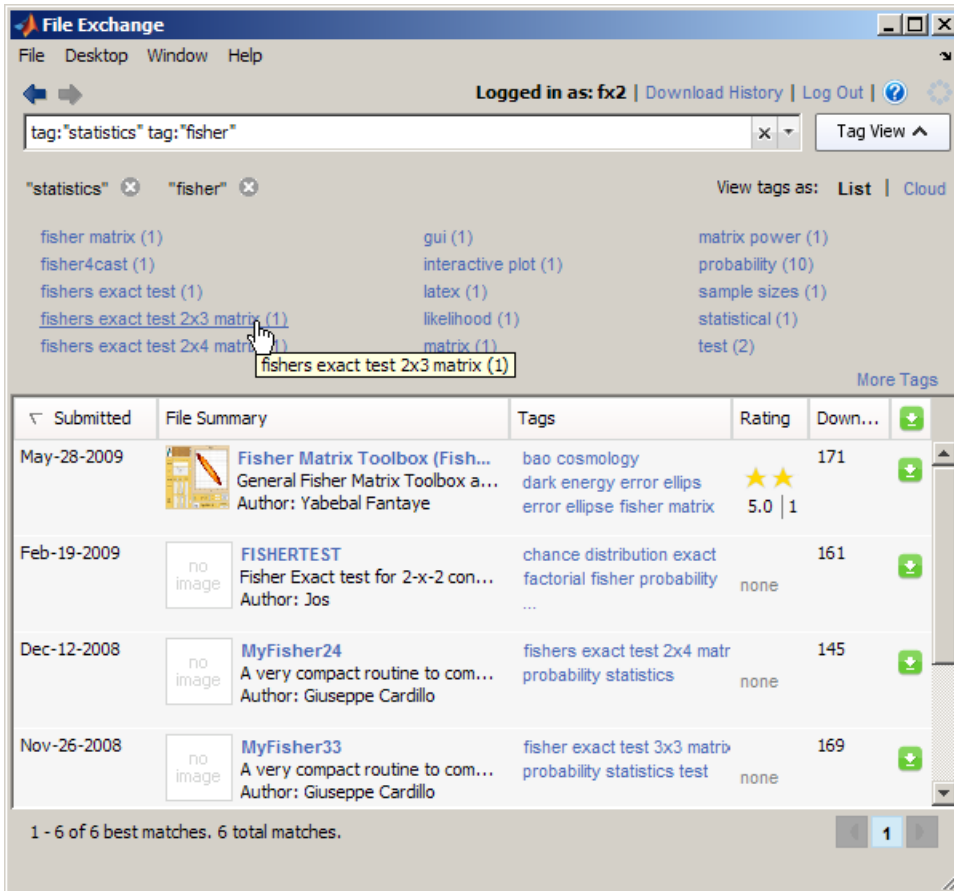
Tag	Times Applied
exact	1
exponential distribution	1
f distribution	1
factorial	1
filter	1
finance	1
fisher	1
function	1
gaussian distribution	1
generalized hyperbolic distribution	1
generator	1
hypothesis	1

- 5** File Exchange reports one file that uses all three specified tags. The file is not of interest.

Expand the results by removing a tag from your criteria. Remove the least relevant tag, *distribution*, by clicking the close button for it. For more information, see “Removing Tags You Already Selected to Expand Results” on page 8-17.



- 6 File Exchange reports six files with the tags `statistics` and `fisher` applied to them. The tags include `fishers exact test 2x3 matrix`, which is what you need. Select the tag.




- 7 One file has the three selected tags applied to it. The file is relevant to you, has a good rating, and was downloaded often. Download it by clicking the Download button.



Applying a Tag to a File

When a tag is associated with a file, you can apply the tag to the file to identify it as relevant to you.

To apply a tag to a file, go to the details page for the file and use **Everyone's Tags**.

For more information, including an example, click the **Tags for This File** information button  on the file details page.

Adding a New Tag to a File

You can add a new tag to a file. See “Adding Tags to a File” on page 8-39.

Clearing Your Criteria

Clear the search field. The default list of files displays, using the sort order you last specified. For more information, see “Viewing the Default List of Files” on page 8-28.

Getting Better Results Using Search and Tags

When too few or too many files match your criteria, try the following suggestions to get better results.

To Get More Matches	To Get Fewer Matches	Explanation
Remove tags.	Add tags.	Because results must have <i>all</i> the tags you chose, specifying fewer tags results in more matches.
Directly enter partial tag names instead of selecting tags.	Select tags or enter full tag names.	You can get more results when you type tag: " <i>partial_tagname</i> " in the search field than when you select a tag. For example, typing tag: "control" in the search field instead of selecting the tag design controller results in more matches.
Use search words instead of tags.	Use tags instead of search words.	You get more results by searching for a word than by selecting a tag of the same name. For example, typing control in the search field instead of selecting the tag control results in more matches.
Remove " " from around search words.	Add " " around search words.	Without quotation marks, you find files that have all the search words, but not necessarily in sequence.
Try again in about 15 minutes.	N/A	When you add a new tag and use it to find files, File Exchange could report no files found. It could take up to 15 minutes after adding a new tag before you can use it.

To Get More Matches	To Get Fewer Matches	Explanation
Check the search field for mistakes.		You could have inadvertently removed a meaningful space or quotation mark. For guidelines, click the arrow in the search field and select Help Searching for Files
Be sure that search attributes and values are valid.		For correct names and values, click the arrow in the search field and select Help Searching for Files .
Start over by clearing the search field.		If you want to start over instead of changing criteria you already provided, clear the search field.

If you have too many results, try the opposite of the suggestions:

- Add tags
- Use full tag names
- Use tags instead of search words

Viewing and Sorting the List of Files in File Exchange

In this section...
“Viewing the List of Files in File Exchange” on page 8-28
“Sorting the List of Files in File Exchange” on page 8-29

Viewing the List of Files in File Exchange

Viewing the Default List of Files

The default list of files:

- Is what you see when you open the File Exchange tool. The search field is empty.
- Shows the 50 files most recently submitted to the repository.

To view to the default list of files at any time:

- 1 Click the clear button in the search field.
- 2 Sort the files to show the most recently submitted files by clicking the **Submitted** column header once or twice.

Viewing the List of Files that Match Your Criteria

When you perform a search, select a tag, or change the sort order, File Exchange:

- Lists up to 50 files that best match your criteria
- Reports the total number of files in the repository that match your criteria

File Exchange does not list more than 50 results at once for performance reasons.

To see up to 50 different files, change the search words, tags, or sort order.

Sorting the List of Files in File Exchange

Use the sort features to help you find files you want:

- To sort files, select the **Submitted**, **Rating**, or **Downloads** column header. File Exchange sorts all files in the repository that match your search words and selected tags.
- To reverse the sort order, click the column header again.

Sorting applies to all files matching your criteria, not just to the files listed:

- When there are more than 50 reported matches, performing a sort usually changes the list of files. It sorts not only the files listed, but all files in the reported number of matches. It does *not* merely change the order of the files listed.
- When there are 50 or fewer matches reported, sorting merely changes the order of the files listed.

The new sort order remains in effect until you do one of the following:

- Change the sort order again
- Log out of File Exchange
- Exit MATLAB

Sorting by Number of Downloads

The File Exchange desktop tool shows the number of downloads for the last 30 days.

Viewing Details About a File

In this section...
“Viewing the File Details Page” on page 8-30
“Viewing the Contents of a File” on page 8-30

Viewing the File Details Page

To get more information about a file than what you see in the list of files, click the file name.

The file details page opens. It includes:

- Files included in the submission
- Required products
- File size
- Comments from users
- A full description, which can contain algorithms and code snippets

To return to the list of files from the file details page, click the back button .

Viewing the Contents of a File

To view the contents of a file, download it and open it.

To view the contents of a file without first downloading it:

- 1 Go to the file details page.
- 2 Click **View More File Details**.

The file details page opens on the Web interface to File Exchange.

- 3 Click **Download Now**.
- 4 Choose the option to open the file.

For submissions that include more than one file, choose the file you want to view from the unzip tool.

Downloading Files from the File Exchange Repository

In this section...

“About Downloading Files” on page 8-32

“Downloading from the List of Files” on page 8-32

“Downloading from the File Details Page to a Location You Choose” on page 8-33

“The Default Folder for Downloaded Files” on page 8-33

“Which Location Should You Choose When Downloading Files?” on page 8-33

“Downloading a Submission that Consists of Multiple Files” on page 8-34

“Viewing and Locating Files You Downloaded” on page 8-34

About Downloading Files


After finding a file of interest, get the file from the repository to use it in MATLAB.

Choose from these options when downloading files:

- Download from the list of files or from the details page
- Download to the default folder, the current folder, or another folder

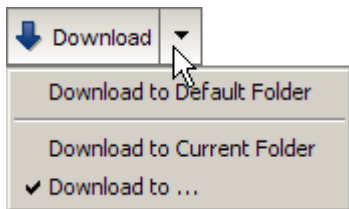
Downloading from the List of Files

The file downloads to the last download folder used. If you did not previously specify a folder, the file downloads to the default folder.

- 1 Go to the list of files.
- 2 For the file you want to download, click the download button .
- 3 Confirm or cancel the download in the resulting dialog boxes.

Downloading from the File Details Page to a Location You Choose

- 1 Go to the file details page.
- 2 Click the arrow on the **Download** button.



- 3 From the drop-down menu, select a location.
- 4 Confirm or cancel the download in the resulting dialog boxes.

To download a file to the last download location used, click **Download** on the file details page.

The Default Folder for Downloaded Files

The default location for downloads is Documents/MATLAB/Downloads.

On your system, Documents could be My Documents or something similar.

Which Location Should You Choose When Downloading Files?

To...	Use this Location
Easily find your downloaded files	Default folder
Easily run a file immediately after downloading it	Current folder in MATLAB
Use a hierarchy of folders for managing files	A folder you specify

Downloading a Submission that Consists of Multiple Files

A “file” in the File Exchange repository is a submission that can be one file or can consist of multiple related files.

For a submission containing multiple files, File Exchange:

- Downloads a single zip file containing the files.
- Creates a folder within the download folder you specified.
- Unzips the files into the folder.

Why You Might See Multiple Folders for One Download

You could see a folder within a folder for one download.

If the author submitted the files to the repository in a folder, File Exchange maintains the files in the folder. When you download, the standard zip and unzip process creates a second folder level.

You can remove one of the extra folders. If you remove an extra folder, the download history becomes inaccurate for that file.

Viewing and Locating Files You Downloaded

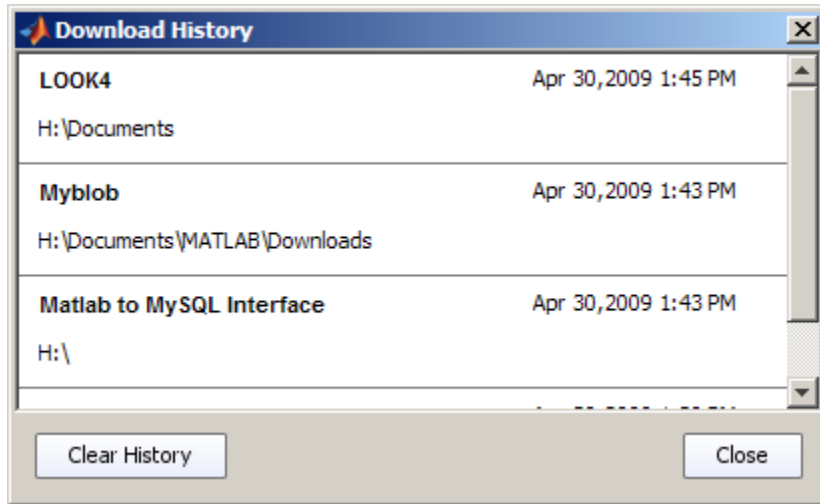
Use the download history to view and locate files you downloaded.

- “Viewing a Log of Files You Downloaded” on page 8-34
- “Locating a File You Downloaded” on page 8-35
- “Viewing Details for a File You Downloaded” on page 8-35
- “Clearing the Download History” on page 8-36

Viewing a Log of Files You Downloaded

- 1** Click the **Download History** button.
- 2** In the resulting dialog box, view the files you downloaded.

Example of download history window:



If you changed the name or location of a downloaded file, the download history does not reflect your changes.

Locating a File You Downloaded

To find a file you downloaded:

- 1 Click the **Download History** button.
- 2 In the resulting dialog box, select a file.
- 3 From the context menu, select **Change Current Folder to Download Location**.

The folder that contains the file becomes the current folder in MATLAB.

To find files easily without using the download history, always download to a single location, such as the default folder, Downloads.

Viewing Details for a File You Downloaded

To display the file details page for a file you downloaded:

- 1 Click the **Download History** button.
- 2 In the resulting dialog box, select a file.
- 3 From the context menu, select **Find in File Exchange**.

Clearing the Download History

To remove all entries in the download history dialog box:

- 1 Click the **Download History** button.
- 2 In the resulting dialog box , click the **Clear History** button.

Best Practices for Using Files Provided by Other Users

In this section...

“Ensure MATLAB Can Access the File” on page 8-37

“Consult the File Details Page” on page 8-37

“Look for Updates to the File” on page 8-37

“Read the File” on page 8-38

“Ask Questions” on page 8-38

Ensure MATLAB Can Access the File

To ensure that MATLAB can access the file, the file must be in the current folder or its folder must be on the MATLAB search path. Here is one method to use:

- 1 Always choose the default Downloads folder as the download location
- 2 Add the Downloads folder and all its subfolders to the search path.

For more information, see “Adding Folders to the Search Path” on page 7-75.

Consult the File Details Page

Review relevant information on the file details page, including:

- The description, which can include advice.
- Required products, as reported by the author.
- The release number (version) for MATLAB, as reported by the author. If you use a different release, the file might not run as expected.
- Comments provided by other users, which can include tips and workarounds.

Look for Updates to the File

A newer version of the file could address issues you have. Here are ways to find out about changes:

- View the last date the file changed on the file details page.
- View a history of changes made to the file. To see the changes, go to the file details page and click **View More File Details**. The file details page at MATLAB Central opens, and includes the log of updates.
- Get notification of changes to a file by adding the file to your watch list. To use the watch list, go to the file details page and click **View More File Details**. The file details page at MATLAB Central opens. On the page, select **Watch this File**.

When you download the new version of the file to the same location as the existing version, File Exchange:

- Replaces the existing version, if you have not changed the file.
- Changes the extension for the existing version to `.bak`, if you had changed the file.

Read the File

You can learn more about how a file works by reviewing it:

- Open the file and skim the comments and code. Look for potential function or variable name conflicts between your files and the downloaded files. For more information, see “About Name Conflicts and Shadowed Files” on page 7-70.
- Look for a `readme` or related file, which could have been provided in the download folder.

Ask Questions

If you still need help to use the file, try either of the following:

- Present your question in a comment about the file. For more information, see “Providing Comments About a File” on page 8-41.
- Contact the author. Select the author name on the file details page to display the author profile, which includes contact information.

Contributing to the File Exchange Repository

In this section...

“How You Can Contribute to the Repository” on page 8-39

“Adding Tags to a File” on page 8-39

“Removing Tags from a File” on page 8-40

“Rating a File” on page 8-40

“Providing Comments About a File” on page 8-41

“Submitting Your Files to the Repository” on page 8-41

How You Can Contribute to the Repository

File Exchange is a valuable resource because of the contributions of users like you.

You can contribute by:

- Adding tags for existing files
- Providing feedback for files you use by rating the files and adding comments
- Submitting your own files

Adding Tags to a File


1 Go to the file details page.

2 In the **Add New Tags** field near the bottom of the page, type the name of the tag to add:

- To add multiple tags at once, separate each tag with a space.
- To add a tag that has multiple words, put the words inside quotation marks. For example "pid controller".
- You can enter up to 64 characters for a tag name, including spaces. File Exchange truncates characters after the 64th.

3 Click **Submit**:

- **Your Tags** includes the tag.
- **Everyone's Tags** includes the tag. If the tag had already been applied to the file, the number of times the tag was applied to the file increases by one. The tag becomes more popular.
- It could take up to 15 minutes until you can use the newly added tag in File Exchange.

For more information, including an example, click the **Tags for This File** information button  on the file details page.

Removing Tags from a File

You have permission to remove tags you added to a file. You cannot remove tags added by other users.

To remove a tag:

- 1 Go to the file details page.
- 2 In the **Tags for This File** area, under **Your Tags**, click - (the remove button) for the tag you want to remove.

After removing a tag:

- **Your Tags** no longer includes the tag.
- If other users applied the tag to the file, the tag remains in **Everyone's Tags**. The number of times that tag was applied to the file decreases by one.

Rating a File

Assign a number of stars to rate a file:

- 1 Go to the file details page.
- 2 Go to **Rate This Submission** at the bottom of the page.
- 3 Click a star. For example, click the third star from the left to assign a rating of three stars.
- 4 Click **Submit**.

If you enter a rating but do not want to submit it, use the back button to leave the file details page.

To change a rating you already submitted, rate the file again. It would be helpful to add a comment about why you changed your rating. File Exchange maintains only your most recent rating. File Exchange reports the number of times users rated a file. When you rate a file more than once, the number of ratings includes all the times you rated a file.

It could take up to 15 minutes until your rating appears in the file list in File Exchange.

Providing Comments About a File

Provide feedback about a file you downloaded. Use comments to let the author and other users know:

- What was useful
- What was problematic
- Ways of using the file

To add comments for a file:

- 1** Go to the file details page.
- 2** Go to **Add Comments** at the bottom of the page.
- 3** Type your comments in the field.
- 4** Click **Submit**.

Submitting Your Files to the Repository

Use the Web interface to submit your own files to the File Exchange repository. Click this link to go directly to the page where you can submit your file: [MATLAB Central File Exchange — Submit New File](#)

Frequently Asked Questions About File Exchange

In this section...
“What Is File Exchange?” on page 8-42
“How Do I Use File Exchange?” on page 8-42
“How Does the File Exchange Desktop Tool Relate to File Exchange on the Web Site?” on page 8-43
“Why Do I See Only 50 Files and How Can I See More?” on page 8-43
“What Are Tags and How Do I Use Them?” on page 8-44
“What Are the Tags Above the List of Files?” on page 8-44
“How Can I See Other Tags?” on page 8-44
“Why Are the Tags Changing?” on page 8-45
“Is Search Looking Inside Files?” on page 8-45
“How Can I Start Over When Looking for Files?” on page 8-45
“How Can I Choose Where to Download a File To?” on page 8-46
“How Do I Contribute My Files to the Repository?” on page 8-46

What Is File Exchange?

- The File Exchange tool allows you to find and get files created by users of MathWorks products.
- The files are at the MATLAB Central community area of the MathWorks Web site.

For more information, see “Before Using File Exchange” on page 8-2.

How Do I Use File Exchange?

- 1 Find files you want by
 - Entering search words.
 - Selecting tags (keywords).

- Sorting the list of files.
- 2** View details for a file by clicking the file name.
 - 3** Get a file you want to use by clicking the download button.
 - 4** After using a file, rate it and provide comments.

For an overview, watch this video: [Accessing the File Exchange from the MATLAB Desktop](#).

For more information, see “How To Use the File Exchange Desktop Tool” on page 8-5.

How Does the File Exchange Desktop Tool Relate to File Exchange on the Web Site?

- The desktop tool accesses the File Exchange repository at MATLAB Central on the MathWorks Web site.
- You can also access the repository using the interface at the Web site.
- Both methods of access offer the same basic features, but there are some differences in features and results.

For more information, see “Ways to Access the File Exchange Repository” on page 8-3.

Why Do I See Only 50 Files and How Can I See More?

File Exchange lists up to 50 files at once for performance reasons:

- By default, you see the 50 most recent submissions.
- When you search, select tags, or change the sort order, you see up to 50 files that best match your criteria.
- If the list of files does not include files you want, change your criteria.

For more information, see “Viewing and Sorting the List of Files in File Exchange” on page 8-28.

What Are Tags and How Do I Use Them?

- Tags are keywords that users associate with files.
- Select tags to help you find files you are interested in.

For more information, see “Using Tags to Find Files in File Exchange” on page 8-14.

What Are the Tags Above the List of Files?

The tags shown above the list of files are the popular tags, in alphabetical order:

- When the search field is empty, you see the most popular tags for all files you can access in the File Exchange repository.
- After you perform a search or select tags, you see the most popular tags associated with the resulting list of files.

You can view popular tags as a cloud or list:

- In cloud view, the font size of the tag name indicates how popular the tag is.
- In list view, the number of times the tag has been applied appears in parentheses, numerically indicating its popularity.

For more information, see “Viewing Popular Tags for a List of Files” on page 8-15.

How Can I See Other Tags?

- Make the File Exchange tool wider, which could show additional popular tags.
- Change the search words, selected tags, or sort order to show different popular tags.
- Click **More Tags** to view more tags in a separate window.

For more information, see “Ways to View Tags” on page 8-15.

Why Are the Tags Changing?

When you select a tag, perform a search, or change the sort order, File Exchange looks through the repository for all files that match your criteria. The list of files changes to show up to 50 files that match your criteria. Because popular tags reflect the resulting list of files:

- The popular tags shown above the list of files change.
- The popularity of each tag changes:
 - In cloud view, the font size changes.
 - In list view, the number in parentheses changes.

For more information, see “Using Tags to Find Files in File Exchange” on page 8-14.

Is Search Looking Inside Files?

File Exchange search does not look inside files at the code or comments.

File Exchange looks for matching words in information associated with files. Search looks in:

- Titles
- Descriptions
- Tags

For more information about searching, click the arrow in the search field, and select **Help Searching for Files**.

How Can I Start Over When Looking for Files?

Clear the search field to return to the default list of files. For more information, see “Viewing the Default List of Files” on page 8-28.

How Can I Choose Where to Download a File To?

- To specify the folder for your downloaded files, use the file details page to perform the download.
- When you download from the list of files, the file downloads to the last location you downloaded to.

For more information, see “Downloading Files from the File Exchange Repository” on page 8-32.

How Do I Contribute My Files to the Repository?

To submit files you created to the File Exchange repository, use the Web interface, MATLAB Central File Exchange — Submit New File.

For more information, see “Contributing to the File Exchange Repository” on page 8-39.

Editing and Debugging MATLAB Code

MATLAB software provides powerful tools for creating, editing, and debugging MATLAB code, as detailed here. For information about the MATLAB language and writing MATLAB code files, see the Programming Fundamentals documentation.

- “MATLAB Code Files” on page 9-2
- “Ways to Edit, Evaluate, and Debug Code” on page 9-4
- “Starting, Creating Files, and Closing the Editor” on page 9-6
- “Customizing the Editor by Setting Preferences” on page 9-13
- “Entering Statements in the Editor” on page 9-38
- “Making MATLAB Code Files More Readable” on page 9-53
- “Navigating an Open File in the Editor” on page 9-71
- “Finding Text in Files” on page 9-78
- “Saving, Printing, and Closing Files in the Editor” on page 9-83
- “Running MATLAB Files in the Editor” on page 9-87
- “Finding Errors, Debugging, and Correcting MATLAB Files” on page 9-104
- “Preventing and Identifying Coding Problems” on page 9-107
- “Debugging Process and Features” on page 9-141
- “Evaluating Subsections of Files Using Code Cells” on page 9-175
- “Debugging Functions” on page 9-202

MATLAB Code Files

In this section...
“What Are MATLAB Code Files?” on page 9-2
“Creating Files from the Command Window and Command History” on page 9-2
“Use Existing MATLAB Code and Examples” on page 9-2

What Are MATLAB Code Files?

MATLAB code files are files with a `.m` extension that contain MATLAB statements and functions. The Editor provides tools for creating and debugging these files.

Creating Files from the Command Window and Command History

Before you begin writing MATLAB code, consider starting with existing code, and then use the MATLAB Editor to modify that code.

You can create and run MATLAB statements in the Command Window, modify those statements to your satisfaction, and then create a file that includes the statements. To facilitate this process, in the Command History, select the MATLAB statements you want to include in the file. Right-click and select **Create Script**. The Editor opens a new file that includes the statements you selected from the Command History. You can also copy the statements from the Command History and paste them into an existing file.

Use Existing MATLAB Code and Examples

See if you can find existing MATLAB code that achieves results like what you want. Then, copy and use that code in your own file, assuming that you have legal permission to do so. The sections that follow provide some resources you can use.


MATLAB and Toolbox Functions

You can access and reuse the code in most MATLAB and toolbox functions that have a `.m` file extension. You cannot use built-in MATLAB or built-in toolbox functions. They are efficient, but their code is not accessible.

If a MATLAB function accomplishes results like what you want, and it is not built in, open the function file in the Editor and use it as a basis for your file. Be sure to save the file using a different name, but with a `.m` file extension in a folder that is not in `matlabroot/toolbox`. For details, see “Saving Files” on page 9-83.

Demos and Examples

The MATLAB product and its toolboxes include demonstration programs. You can view the code in the demos and copy it for use in your own files. To see the demos, type `demo`, which opens the Help browser to the **Demos** pane. For more information about demos, see “Learning from Demos” on page 4-25.

There are also code examples in the online documentation. To see a list of examples for a product, type `helpbrowser` to open the Help browser. In the **Contents** pane, click **+** for a product to view the help topics, and then select the  **Examples** entry.

File Exchange

File Exchange enables you to use files created by others. Before creating a file, consider seeing if someone has provided a file that achieves results like what you need. This practice can save time and provide new ideas. In addition, consider submitting files you create for others to use. For more information, see Chapter 8, “File Exchange — Finding and Getting Files Created by Other Users”.

Ways to Edit, Evaluate, and Debug Code

There are several methods for creating, editing, evaluating, and debugging files with MATLAB software.

Creating and Editing Files – Options	Instructions
MATLAB Editor	For typical, interactive use of the Editor, see: <ul style="list-style-type: none"> • “Starting, Creating Files, and Closing the Editor” on page 9-6 • “Entering Statements in the Editor” on page 9-38 • “Navigating an Open File in the Editor” on page 9-71 • “Saving, Printing, and Closing Files in the Editor” on page 9-83
Any text editor, such as Emacs or vi	To specify another editor as the default for use with MATLAB software, select File > Preferences > Editor/Debugger , and for Editor , specify the Text editor . Click the Help button in the Preferences dialog box for details. Use that editor by default, or use any other editor you open. Regardless of the editor you use by default, you can debug MATLAB code files using the MATLAB Editor or debugging functions.
Debugging MATLAB Code – Options	Instructions
General debugging tips	See “Finding Errors, Debugging, and Correcting MATLAB Files” on page 9-104.
MATLAB Editor	See the following: <ul style="list-style-type: none"> • “Preventing and Identifying Coding Problems” on page 9-107 to identify errors and make improvements • “Debugging Process and Features” on page 9-141 to help you isolate run-time problems

Debugging MATLAB Code - Options	Instructions
MATLAB debugging functions (for use in the Command Window)	See function alternatives in “Debugging Process and Features” on page 9-141.

See “Customizing the Editor by Setting Preferences” on page 9-13 for information on setting up the editing and debugging environment to meet your needs.

For information about the MATLAB language and writing functions and scripts, see the Programming Fundamentals documentation.

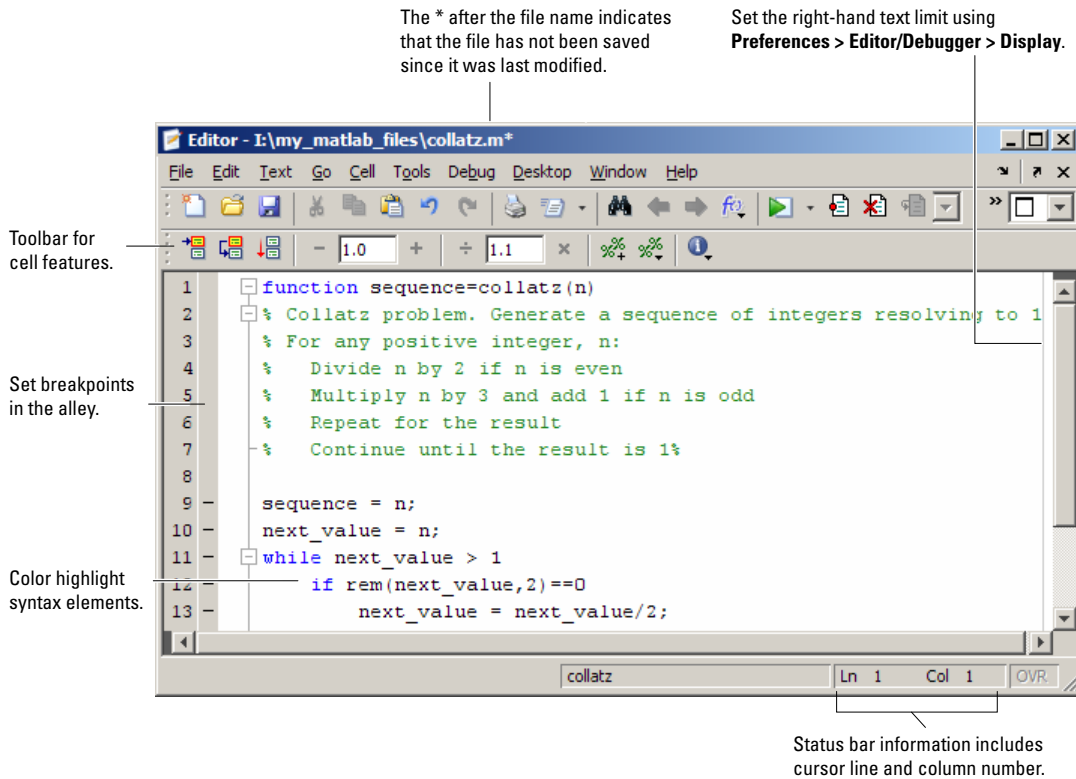
Starting, Creating Files, and Closing the Editor

In this section...
“Starting the Editor” on page 9-6
“Creating New Files in the Editor” on page 9-7
“Opening Existing Files Using the Editor” on page 9-9
“Arranging Editor Documents” on page 9-11
“Closing the Editor” on page 9-11

Starting the Editor

The MATLAB Editor provides a graphical user interface for basic text editing features for any file type, as well as for debugging MATLAB code files. The Editor is a single tool that you can use for editing, debugging, or both. There are various ways to start the Editor. The Editor automatically starts when you open a document or create a new one. Once started, you can customize the Editor to suit your needs.

This figure shows an example of the Editor opened outside of the desktop containing an existing file.




Creating New Files in the Editor

You can edit any type of text file using the MATLAB Editor. For example, you can open and edit an HTML file. However, you can run or debug only MATLAB code files from the Editor.

In all cases, the Editor opens an untitled file in the MATLAB current folder.

With the Editor as the current (active) window, you can:

- Create an empty file for MATLAB code or for a text file (such as for an XML or HTML file).

Select **File > New > Script** or click the New Script button  on the MATLAB desktop toolbar.

- Create a file prepopulated with basic MATLAB function elements.
Select **File > New > Function**.
- Create a class definition file (`classdef`), prepopulated with basic class structure elements.
Select **File > New > Class**.
- Create an enumeration class file, prepopulated with enumeration class elements.
Select **File > New > Enumeration**.

When working with MATLAB, C/C++, Java, TLC, VHDL, and Verilog programming languages, as well as XML or HTML, you can specify syntax highlighting. In addition, for some languages, you can specify indenting preferences. Select **File > Preferences > Editor/Debugger > Language**. For details, click the **Help** button in the dialog box.

Other tools also provide features for creating new files. For details, see:

- “Performing Actions on Statements in the Command History Window” on page 3-68
- “Creating New Files and Folders” on page 7-36

Function Alternative for Creating New Files


You can do either of the following:

- Create a blank unnamed file in the Editor by typing `edit` in the Command Window.
- Create a blank file named `filename.ext` in the Editor by typing `edit filename.ext`. If `filename.ext` exists in the current folder or on the MATLAB search path, MATLAB opens the existing file. If `filename.ext` does not exist in the current folder or on the MATLAB search path, by default, a confirmation dialog box appears. It asks if you want to create the named file.

For more information about the confirmation dialog box, see “Confirmation Dialogs Preferences” on page 2-132.

Opening Existing Files Using the Editor

To open an existing file or files using the Editor, choose the option that achieves your goals, as described in this table.

Goal	Steps	Additional Information
<p>Open with associated tool</p> <p>Open a file using the appropriate MATLAB tool for the file type.</p>	<p>Select File > Open or click the Open file button  on the toolbar.</p>	<p>For example, this option opens a file with a <code>.m</code> extension in the Editor and loads a MAT-file into the Workspace browser.</p>
<p>Open as text file</p> <p>Open a file in the Editor as a text file, even if the file type is associated with another application or tool.</p>	<p>Select File > Open as Text.</p>	<p>This is useful, for example, if you have imported a tab-delimited data file (<code>.dat</code>) into the workspace and you find you want to add a data point. Open the file as text in the Editor, make your addition, and then save the file.</p>
<p>Open function from within file</p> <p>Open a subfunction or function file from within a file in the Editor.</p>	<p>Position the cursor on the name within the open file, and then right-click and select Open <i>file-name</i> from the context menu.</p>	<p>You also can use this method to open a variable or Simulink model.</p> <p>For details, see “Opening a File or Variable from Within a File” on page 9-76.</p>

Goal	Steps	Additional Information
<p>Reopen file</p> <p>Reopen a recently used file.</p>	<p>At the bottom of the File menu, select a file.</p>	<p>For information on changing the number of files on the list, see “Setting General Preferences for the Editor/Debugger” on page 9-15.</p>
<p>Reopen files at startup</p> <p>At startup, automatically open the files that were open when the previous MATLAB session ended.</p>	<p>Select File > Preferences > Editor/Debugger, and then select On restart reopen files from previous MATLAB session.</p>	<p>For details, see “Setting General Preferences for the Editor/Debugger” on page 9-15.</p>
<p>Open file displaying in another tool</p> <p>Open a file name displaying in another MATLAB desktop tool or Microsoft tool.</p>	<p>Drag the file from the other tool into the Editor.</p>	<p>For example, drag files from the Current Folder browser or from Windows Explorer.</p>
<p>Open file using a function</p>	<p>Use the <code>edit</code> or <code>open</code> function.</p>	<p>For example, type the following to open <code>collatz.m</code>:</p> <pre data-bbox="1015 1048 1222 1072">edit collatz.m</pre> <p>If <code>collatz.m</code> is not on the search path or in the current folder, use the relative or absolute path for the file.</p>

For special considerations on the Macintosh platform, see “Using File Browser GUIs on Macintosh Platforms to Navigate Within the MATLAB Root Folder” on page 2-172.

MATLAB Code Cells in Files


Use cell mode for running sections of code and publishing. Denote MATLAB code cells with two comment characters (`%%`) at the start of a line and save

the file using `.m` as the extension. If you open a file that contains MATLAB code cells when cell mode is enabled, then highlighting and horizontal lines appear in the file. The Editor reflects the cell toolbar state and the cell display preferences, such as yellow highlighting of the current cell and gray lines between cells.

If you do not want cell mode enabled, select **Cell > Disable Cell Mode**. If cell mode is disabled when you quit a MATLAB session, it is disabled the next time you start a MATLAB session. The converse is true, as well.

The first time you open a file that contains cells, an information bar appears below the cell toolbar, providing links for details about cell mode. Control the display of the information bar as follows:

- Close the information bar by clicking the Close button on the right side of the bar.

The information bar does not appear again, but you can access information about Editor cells from the Show Cell Mode information button  on the cell toolbar.

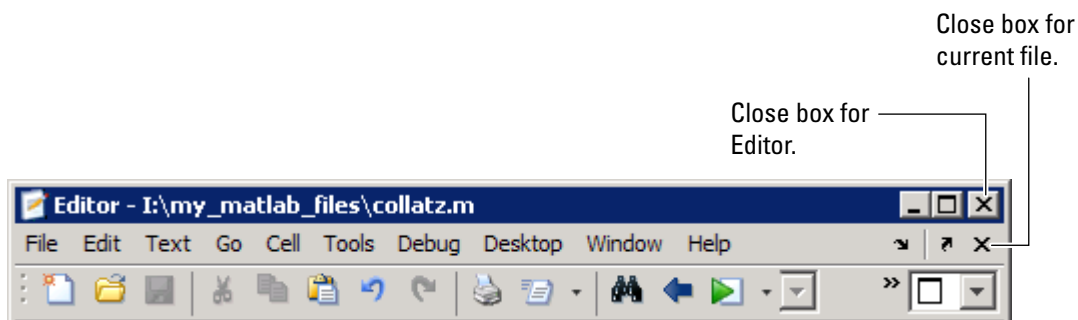
- Hide the cell toolbar by right-clicking in the toolbar and selecting **Cell Toolbar** from the context menu.

Arranging Editor Documents

You can arrange the size and location of documents you open in the Editor. Editor documents follow the same arrangement practices as other desktop documents. For details, see “Opening and Arranging Desktop Documents” on page 2-20.

Closing the Editor

To close the Editor, click the Close button in the *title bar* of the Editor. This button is different from the Close button in the Editor *menu bar*. The latter Close button closes the current file when multiple files are open in a single window.



If multiple files are open, each in a separate window, you can either:

- Close each window separately by clicking the associated Close button.
- Close all documents in all tools at once by selecting **Window > Close All Documents**.

In addition to closing the Editor files and the Editor, this choice closes other desktop documents and their containing tools as well. For example, it closes variables in the Variable Editor and the Variable Editor itself.

Customizing the Editor by Setting Preferences

In this section...

“Overview of Setting Editor/Debugger Preferences” on page 9-13

“Setting General Preferences for the Editor/Debugger” on page 9-15

“Setting Display Preferences” on page 9-16

“Setting Tab and Indent Preferences” on page 9-19

“Setting Language Preferences” on page 9-20

“Setting MATLAB Language Preferences” on page 9-21

“Setting TLC Language Preferences” on page 9-28

“Setting VHDL Language Preferences” on page 9-28

“Setting Verilog Language Preferences” on page 9-29

“Setting C/C++ Language Preferences” on page 9-30

“Setting Java Language Preferences” on page 9-32

“Setting XML/HTML Language Preferences” on page 9-33

“Setting Code Folding Preferences” on page 9-34

“Setting Autosave Preferences” on page 9-35

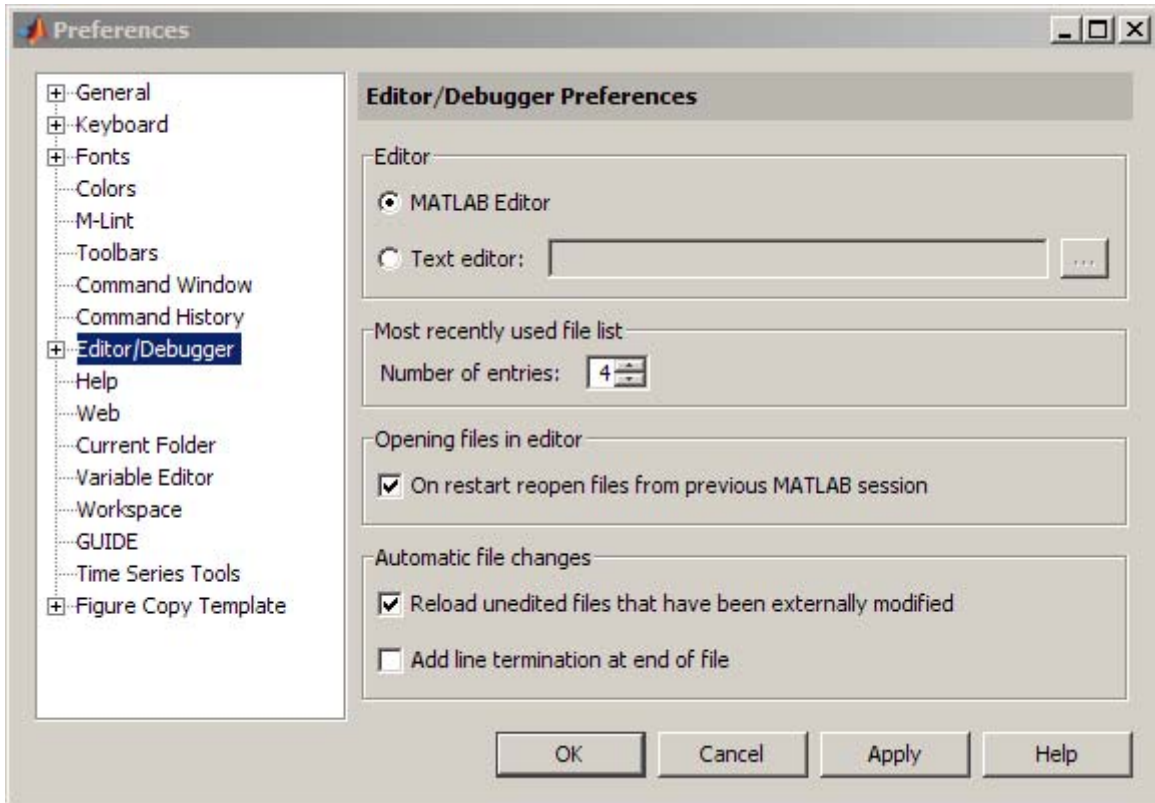
“Additional Information About Editor/Debugger Preferences” on page 9-37

Overview of Setting Editor/Debugger Preferences

To specify the default behavior for various aspects of the Editor:

- 1 Select **File > Preferences > Editor/Debugger**.

The Preferences dialog box opens showing **Editor/Debugger Preferences**.



2 Click next to Editor/Debugger in the left pane to view all categories of Editor/Debugger preferences.

3 Select a category.

The preference pane for that category displays.

4 Modify preferences, and then click **Apply** or **OK**.

You can also set preferences for the Editor toolbars. For details, see “Setting Toolbars Preferences for Desktop Tools” on page 2-156.

Setting General Preferences for the Editor/Debugger

Select **File > Preferences > Editor/Debugger** to specify the preferences on the main pane, listed here. Or click the **+** to see additional Editor/Debugger preferences.

- “Editor” on page 9-15
- “Most Recently Used File List” on page 9-15
- “Opening Files in the Editor” on page 9-16
- “Automatic File Changes” on page 9-16

Editor

MATLAB Editor. Select **MATLAB Editor** to have the MATLAB desktop use the built-in Editor.

Text editor. Select this option to specify that a text editor other than the MATLAB Editor open a file with a `.m` extension from within MATLAB. Specify the full path for the editor application you want to use, such as Emacs or vi.

For example, suppose you do the following:

- 1** Select **Text editor** and specify `c:/Applications/Emacs.exe` in the text field.
- 2** Open a file by selecting **File > Open** in the MATLAB desktop.

The file you specify in step 2 opens in Emacs instead of in the MATLAB Editor.

If you use an editor other than the MATLAB Editor, some Editor/Debugger preferences are still available to MATLAB. Some Editor/Debugger preferences apply to other MATLAB tools.

Most Recently Used File List

Use this preference to specify the number of files that appear in the list of most recently used files at the bottom of the **File** menu. You select a file from the list to open it.

Opening Files in the Editor

On restart reopen files from previous MATLAB session. Select this option to specify that, when it restarts, MATLAB open the files that were open when you last shut it down. The Editor does not open upon startup if this option was cleared and files were open in the Editor when you last closed MATLAB.

Automatic File Changes

Reload unedited files that have been externally modified. This option is useful when you edit files in the MATLAB Editor and outside of MATLAB.

If you select this preference, then the Editor automatically reloads the version of the file open outside of MATLAB, when all the following are true:

- The file is open in both the Editor and outside of MATLAB.
- You have not edited the file in the Editor since you last saved it.
- You edit and save the file outside of MATLAB.

If you clear this preference, and you change the version of the file that is open outside of MATLAB, then MATLAB displays a dialog box. The dialog box notifies you that the current file changed outside of MATLAB, and asks if you want to use the latest version.

Add line termination at end of file. Select this preference to have the Editor add a new empty line to the end of the file automatically, if the last line in the file is not empty. This option is useful if you use your files with other products that expect a new line (sometimes referred to as a <CR>) at the end of the file.

Setting Display Preferences

Select **File > Preferences > Editor/Debugger > Display** to specify these preferences:

- “General Display Options” on page 9-17
- “Right-Hand Text Limit” on page 9-17

See also Desktop Preferences.

General Display Options

Highlight Current Line. Select this preference to highlight the current line, that is, the row with the caret (also called the cursor). This preference is helpful, for example, to see where copied text will appear when you paste it. Then specify the color used to highlight the line.

Show line numbers. Select this preference to show line numbers along the left edge of the Editor window. Showing line numbers helps you to use various Editor features, such as **Edit > Go To Line**. When you clear this item, line numbers do not display.

Enable datatips in edit mode. Select this preference to see data tips while editing a MATLAB code file. By default, data tips do not display while editing (edit mode), although they always display while debugging (debug mode). In edit mode, the data tips display the values of variables in the base workspace, so this is useful for MATLAB scripts rather than functions. In edit mode for a function, the data tip for a variable displays a value if that variable also exists in the base workspace. In this case, the data tip displays the value of the base workspace variable, not the value of the variable in the function file.

While you are debugging, you cannot turn off the display of data tips, and they show the value of the variables in the workspace selected in the **Stack**. For more information about data tips, see “Viewing Values as Data Tips in the Editor” on page 9-153.

Right-Hand Text Limit

By default, a gray vertical line that is 1-pixel wide appears at column 75 in the Editor. This vertical line indicates when a line of code becomes wider than desired. Consider setting a right-hand text limit for reasons such as the following:

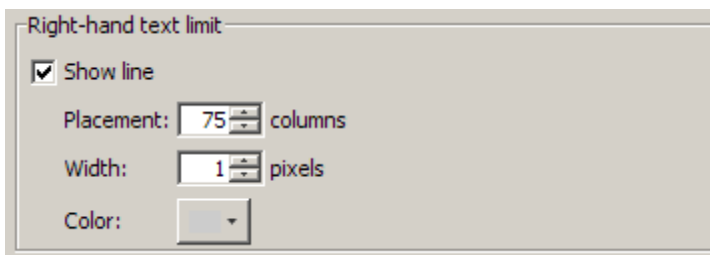
- To prevent the need to scroll from left to right to see an entire line of text in the Editor
- To keep each line below a limit imposed by another text editor in which you intend to view the code

- To keep each line below a character limit required to ensure that the file prints without cropped text

Consider printing a test page to determine the most appropriate value for the printer printing the file, the margin settings for the printer, and the font size you are using.

This limit is a visual cue only and does not prevent text from exceeding the limit. For information on setting a value to wrap comment text at a specified column number automatically, see “Comment Formatting” on page 9-22.

Note If the vertical line appears to be in the wrong column, it might be because you are not using a monospaced font. See “Setting Fonts Preferences for Desktop Tools” on page 2-141 for details.



Show line. Select this preference to display the vertical line in the Editor. Clear it to hide the vertical line.

Placement. Type or select a new value for this field to change the column where the vertical line appears.

The actual column number where you set this preference is most useful when the font preference for the Editor is a monospaced font. In a monospace font, all characters are the same width. For details, see “Setting Fonts Preferences for Desktop Tools” on page 2-141.

Width. Type or select a value for this field to change the width (in pixels) of the vertical line.

Color. Click the down arrow next to the color block to open a palette of colors from which you can choose a new color for the vertical line. Increasing this value might improve the visibility of the vertical line on liquid crystal displays (LCDs) and projectors.

Setting Tab and Indent Preferences

Select **File > Preferences > Editor/Debugger > Tab** to specify the preferences that follow.

Tabs and Indents

Tab size. Specify the amount of space inserted when you press the **Tab** key. When you change the **Tab size**, it changes the tab size for existing lines in that file, unless the **Tab key inserts spaces** also is selected.

Tab key inserts spaces. Select this item if you want a series of spaces to be inserted when you press the **Tab** key. If the item is not selected, a tab acts as one space whose length is equal to the **Tab size**.

Indent size. Specify the indent size for smart indenting.

Emacs-style Tab key smart indenting. This indenting convention is based on the style used by the Emacs editor and is like the **Apply smart indenting while typing** preference. With this preference selected, lines indent according to smart indenting preferences when you position the cursor in a line or select a group of lines, and then press the **Tab** key. With this preference selected, you cannot use tabs within a line.

See Also

The following sections provide information about additional preferences for indenting:

- “Setting Language Preferences” on page 9-20 for additional indenting preferences
- “Setting Keyboard Preferences for Desktop Tools” on page 2-138 “Setting Keyboard Preferences for Desktop Tools” on page 2-138 for function hints, tab completion, and delimiter matching preferences

- “Specifying Options for MATLAB Using Preferences” on page 2-124

Setting Language Preferences

MATLAB applies language preferences based on the file name extension of the file open in the Editor.

To specify language preferences, such as syntax highlighting:

- 1** Select **File > Preferences > Editor/Debugger > Language**, for the type of file you are editing.
- 2** Select the **Language**.

The preference dialog box displays the preferences for that language. If the language you are using is not an option, it means that setting language preferences for that language is not available.

- 3** Change preferences for the language you selected, and then click **Apply**.

For example, when you edit a file with a `.html` extension, the language preferences set for XML/HTML apply to that file. If you have a Java file open at the same time, the Java language preferences apply to the Java file.

File Extensions

To specify a file extension associated with the selected language:

- 1** Optionally, add or change the file extensions associated with the language in the **Language** field by clicking the **Add** and **Remove** buttons

Changes you make to the file extensions do not apply to open files until you close and reopen them.

- 2** Select a file extension from the **File extensions** list.
 - If a file has no extension (for example, a new untitled file), MATLAB treats it as a file with a `.m` extension.
 - If a file has an extension, but it is not in the **File extensions** list, MATLAB does not apply any language preferences to that file.
- 3** Click **Apply**.

The following table presents the default file extensions for each language.

Language Preference	Default File Extensions
MATLAB	.m (uppercase or lowercase)
TLC	.tlc
VHDL	.vhd, .vhdl
Verilog	.v
C/C++	.c, .cpp, .h, .hpp
Java	.java
XML/HTML	.xml, .xsl, .wsdl, .html, .htm, .shtml

For details about preferences for each language, see:

- “Setting MATLAB Language Preferences” on page 9-21
- “Setting TLC Language Preferences” on page 9-28
- “Setting VHDL Language Preferences” on page 9-28
- “Setting Verilog Language Preferences” on page 9-29
- “Setting C/C++ Language Preferences” on page 9-30
- “Setting Java Language Preferences” on page 9-32
- “Setting XML/HTML Language Preferences” on page 9-33

Setting MATLAB Language Preferences

Select **File > Preferences > Editor/Debugger**, and then from **Languages**, select **MATLAB** to specify these preferences for editing MATLAB code files:

- “Syntax highlighting” on page 9-22
- “Comment Formatting” on page 9-22
- “Indenting” on page 9-23
- “File extensions” on page 9-27

See also “Setting Code Analyzer Preferences” on page 9-124.

Syntax highlighting

Enable syntax highlighting.

- Select to show colors that help you identify certain constructs, like matching `if/else` statements.

Click **Set syntax colors** to open the **Colors** preference pane, and specify the colors that you want the Editor to use for syntax highlighting.

- Clear to show all text in black.

The syntax colors preferences apply to all tools that use syntax highlighting, including the Command Window, the Command History, the Editor, and other tools. For a description of syntax highlighting, see “Setting Colors Preferences” on page 2-150.

Comment Formatting

You can specify where and when to wrap comments in MATLAB code files using these preferences:

- **Max width**—Type a value for the maximum width for comment text (in number of columns).
- Select where you want column counting to begin:
 - Consider selecting columns from start of line when the absolute width of the comments is important. For example, 75 columns from the start of the line is the width that fits on a printed page when you use the default font for the Editor.
 - Consider selecting columns from start of comment when comments are indented, and you want each block of comments to have a consistent indent and width.
- To have MATLAB wrap comments at the **Max width** value as you enter new comments in a file, select **Wrap comments automatically while typing**.

Existing comments and comments that you paste into a file are unaffected. If you want, you can wrap such comments manually.

For information on wrapping comments manually, see “Wrapping Comments in MATLAB Code” on page 9-45.

For information on adjusting the gray vertical line that appears by default at the 75th column in the Editor, see the section entitled “Right-Hand Text Limit” on page 9-17.

Example of Comment Formatting. This example demonstrates comment wrapping settings that are useful for formatting comments for output from the help function:

- 1 Set the **Max width** to 75 columns from start of comment.
- 2 Select **Wrap comments automatically while typing**.
- 3 Starting at column 12, type a comment in a MATLAB code file.

As you type the text, when you reach the 87th column, the comment automatically continues on the next line, indenting 12 columns.

This behavior keeps comment lines contained within a block at the same width, regardless of where the comment starts.

Indenting

You can set preferences to have MATLAB automatically apply indenting to your MATLAB code. The indenting options you specify, however, apply only to lines you enter after changing the preference; they do not affect indenting for existing lines. For information on changing the indenting for existing lines, see “Manually Applying Indenting” on page 9-54.

Apply smart indenting while typing. Select this option to specify that you want:

- The body of loops to indent automatically within the start and end of the loop statement.
- The lines you indent using tabs or spaces to result in subsequent lines automatically aligning with the indented line.
- Functions to indent automatically as specified with the **Function indenting format** option.

After entering a new line, select **Text > Decrease Indent** to move text back to a previous indent level, if you want.

Although you can manually insert tabs at the start of a line, smart indenting might not work properly if you do. To correct selected lines, select **Text > Smart Indent**.

Clear **Apply smart indenting while typing** to specify that you want:

- Lines to align on the left by default
- To specify indenting manually, using the **Tab** and space bar keys
- Functions to not indent automatically as specified by the **Function indenting format** option

When you clear **Apply smart indenting while typing** you can still indent functions, as specified by the **Function indenting format** option by:

- 1 Selecting the functions in the Editor.
- 2 Selecting **Text > Smart Indent**.

To set the indent size for smart indenting and the tab size for manual indenting, see “Setting Tab and Indent Preferences” on page 9-19.

Example of Smart Indenting.

```
sequence = n
next_value = n;
while next_value > 1
    if rem(next_value,2)==0
        next_value = next_value/2;
    else
        next_value = 3*next_value+1;
    end
    sequence = [sequence, next_value]
end
```

Created with Smart indenting selected. Did not insert tabs manually. Indenting was inserted automatically as text was entered.

Example of Smart Indenting Cleared with Manual Tabs.

```
sequence = n
next_value = n;
while next_value > 1
    if rem(next_value,2)==0
        next_value = next_value/2;
    else
        next_value = 3*next_value+1;
    end
    sequence = [sequence, next_value]
end
```

Created with Smart indenting deselected. Indentation created by manually inserting tab before each indented line

Example of Smart Indenting Cleared Without Tabs.

```
sequence = n
next_value = n;
while next_value > 1
if rem(next_value,2)==0
next_value = next_value/2;
else
next_value = 3*next_value+1;
end
sequence = [sequence, next_value]
end
```

Created with Smart indenting disabled and no tabs manually inserted.

Function Indenting Format. Specify how functions indent in the Editor, as follows:

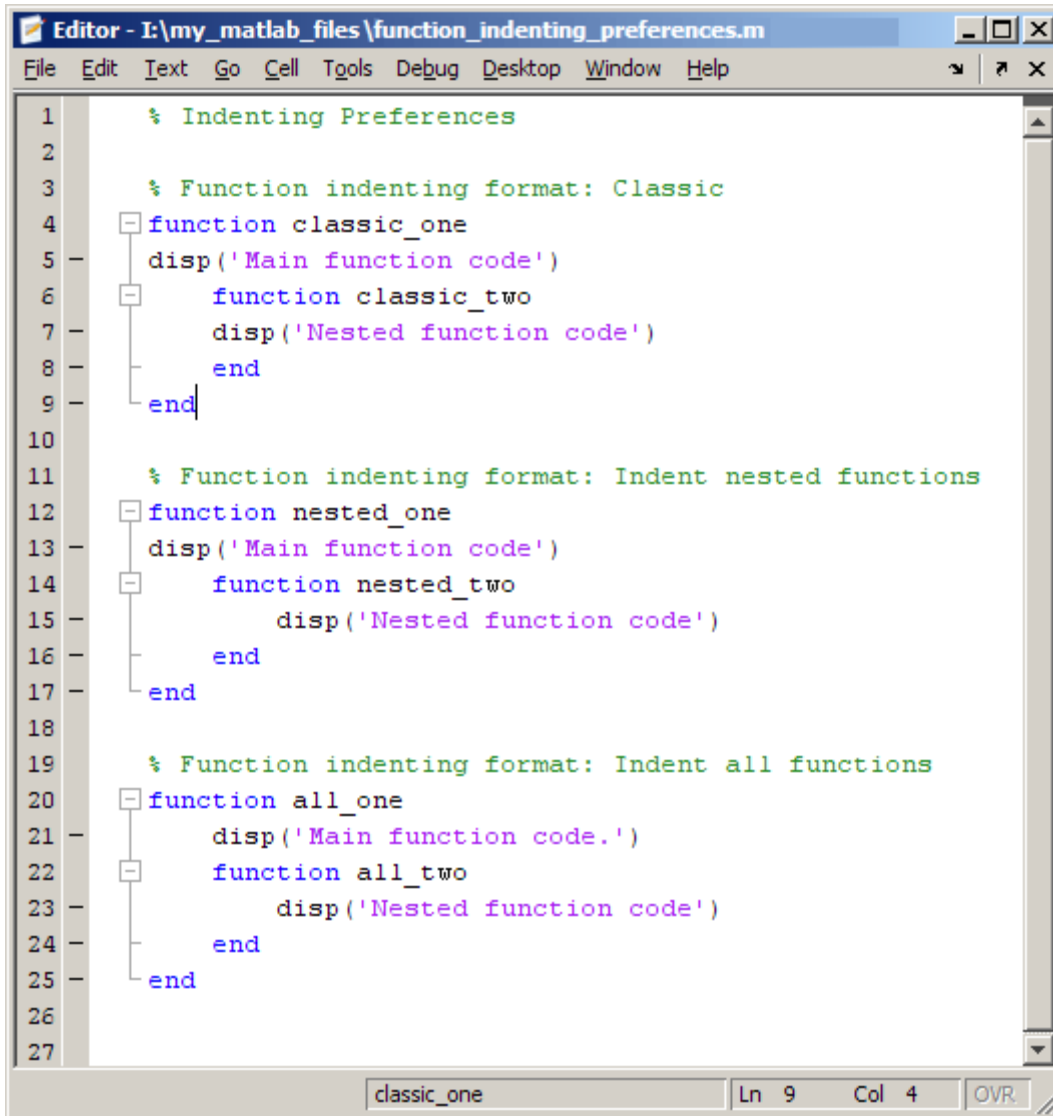
- 1 Select or clear **Apply smart indenting while typing**, depending on whether you want your function indenting choice applied automatically, as you type code in the Editor. See “Apply smart indenting while typing” on page 9-23 for details.
- 2 Choose a function indenting format. The styles for indenting functions are:
 - **Classic** — The Editor aligns the function code with the function declaration.
 - **Indent nested functions** — The Editor indents the function code within a nested function.

- **Indent all functions** — The Editor indents the function code for both main and nested functions.

3 For nested functions, provide an end statement at the start of a line for each function declaration.

An end statement aligns with its function declaration only when the end statement appears at the start of the line.

Examples of Function Indenting Format Preference. The following image illustrates each of the function indenting formats.



```
1      % Indenting Preferences
2
3      % Function indenting format: Classic
4      function classic_one
5          disp('Main function code')
6          function classic_two
7              disp('Nested function code')
8          end
9      end
10
11     % Function indenting format: Indent nested functions
12     function nested_one
13         disp('Main function code')
14         function nested_two
15             disp('Nested function code')
16         end
17     end
18
19     % Function indenting format: Indent all functions
20     function all_one
21         disp('Main function code.')
22         function all_two
23             disp('Nested function code')
24         end
25     end
26
27
```

classic_one Ln 9 Col 4 OVR

File extensions

See “File Extensions” on page 9-20 for general Language preferences.

Setting TLC Language Preferences

Target Language Compiler (TLC) is an integral part of Real-Time Workshop®. Select **File > Preferences > Editor/Debugger > Language > TLC** to specify these preferences for editing TLC files:

- “Syntax highlighting” on page 9-28
- “File Extensions” on page 9-28

See also “Setting Code Analyzer Preferences” on page 9-124.

Syntax highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify TLC constructs, such as commands and macros. Optionally, change the colors used for these elements.

Commands. Color for TLC commands, such as `LibBlockMatrixParameter`.

Comments. Color for the comment indicator, `%%` or `/%...%/`, and its associated text.

C Strings. Color for C strings.

Macros. Color for TLC macros.

File Extensions

See “File Extensions” on page 9-20 for general Language preferences.

Setting VHDL Language Preferences

Select **File > Preferences > Editor/Debugger > Language** to specify these preferences for editing VHDL files:

- “Syntax highlighting” on page 9-29
- “File extensions” on page 9-29

Syntax highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify certain VHDL constructs, such as keywords and operators. Specify the colors used for these elements.

Keywords. Color for VHDL keywords, such as SIGNAL.

Comments. Color for the comment indicator, -- , and its associated text.

Operators. Color for operators, such as <=.

Strings. Color for strings, such as "0000".

File extensions

See “File Extensions” on page 9-20 for general Language preferences.

Setting Verilog Language Preferences

Select **File > Preferences > Editor/Debugger > Language** to specify these preferences for editing Verilog files:

- “Syntax highlighting” on page 9-29
- “File extensions” on page 9-30

Syntax highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify certain Verilog constructs, such as keywords and operators. Specify the colors used for these elements.

Keywords. Color for Verilog keywords, such as assign.

Comments. Color for the comment indicator, // , and its associated text.

Operators. Color for operators, such as =.

Strings. Color for strings, such as "Test Completed".

File extensions

See “File Extensions” on page 9-20 for general Language preferences.

Setting C/C++ Language Preferences

Select **File > Preferences > Editor/Debugger > Language** to specify these preferences for editing C or C++ language files:

- “Syntax highlighting” on page 9-30
- “Indenting” on page 9-30
- “File extensions” on page 9-31

Syntax highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify certain C constructs, such as methods. Specify the colors used for these elements.

Keywords. Color for keywords, such as `if`.

Strings. Color for terms enclosed in double quotation marks, for example, `"default"`.

Characters. Color for terms enclosed in single quotation marks, for example, `'a'`.

Comments. Color for text following the comment indicator, `//`, as well as for the block comment indicators, `/*` and `*/`, and the code in between.

Preprocessor. Color for text following the preprocessor symbol, `#`.

Bad characters. Color for illegal characters.

Indenting

You can set preferences to specify that you want MATLAB to apply indenting to your C/C++ files automatically. The indenting options you specify, however, apply only to lines you enter after changing the preference; they do not affect

the indenting for existing lines. For information on changing the indenting for existing lines, see “Manually Applying Indenting” on page 9-54.

Apply smart indenting while typing. Select this option to specify that you want:

- The body of loops to indent automatically within the start and end of the loop statement
- The lines you indent using tabs or spaces to result in subsequent lines automatically aligning with the indented line

After entering a new line, press **Shift+Tab** to move text back to a previous indent level if you want.

Although you can manually insert tabs at the start of a line, be aware that smart indenting might not work properly. Use the **Text** menu entry for **Smart Indent** to correct selected lines.

Clear **Apply smart indenting while typing** to specify that you want:

- Lines to align on the left by default
- To specify indenting manually using the tab and space keys

To set the indent size for smart indenting and the tab size for manual indenting, see “Setting Tab and Indent Preferences” on page 9-19.

For examples, see:

- “Example of Smart Indenting” on page 9-24
- “Example of Smart Indenting Cleared with Manual Tabs” on page 9-25
- “Example of Smart Indenting Cleared Without Tabs” on page 9-25

File extensions

See “File Extensions” on page 9-20 for general Language preferences.

Setting Java Language Preferences

Select **File > Preferences > Editor/Debugger > Language > Java** to specify these preferences for editing Java files:

- “Syntax highlighting” on page 9-32
- “Indenting” on page 9-32
- “File extensions” on page 9-33

See also “Setting Keyboard Preferences for Desktop Tools” on page 2-138 for the Editor/Debugger and Command Window and “Setting Language Preferences” on page 9-20.

Syntax highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify certain Java constructs, such as methods. Specify the colors used for these elements.

Show methods. Text style for methods to appear in when you type them: Bold, Italic, or Plain (no special highlighting).

Keywords. Color for keywords, such as `if`.

Strings. Color for terms enclosed in double quotation marks, for example, `"alive"`.

Characters. Color for terms enclosed in single quotation marks, for example, `'a'`.

Comments. Color for text following the comment indicator, `//`, as well as for the block comment indicators, `/*` and `*/`, and the code in between.

Bad characters. Color for illegal characters.

Indenting

Select or clear **Apply smart indenting while typing** to specify whether you want the Editor to apply indenting to your Java files when you press the

Enter key to type in a new line. The effects are the same as those for C/C++ files. For details and examples, see the C/C++ preference for indenting.

File extensions

See “File Extensions” on page 9-20 for general Language preferences.

Setting XML/HTML Language Preferences

Select **File > Preferences > Editor/Debugger > Language > XML/HTML** to specify these preferences for editing XML, XSL, WSDL, HTML, HTM, and SHTML files:

- “Syntax highlighting” on page 9-33
- “Indenting” on page 9-34
- “File extensions” on page 9-34

Syntax highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify XML, WSDL, and HTML constructs, such as elements and tags. Optionally, change the colors used for these items.

Attribute name. Color for attribute names.

Attribute value. Color for values, such as the source for an image, for example "myimage.gif", in ``.

CDATA section. Color for a CDATA section in XML files.

Character. Color for characters and entities, such as the nonbreaking space, ` `.

Comment. Color for text contained within comment indicators, `<!--` and `-->`.

DOCTYPE declaration. Color for DOCTYPE declarations.

Error. Color for invalid entries, such as `<+1>` for font size, which is deprecated in favor of style sheets in the HTML 4.01 specification.

See the W3C Web site for details.

Operator. Color for operators, such as the equal sign (=).

Processing instruction. Color of the instructions to the application that processes the XML file.

Tag. Color for tags and elements, such as or .

Indenting

Select or clear **Apply smart indenting while typing** to specify whether you want the Editor to apply indenting to your files when you press the **Enter** key. The effects are the same as those for C/C++ files. For details and examples, see the C/C++ preference for indenting.

File extensions

See “File Extensions” on page 9-20 for general Language preferences.

Setting Code Folding Preferences

Select **File > Preferences > Editor/Debugger > Code Folding** to set these preferences for hiding and revealing code in MATLAB files, including function and class code, function and class help code, programming control blocks, and so on.

- “Enable Code Folding” on page 9-34
- “Enable Code Folding by Programming Construct” on page 9-35
- “Fold Initially” on page 9-35

See also “Making MATLAB Code Files More Readable” on page 9-53, and “Working with Functions in Files”.

Enable Code Folding

Select this option to enable code folding in MATLAB files; clear this option to disable code folding in MATLAB files. (Use the table that appears below this option to enable and disable code folding for selected programming constructs within a MATLAB file.)

This option has the following effects:

- If you select this option, code folding is enabled for all programming constructs that you enable in the table that appears below this option in the Preferences dialog box.
- If you clear this option, code folding is disabled for all programming constructs, regardless of the programming constructs that you may have previously enabled in the table that appears below this option in the Preferences dialog box.

Enable Code Folding by Programming Construct

If you select the **Enable code folding** option, you can independently enable or disable code folding for individual programming constructs (function code, function help, programming control blocks, class code, and so on) by selecting or clearing the **Enable** check box that corresponds to each construct in the table on the Preferences dialog box. By default, code folding is enabled for all programming constructs except if/else blocks and switch/case blocks.

Fold Initially

If you enable code folding for a programming construct, you can specify how the Editor displays that construct the first time that you open a MATLAB file that existed prior to MATLAB version 7.5. Select or clear the **Fold Initially** check box associated with a construct to direct the Editor to collapse or expand the associated construct, respectively, the first time you open a file that existed prior to MATLAB version 7.5 (R2007b) using MATLAB version 7.5 or later.

Setting Autosave Preferences

Select **File > Preferences > Editor/Debugger > Autosave** to specify these preferences for the Editor's autosave feature.

Enable autosave in the MATLAB Editor

The MATLAB Editor automatically saves a copy of the current version of the file you are editing. Clear this check box if you do not want the MATLAB Editor to save the copy automatically.

Save Options

Save every n minutes. Specify how often you want the Editor to save, automatically, a copy of the file you are editing.

Save untitled files. Select this check box if you want the Editor to save automatically a copy of new files that you have not yet saved, which are therefore untitled. If selected, the first autosave file is `Untitled.asv`. If the folder already contains a file named `Untitled.asv`, the autosave file is `Untitled2.asv`, and so on, for additional unnamed files.

If the autosave feature creates `Untitled.asv` and you later save the file as `filename.m`, the next autosave version is `filename.asv`. `Untitled.asv` remains until you delete it.

Close Options

Automatically delete autosave files. With this preference selected, MATLAB deletes the autosave file when you close the source file in the Editor.

Filename

Specify the extension used for autosave files. The default setting for Microsoft Windows platforms is the extension `.asv` (for *autosave*), making the autosave file `filename.asv`. For Microsoft Windows and UNIX platforms, you can select **Replace extension with** and specify any extension.

For UNIX platforms, the default is **Append file with** the tilde (`~`) character, making the autosave file `filename.m~`. For Windows and UNIX platforms, you can select **Append file with** and specify a different string to append to the file.

Location

Specify the full path for the folder where you want autosave files stored. You can specify a single folder for all autosave files, such as a folder you create called `autosave_files`. For the Editor to create an autosave file, you must have write permission for the specified location.

If you do not specify a location, MATLAB:

- Stores the autosave file for each named file in the same folder as the source file, that is, the file you are editing.
- Stores the autosave file for each untitled file in the folder that was the current folder when you opened the file.

Additional Information About Editor/Debugger Preferences

Additional information that relates to the Editor and preferences includes:

- “Setting Autosave Preferences” on page 9-35
- “Setting Fonts Preferences for Desktop Tools” on page 2-141
- “Setting Colors Preferences” on page 2-150
- “Setting Color Preferences for Programming Tools” on page 2-154
- “Specifying Options for MATLAB Using Preferences” on page 2-124
- “Setting Keyboard Preferences for Desktop Tools” on page 2-138

Entering Statements in the Editor

In this section...

- “Using Command Window Features in the Editor” on page 9-38
- “Entering Text in Insert or Overwrite Mode” on page 9-39
- “Changing the Case of Selected Text” on page 9-39
- “Undoing and Redoing Editor Actions” on page 9-40
- “Adding Comments” on page 9-40
- “Completing Statements in the Editor — Tab Completion” on page 9-46

Using Command Window Features in the Editor

After opening an existing file or creating a new file in the Editor, enter statements in the file. Use the same practices as for entering statements in the Command Window as described in Chapter 3, “Running Functions — Command Window and History”:

- “Case and Space Sensitivity” on page 3-17
- “Entering Multiple Lines Without Running Them” on page 3-18
- “Entering Multiple Functions in a Line” on page 3-20
- “Entering Multiple-Line (Long) Statements Using Line Continuation” on page 3-20
- “Suppressing Output” on page 3-47
- “Formatting and Spacing Numeric Output” on page 3-48
- “Matching Delimiters (Parentheses)” on page 3-24
- “Viewing Function Syntax Hints While Entering a Statement” on page 3-33
- “Getting Help for a Function Shown in the Command Window or Editor” on page 3-38
- “Finding Functions Using the Function Browser” on page 3-40

Entering Text in Insert or Overwrite Mode

On Windows and UNIX platforms, the Editor enables you to enter text in either insert or overwrite mode. By default, when you enter text in the Editor, you use *insert mode*. In this mode, the Editor inserts the text you type within the existing text. In *overwrite mode*, the text you type overwrites the existing text.

Note The Macintosh platform does not support overwrite mode.

Determining the Current Typing Mode

The Editor indicates the current mode as follows:

- In insert mode, the cursor is a vertical bar and the text **OVR** appears dimmed in the status bar.
- In overwrite mode, the cursor is a wide block and the text **OVR** is not dimmed in the status bar.

Toggling Between Insert and Overwrite Mode

To toggle between insert and overwrite mode:

- 1** In the Editor, place the cursor where you want to enter text.
- 2** Press the **Insert** key to toggle the typing mode from insert to overwrite mode, or the reverse.

The **Insert** key is the default keyboard shortcut for changing the typing mode. For details, on changing keyboard shortcuts, see “Customizing Keyboard Shortcuts” on page 2-79.

Changing the Case of Selected Text

To change the case of text in the Editor, select the text. Then, from the **Text** menu, select one of the following:

- **Change to Upper Case** to change all text to uppercase

- **Change to Lower Case** to change all text to lowercase
- **Reverse Case** to change the case of each letter

This is useful, for example, when copying syntax from command-line help for a function. In command-line help, function and variable names are distinguished from the rest of the text by using uppercase. However, such code will not run in the MATLAB Editor or Command Window. In this example, the text is copied and pasted from the output of `help get`.

```
V = GET(H, 'Default')
```

Select all of the text. Select **Text > Change to Lower Case**. The text becomes

```
v = get(h, 'default')
```

If instead you select **Reverse Case** for

```
V = GET(H, 'Default')
```

the case changes to

```
v = get(h, 'dEFAULT')
```

Undoing and Redoing Editor Actions

You can undo many of the Editor actions listed in **Edit** and **Text** menus. Select **Edit > Undo**. You can undo multiple times in succession until there are no remaining actions to undo. Select **Edit > Redo** to reverse an undo.

Adding Comments

Comments in MATLAB code are strings or statements that do not execute. Add comments in a file to describe the code or how to use it. Comments determine what text displays when you run `help` for a file name. Use comments when testing your files or looking for errors—temporarily turn lines of code into comments to see how the file runs without those lines. These topics provide details:

- “Commenting in MATLAB Code Using the MATLAB Editor” on page 9-41

- “Commenting in Java and C/C++ Files Using the MATLAB Editor” on page 9-42
- “Commenting in MATLAB Code Using Any Text Editor” on page 9-42
- “Commenting Out Part of a Statement” on page 9-44
- “Wrapping Comments in MATLAB Code” on page 9-45

Commenting in MATLAB Code Using the MATLAB Editor

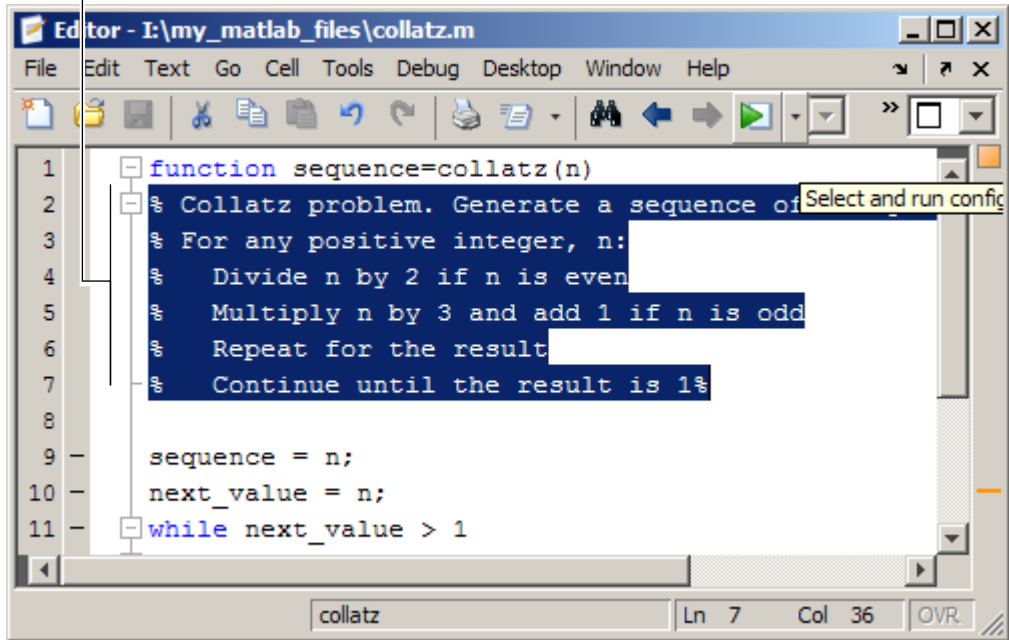
You can comment the current line or a selection of lines in MATLAB code.

- 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.

The Editor adds a comment symbol, % at the start of each selected line. The color of the commented text becomes green, or the color specified for comments. See “Syntax Highlighting” on page 9-53.

To uncomment the current line or a selected group of lines, select **Uncomment** from the **Text** menu, or right-click and select it from the context menu.

Click to the left of a line to select it.
Click + drag to select multiple lines.



Commenting in Java and C/C++ Files Using the MATLAB Editor

For Java and C/C++ files, selecting **Text > Comment** adds the // symbols at the front of the selected lines. Similarly, **Text > Uncomment** removes the // symbols from the front of selected lines in Java and C/C++ files.

Commenting in MATLAB Code Using Any Text Editor

You can make any line in MATLAB code a comment by typing % at the beginning of the line. To put a comment within a line, type % followed by the comment text. MATLAB software treats all the information after the % on a line as a comment. For instance, if the following line appears in a file, then MATLAB ignores the line when you run the file:

```
% This is a comment.
```


To uncomment any line, delete the comment symbol, %.

To comment a contiguous group of lines, type %{ before the first line and %} after the last line you want to comment. This is referred to as a block comment. The lines that contain %{ and %} can contain spaces, but no other text. After typing the opening block comment symbol, %{, all subsequent lines assume the syntax highlighting color for comments until you type the closing block comment symbol, %}. Remove the block comment symbols, %{ and %}, to uncomment the lines.

This example shows some lines of code commented out. When you run the file, the commented lines do not execute. This is useful when you want to identify the section of a file that is not working as expected.

Comment a block of code by adding %{ before the first line and %} after the last line.

```

a = magic(3)
%{
sum(a)
diag(A)
sum(diag(a))
%}
sum(diag(fliplr(a)))

```

You can easily extend a block comment without losing the original block comment, that is, create a nested block comment, as shown in the following example.

Create a nested comment, that is, a block comment within a block comment.

Extended comment

```

%{
a = magic(3)
%{
sum(a)
diag(A)
sum(diag(A))
%}
sum(diag(fliplr(a)))
%}

```

Original comment

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 software ignores any text on the line after the %.

```
a = zeros(10) % Initialize matrix
```

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

To comment out text within a multiline statement, use the ellipsis (...). MATLAB ignores any text appearing after the ... on a line and continues processing on the next line. This effectively makes a comment out of anything on the current line that follows the The following example comments out the Middle Initial line.

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

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

```
... 'Middle Initial, '...
```

Notice that Middle Initial is green, which is the syntax highlighting color for a comment.

MATLAB continues processing the statement with the next line

```
'Title']
```

MATLAB effectively runs

```
headers = ['Last Name, ' ...  
'First Name, ' ...  
'Title']
```

Wrapping Comments in MATLAB Code

By default, as you type them in the Editor, comments wrap whenever they reach a column width of 75. In addition, you can manually wrap comment lines, such as those that you paste into a file. Use a combination of these techniques when you open an existing file with comments that exceed the desired width. Manually wrap existing comments and allow the Editor to automatically wrap the comments you type into the file.

Wrapping Comments Manually.

- 1 Place the cursor anywhere within a block of contiguous lines of comments.
- 2 From the context menu, select **File > Wrap Comments**.

MATLAB wraps all the comments in the block. To wrap only a subset of the comments, select the subset before you wrap it.

For information on changing the default comment wrapping behavior, see “Comment Formatting” on page 9-22.

Exclusions from Comment Wrapping. When you wrap comments, the Editor does *not* wrap the following:

- A bulleted list item onto the line that precedes it
A bulleted list item begins with * or # .
- Code cell titles (%% followed by a space)
- Long strings, such as URLs

This behavior preserves the formatting required for Publishing MATLAB Code Files.

Completing Statements in the Editor – Tab Completion

The Editor helps you automatically complete the names of these items as you type them in MATLAB program files:

- Functions, including subfunctions and nested functions
- Models
- Class and package names
- Variables, including structures and objects
- Handle Graphics properties

Type the first few characters of the item name, and press the **Tab** key. MATLAB also offers tab completion in the Command Window.

The following requirements and limitations apply:

- The tab completion preference for the Editor must be enabled. For details, see “Setting Keyboard Preferences for Desktop Tools” on page 2-138.
- The Editor searches for functions, models, classes, and package names on the MATLAB search path and in the current folder.
- The Editor completes the names of subfunctions and nested functions within the current file. Nested functions appear in the list only when they are available at the current location of the cursor.
- For variables and figures, the Editor:
 - Completes names of all variables and properties of figures in the current workspace. The current workspace appears in the **Stack** on the toolbar.
 - Completes names of variables defined in the current file. The variable must be valid at the current location of the cursor (that is, already defined).
 - Does not complete the field names of structure arrays defined only within the current file.
 - Does not complete method or property names for objects defined only within the current file.

For more information, see the following examples:

- “Basic Example — Unique Completion” on page 9-47
- “Multiple Possible Completions” on page 9-48
- “Narrowing Completions Shown” on page 9-49
- “Tab Completion for Structures” on page 9-50
- “Tab Completion for Properties” on page 9-51
- “Using Tab for Spacing” on page 9-52

Basic Example — Unique Completion

This example illustrates a basic use for tab completion in the Editor. In a MATLAB program file open in the Editor, type the beginning of a function name, for example,

```
horz
```

and press **Tab**. The Editor automatically completes the name, which for this example displays the function name

```
horzcat
```

Complete the statement, adding any arguments, operators, or options. If the Editor does not complete the name `horzcat` but instead moves the cursor to the right, you do not have the preference set for tab completion.

You can use tab completion anywhere in the line, not just at the beginning. For example, if you type

```
a = horz
```

and press **Tab**, the Editor completes `horzcat`.

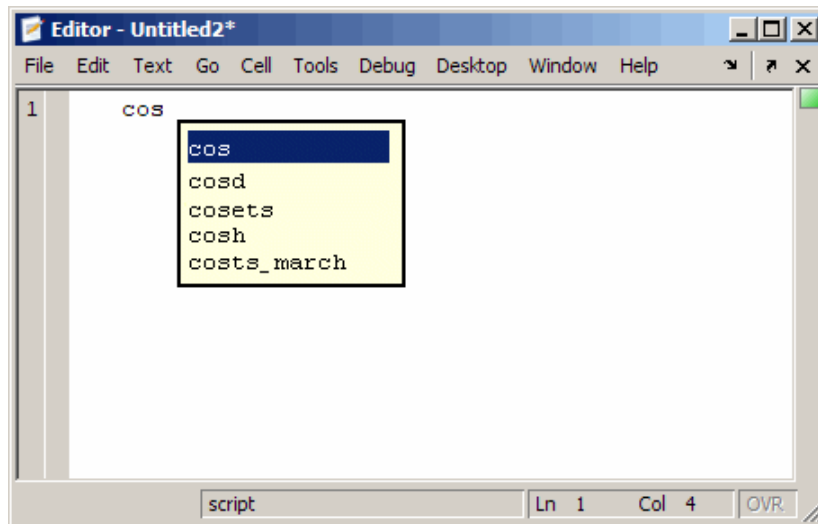
The Editor also completes the names of variables. For example, if you created a variable `costs_march`, type `cost` and press **Tab**. The Editor completes the variable name `costs_march`. If the Editor displays `No Completions Found`, `costs_march` is not available at the current location of the cursor within the file.

Multiple Possible Completions

If there is more than one name that starts with the characters you typed, when you press the **Tab** key, the Editor displays a list of all names that start with those characters. For example, assume that you had created the variable `costs_march` in the base workspace. In a MATLAB program file open in the Editor, type

```
cos
```

and press **Tab**. The Editor opens.



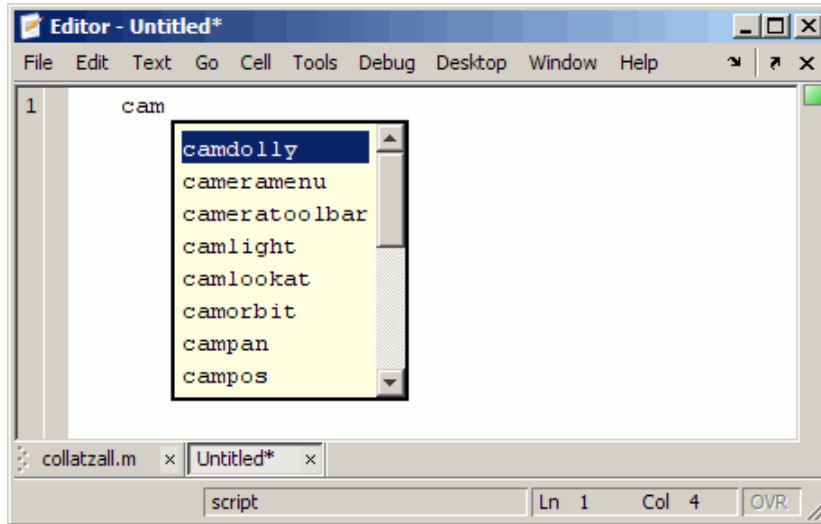
The resulting list of possible completions includes the variable name you created, `costs_march`, but also includes functions and models that begin with `cos`, including `cosets` from Communications Toolbox, if it is installed and on the MATLAB search path.

Continue typing to make your entry unique. For example, type the next character, such as `t` in the example. The Editor selects the first item in the list that matches what you typed, in this case, `costs_march`. Press **Enter** (or **Return**) or **Tab** to select that item, which completes the name in the file. In the example, the Editor displays `costs_march` at the prompt.

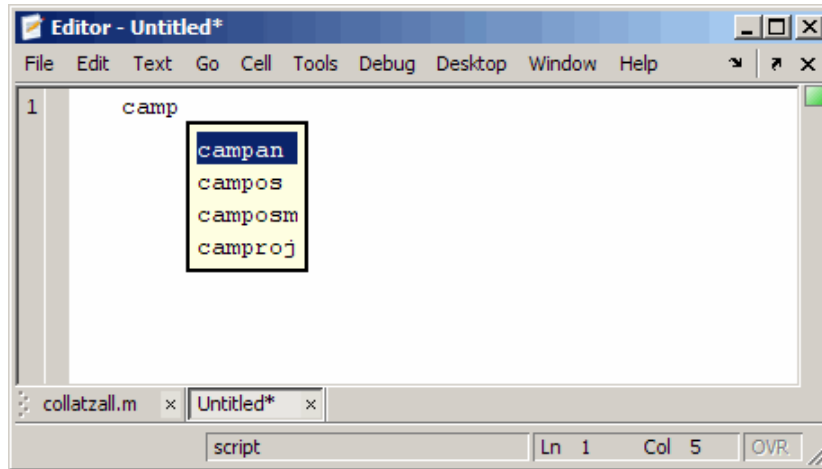
You can navigate the list of possible completions using up and down arrow keys, and **Page Up** and **Page Down** keys. You can clear the list without selecting anything by pressing **Esc**. The list of possible completions might include items that are not valid commands, such as private functions.

Narrowing Completions Shown

You can narrow the list of completions shown by typing a character and then pressing **Tab** if the **Keyboard** preference **Tab key narrows completions** is selected. This is particularly useful for large lists. For example, type `cam` and press **Tab** to see the possible completions. There is a scroll bar with the list because there are too many completions to be seen at once.



Type `p` and press **Tab** again. The Editor narrows the list, showing only all possible `camp` completions.



Continue narrowing the list in the same way. For the preceding example, type `o` and press **Tab** to narrow the list further. Press **Enter** or **Return** to select an item, which completes the name at the prompt.

Tab Completion for Structures

For structures that are in the current workspace, after the period separator, press **Tab**. For example, type

```
mystruct.
```

and press **Tab** to display all fields of `mystruct`. If you type a structure and include the start of a unique field after the period, pressing **Tab** completes that structure and field entry.

For example, type

```
mystruct.n
```

and press **Tab**, which completes the entry `mystruct.name`, where `mystruct` is in the current workspace and contains no other fields that begin with `n`.

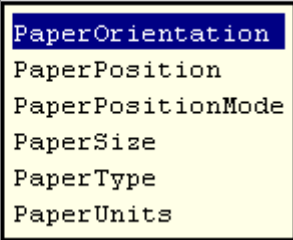
Tab Completion for Properties

Complete property names for figures in the current workspace using tab completion, as in this graphics example. Here, `f` is a figure. Type

```
set(f, 'pap
```

and press **Tab**. The Editor displays

```
set(f, 'paper
```



- PaperOrientation
- PaperPosition
- PaperPositionMode
- PaperSize
- PaperType
- PaperUnits

Select a property from the list. For example, type

```
u
```

and press **Enter**. The Editor completes the property, including the closing quote.

```
set(f, 'paperunits'
```

Continue adding to the statement, as in this example,

```
set(f, 'paperunits', 'c
```

and press **Tab**. The Editor automatically completes the property

```
set(f, 'paperUnits', 'centimeters'
```

because `centimeters` is the only possible completion.

Using Tab for Spacing

If the preference for tab completion is selected, and you want to use the **Tab** key to add spacing within your statements also, add a space before pressing **Tab**. For example, to create this statement

```
if a=mate    %test input value
```

add a space after `mate` and then press **Tab**. If you do not include the space, the following happens instead:

```
if a=material
```

This is because the tab completion feature automatically causes `mate` to complete as the `material` function.

Alternatively, turn off the tab completion preference to use **Tab** for spacing in the Editor.

Making MATLAB Code Files More Readable

In this section...

“Syntax Highlighting” on page 9-53

“Indenting” on page 9-53

“Function Indenting” on page 9-55

“Line and Column Numbers” on page 9-55

“Highlight Current Line” on page 9-55

“Right-Side Text Limit” on page 9-56

“Class, Function, or Subfunction” on page 9-57

“Code Folding — Expanding and Collapsing File Constructs” on page 9-57

“Displaying Two Parts of a File Simultaneously” on page 9-67

Note You can specify the default behaviors for some of these features—see “Specifying Options for MATLAB Using Preferences” on page 2-124.

Syntax Highlighting

Some entries appear in different colors to help you better find matching elements, such as `if/else` statements. Similarly, unterminated strings have a different color than terminated strings. This is called syntax highlighting and is used in the Command Window, Command History, and the Editor. For more information, see the Command Window documentation for “Highlighting Syntax to Help Ensure Correct Entries” on page 3-23.

When you paste or drag a selection from the Editor to another application, such as Microsoft Word, the pasted text maintains the syntax highlighting colors and font characteristics from the Editor. MATLAB software pastes the selection to the Clipboard in RTF format, which many Microsoft Windows and Macintosh applications support.

Indenting

This section describes how to apply indenting to code in the Editor.

Enabling Automatic Indenting

Set an indenting preference so that program control entries indent automatically to make reading statements such as `while` loops easier:

- 1** Select **File > Preferences > Editor/Debugger > Language**.
- 2** From the **Language** drop-down menu, select a language.
- 3** Under **Indenting**, select **Apply smart indenting while typing**.
- 4** Click **OK**.

If you want to indent manually, clear **Apply smart indenting while typing** in step 3. For more information about indenting preferences, click **Help** in the Preferences dialog box. Specify the indenting size and other options by selecting **File > Preferences > Editor/Debugger > Tab**.

Manually Applying Indenting

Indenting can help you to read the code sequence. Manually apply smart indenting, so that lines indent if they start with a keyword function or if they follow lines containing certain keyword functions:

- 1** Select the lines to indent.
- 2** Select **Text > Smart Indent** or right-click and select **Smart Indent** from the context menu.
- 3** Adjust the text by doing one of the following:
 - Move the current or selected lines further to the left, by selecting **Text > Decrease Indent**.
 - Move the current or selected lines further to the right, by selecting **Text > Increase Indent**.

Additional methods for indenting a line are:

- Press the **Tab** key at the start of a line.
- Select a line or group or lines and press the **Tab** key.

Press **Shift+Tab** to decrease the indent for the selected lines. This works differently if you select the Editor/Debugger **Tab** preference for **Emacs-style Tab key smart indenting**—when you position the cursor in any line or select a group of lines and press **Tab**, the lines indent according to smart indenting practices.

For more information about manual indenting, select **File > Preferences > Editor/Debugger > Tab**, and then click **Help**.

Function Indenting

You can select from three indenting options when you enter a subfunction or a nested function (a function within a function) in the Editor. For details, see “Function Indenting Format” on page 9-25.

Line and Column Numbers

Line numbers display along the left side of the Editor window. You can elect not to show the line numbers using preferences. For details, select **File > Preferences > Editor/Debugger > Display**, and then click **Help**.

The line and column numbers for the current cursor position display in the far right side of the status bar in the Editor.

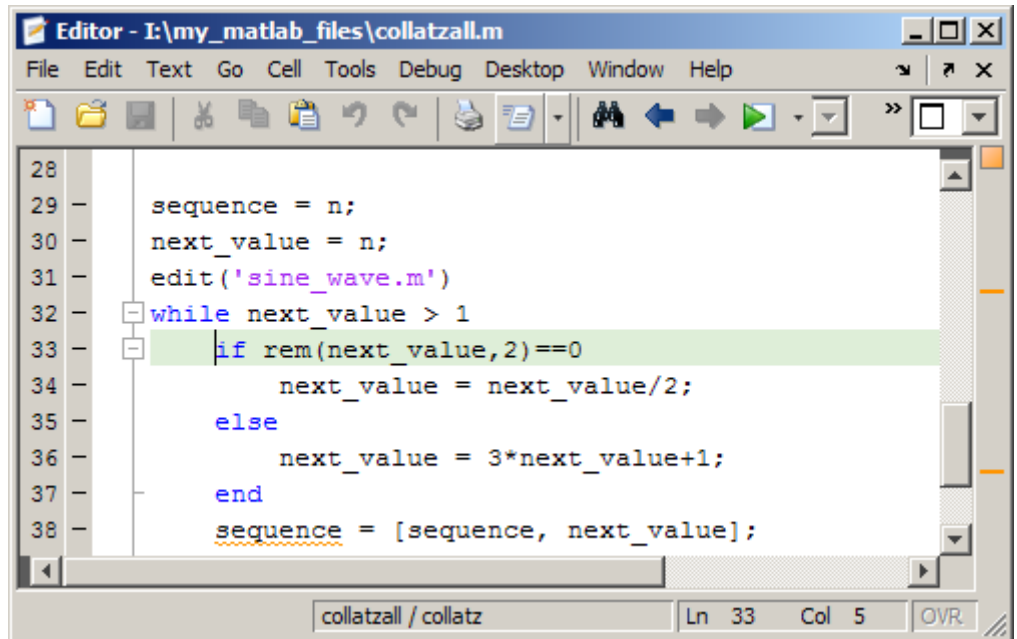
Highlight Current Line

You can set a preference to highlight the current line, that is the line with the caret (also called the cursor). This helps you see where copied text will be inserted when you paste, for example.

To highlight the current line:

- 1 Select **File > Preferences > Editor/Debugger > Display**.
- 2 Under **General display options**, select **Highlight current line**.
- 3 If you want a highlight color other than green, click the color palette and choose a different color.

In the following image, the current line is highlighted in the default highlighting color—green.



Right-Side Text Limit

By default, a light gray vertical line (rule) appears at column 75 in the Editor, indicating where a line becomes wider than desired. This is useful, for example, if you want to keep each line below a limit imposed by another text editor in which you intend to view the code. You can change the width and color of the vertical line, as well as the column number at which the vertical line appears. If you want, you can hide the vertical line. For more information, select **File > Preferences > Editor/Debugger > Display**, and then click **Help**.

Note This limit is a visual cue only and does not prevent text from exceeding the limit. For information on setting a value to wrap comment text at a specified column number automatically, see “Comment Formatting” on page 9-22.

Class, Function, or Subfunction

The right side of the Editor status bar shows the class, function, or subfunction where the cursor is currently placed, depending on the type of file you are viewing:

- Class file — The name of the class followed by the name of the current function (if any) that the cursor is within. This is true regardless of the type of function in which the cursor is placed (nested, in a methods block, outside a classdef file, and so on).
- Function file — The name of the main function followed by the name of the current function the cursor is within (if any). This is true regardless of the type of function in which the cursor is placed (subfunction or nested).

Code Folding — Expanding and Collapsing File Constructs

Code folding is the ability to expand and collapse certain MATLAB programming constructs. This improves readability when a file contains numerous subfunctions or other blocks of code that you want to hide when you are not currently working with that part of the file.

You can set preferences to enable or disable the ability to expand and collapse the following MATLAB programming constructs:

- Help block comments
- Cells used for running sections of code and publishing code
- Class code
- Class enumeration blocks
- Class event blocks
- Class method blocks

- Class properties blocks
- For and parfor blocks
- Function and class help
- Function code
- If/else blocks
- Single program, multiple data (spmd) blocks
- Switch/case blocks
- Try/catch blocks
- While blocks

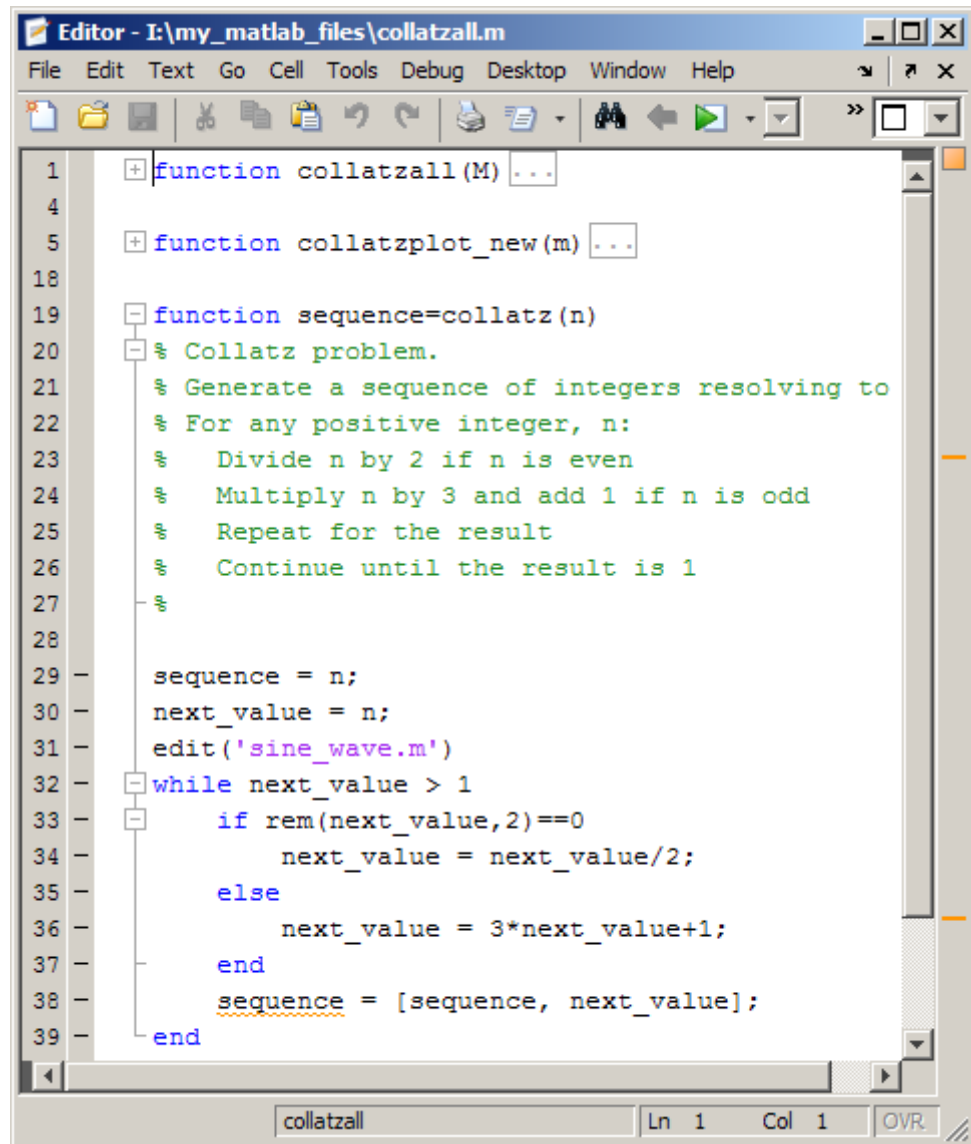
By default, code folding is enabled for all programming constructs except if/else blocks and switch/case blocks. Select **File > Preferences > Editor/Debugger > Code Folding**, and then click **Help** for details on setting preferences.

When you fold a construct, all the code associated with that construct is collapsed such that the Editor displays only the first line of the construct prepended by the plus sign (+) and appended with an ellipsis (...) to indicate there is more code. When you expand a construct, all the code associated with that construct appears and the first line of the construct is prepended with a minus sign (-).

To open the code used in the images in this section, enter the following in the Command Window:

```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...  
'examples','collatzall.m'))
```

The following image shows the `collatzall` and `collatzplot_new` functions collapsed and the `collatz` function code expanded.




```

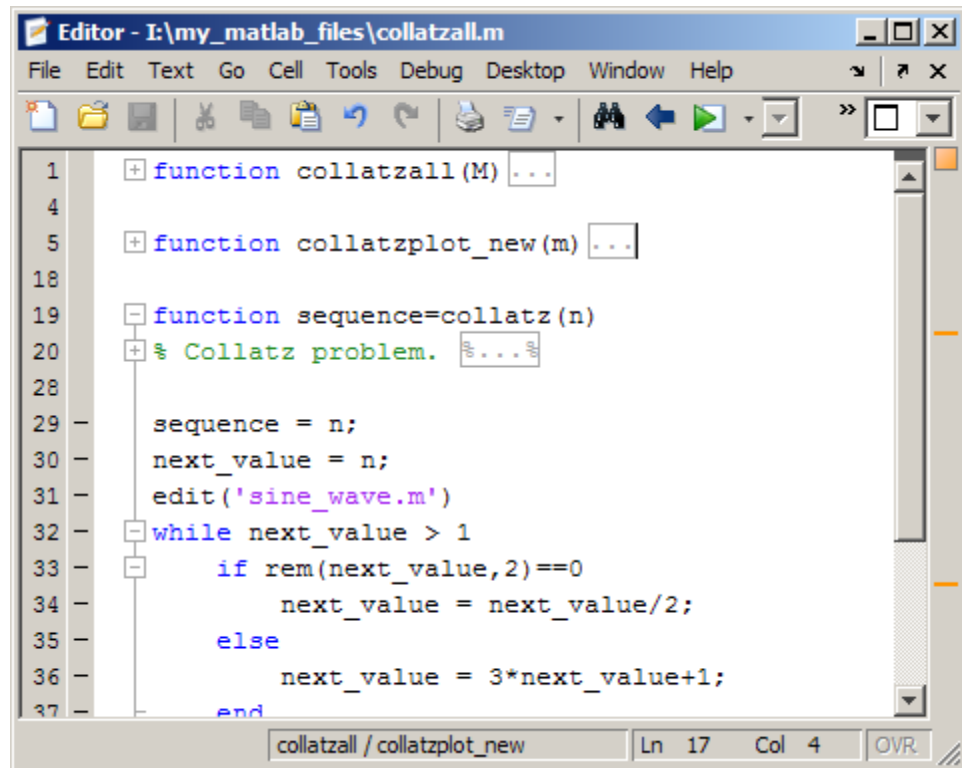
1  [+ function collatzall(M) ...
4
5  [+ function collatzplot_new(m) ...
18
19 [- function sequence=collatz(n)
20 [- % Collatz problem.
21   % Generate a sequence of integers resolving to
22   % For any positive integer, n:
23   %   Divide n by 2 if n is even
24   %   Multiply n by 3 and add 1 if n is odd
25   %   Repeat for the result
26   %   Continue until the result is 1
27   %
28
29   sequence = n;
30   next_value = n;
31   edit('sine_wave.m')
32 [- while next_value > 1
33   [- if rem(next_value,2)==0
34     next_value = next_value/2;
35   else
36     next_value = 3*next_value+1;
37   end
38   sequence = [sequence, next_value];
39 end

```


collatzall Ln 1 Col 1 OVR

When you expand a function or class, but collapse its associated help block code, the Editor displays all the function or class code and just the first line of the help code. The first line of the help code ends with a commented ellipsis


 to indicate there is additional help code, as shown in the following image.



To expand code for a construct that is currently collapsed, do one of the following:

- Click the plus sign  to the left of the construct that you want to expand.
- Place your cursor in the code that you want to expand, right-click, and then select **Code Folding > Expand** from the context menu.

To collapse code for a construct that is currently expanded, do one of the following:


- Click the minus sign  to the left of the construct that you want to collapse.

- Place your cursor in the code that you want to collapse, right-click, and then select **Code Folding > Collapse** from the context menu.

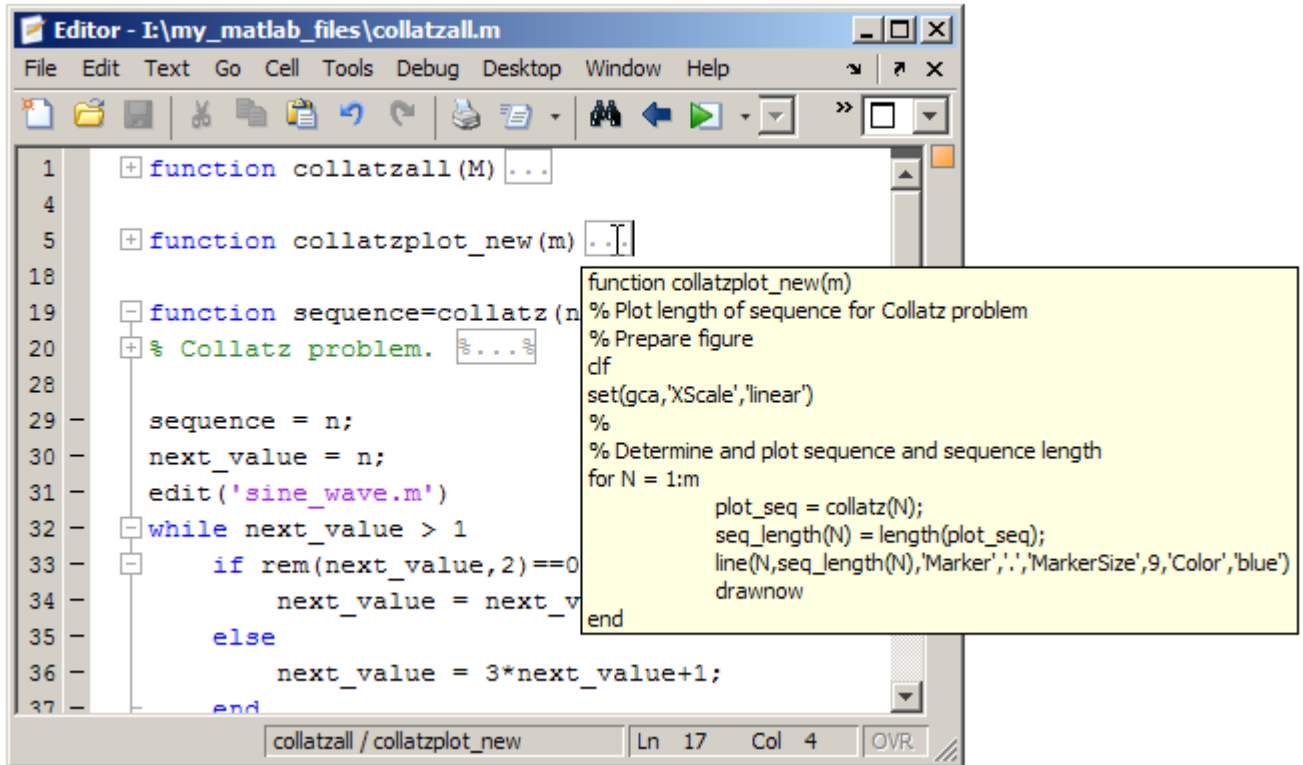
To expand or collapse all of the code in a file, place your cursor anywhere within the file, right-click, and then select **Code Folding > Expand All** or **Code Folding > Collapse All** from the context menu.

For information on the structure of a MATLAB code file, including a description of a function definition line and an H1 line, see Basic Parts of a Program File in the Programming Fundamentals documentation.

Viewing Folded Code in a Tooltip

You can view code that is currently folded by positioning the pointer over its ellipsis . The code appears in a tooltip. This lets you quickly view the code without unfolding it.

The following image shows the tooltip that appears when you place the pointer over the ellipsis on line 6 of `collatzall.m` when the `collatzplot_new` function is folded.



Code Folding Behavior and Preferences

- “Changing Code Folding Preferences” on page 9-63
- “Default Code Folding Behavior for Files Created Before MATLAB Version 7.5” on page 9-63
- “Reopening a File After Collapsing Code Folds” on page 9-63
- “Copying and Pasting Folded Code” on page 9-63
- “Printing Files with Collapsed Code” on page 9-63
- “Behavior When Functions Have No Explicit End Statement” on page 9-63
- “Effect of Syntax Errors on Code Folding” on page 9-66

Changing Code Folding Preferences. To change the current code folding settings, select **File > Preferences > Editor/Debugger > Code Folding**. If needed, click **Help** for assistance.

Default Code Folding Behavior for Files Created Before MATLAB Version 7.5. By default, the first time you open a MATLAB code file that existed before MATLAB Version 7.5 (R2007b) using MATLAB Version 7.5 (R2007b) or later, code folding is enabled and all constructs are expanded.

Reopening a File After Collapsing Code Folds. Constructs that are collapsed when you close a file remain collapsed when you reopen the file.

Copying and Pasting Folded Code. If you copy a collapsed construct from one region of a file and paste it in another region, the construct is expanded in the pasted location.

Printing Files with Collapsed Code. If you print a file with one or more collapsed constructs, those constructs are expanded in the printed version of the file.

Behavior When Functions Have No Explicit End Statement. If you enable code folding for functions and a function in your code does not end with an explicit end statement, you see the following behavior:

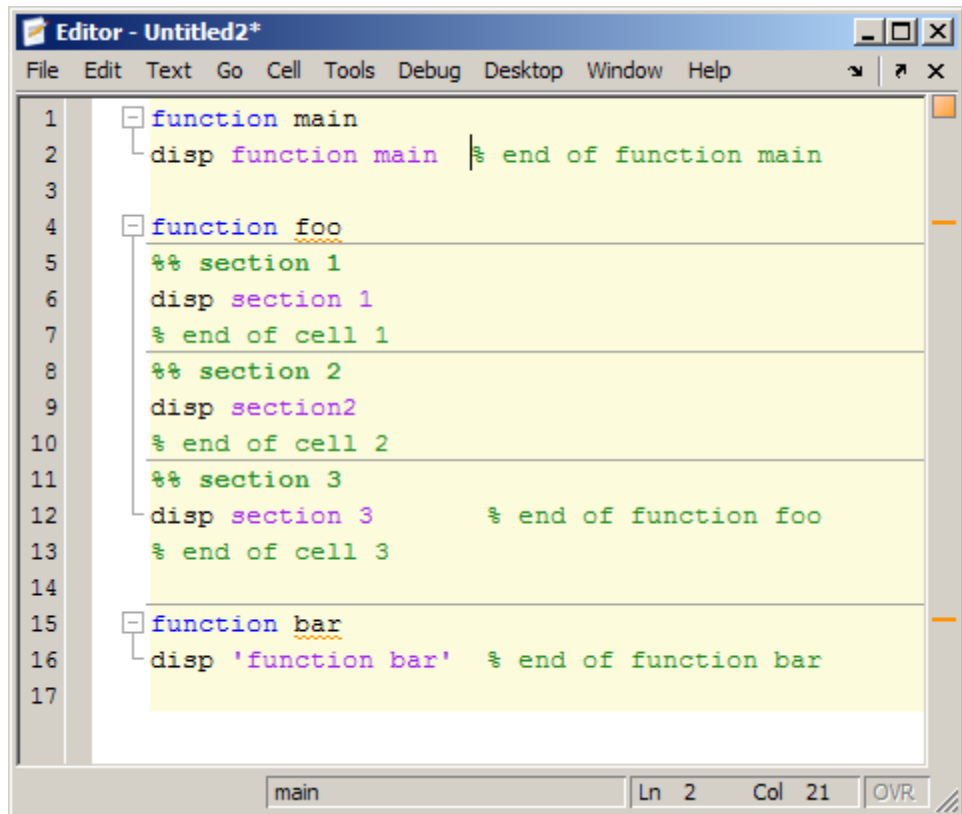
- If a line containing only comments appears at the end of such a function, then the Editor does not include that line when folding the function. MATLAB does not include trailing white space and comments in a function definition that has no explicit end statement.

Code Folding Enabled for Function Code Only on page 9-64 illustrates this behavior. Line 13 is excluded from the fold for the `foo` function.

- If a fold for a cell overlaps the function code, then the Editor does not show the fold for the overlapping cell.

The three figures that follow illustrate this behavior. The first two figures, Code Folding Enabled for Function Code Only on page 9-64 and Code Folding Enabled for Cells Only on page 9-65 illustrate how the code folding appears when you enable it for function code only and then cell code only, respectively. The last figure, Code Folding Enabled for Both Functions and Cells on page 9-66, illustrates the effects when code folding is enabled for

both. Because the fold for cell 3 (lines 11–13) overlaps the fold for function foo (lines 4–12), the Editor does not display the fold for cell 3.

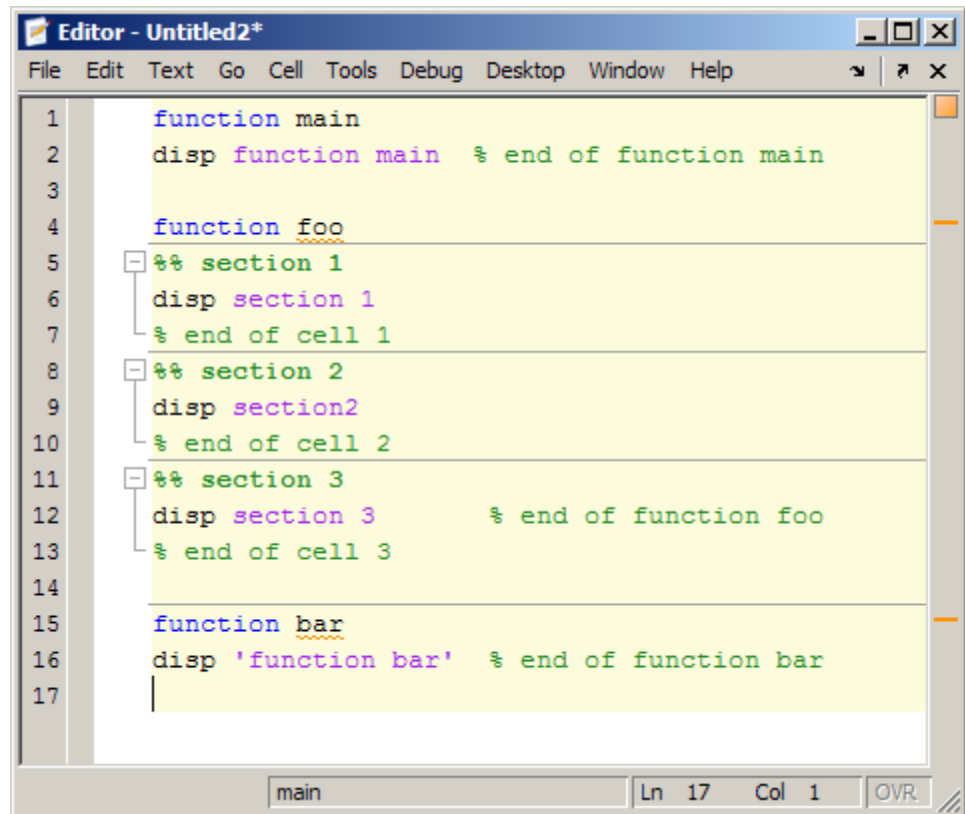


The screenshot shows the MATLAB Editor window titled "Editor - Untitled2*". The code is as follows:

```
1 function main
2   disp function main   % end of function main
3
4 function foo
5   %% section 1
6   disp section 1
7   % end of cell 1
8   %% section 2
9   disp section2
10  % end of cell 2
11  %% section 3
12  disp section 3       % end of function foo
13  % end of cell 3
14
15 function bar
16  disp 'function bar'  % end of function bar
17
```

The code is displayed with syntax highlighting. The function definitions are folded, indicated by minus signs in the left margin. The fold for function foo (lines 4-12) overlaps the fold for cell 3 (lines 11-13), so the fold for cell 3 is not visible. The status bar at the bottom shows "main", "Ln 2", "Col 21", and "OVR".

Code Folding Enabled for Function Code Only

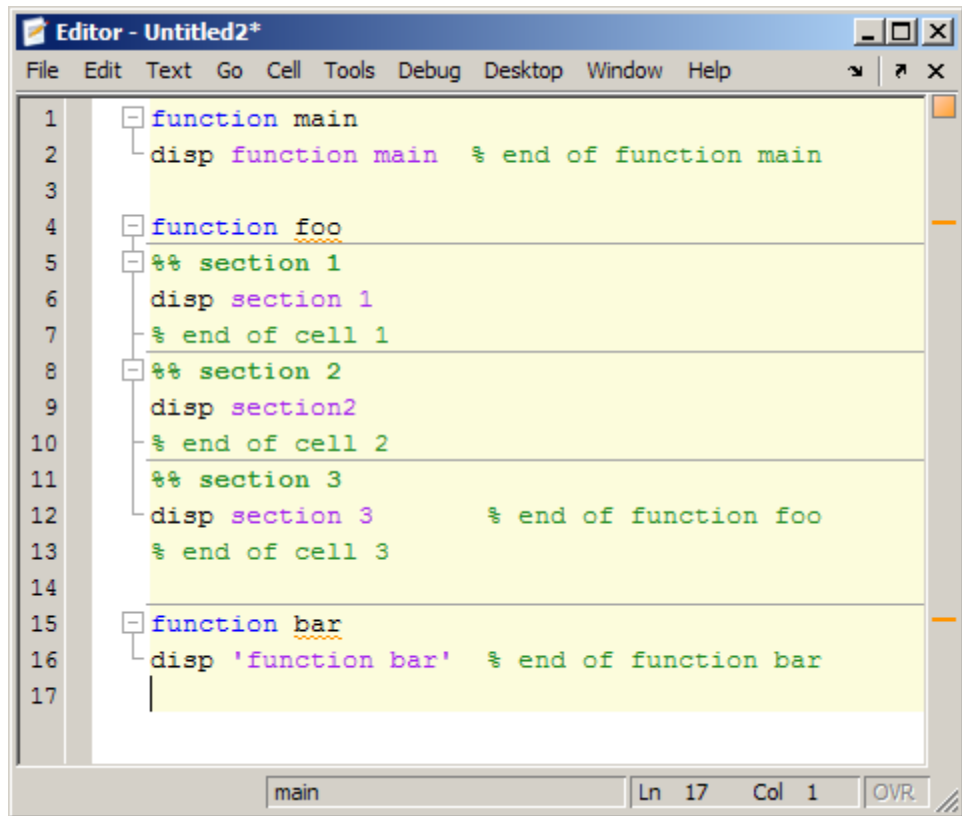


The screenshot shows the MATLAB Editor window titled "Editor - Untitled2*". The menu bar includes File, Edit, Text, Go, Cell, Tools, Debug, Desktop, Window, and Help. The code is as follows:

```
1 function main
2 disp function main % end of function main
3
4 function foo
5 %% section 1
6 disp section 1
7 % end of cell 1
8 %% section 2
9 disp section2
10 % end of cell 2
11 %% section 3
12 disp section 3 % end of function foo
13 % end of cell 3
14
15 function bar
16 disp 'function bar' % end of function bar
17
```

The code is displayed with a light yellow background. The function definitions are in blue, comments are in green, and the `disp` commands are in purple. The code is folded into three sections: "section 1" (lines 5-7), "section 2" (lines 8-10), and "section 3" (lines 11-13). Each section is collapsed, indicated by a minus sign in a small square on the left margin. The status bar at the bottom shows "main", "Ln 17 Col 1", and "OVR".

Code Folding Enabled for Cells Only



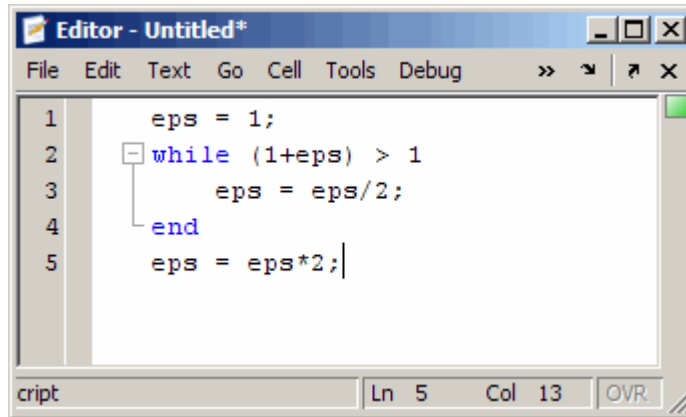
The screenshot shows the MATLAB Editor window titled "Editor - Untitled2*". The window contains MATLAB code with code folding enabled. The code is as follows:

```
1 function main
2     disp function main % end of function main
3
4 function foo
5     %% section 1
6     disp section 1
7     % end of cell 1
8     %% section 2
9     disp section2
10    % end of cell 2
11    %% section 3
12    disp section 3 % end of function foo
13    % end of cell 3
14
15 function bar
16     disp 'function bar' % end of function bar
17
```

The code is displayed with a yellow background. The function definitions are folded, indicated by minus signs in small boxes to the left of the function names. The code is also organized into cells, with each cell having its own folding indicator. The status bar at the bottom shows "main", "Ln 17", "Col 1", and "OVR".

Code Folding Enabled for Both Functions and Cells

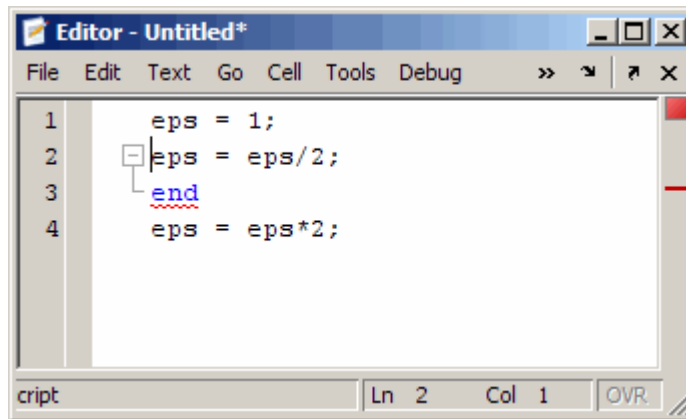
Effect of Syntax Errors on Code Folding. If your code contains syntax errors, the code folding indicators might appear to be in the wrong location. For example, suppose your code currently appears as shown in the first figure that follows. If you delete the `while` statement, it introduces a syntax error at line 3, as shown in the second figure that follows. Notice that the minus sign remains in the same location it held for the syntactically correct code. After you correct the syntax error, the Editor adjusts and displays the code folding indicators appropriately.



The image shows a MATLAB Editor window titled "Editor - Untitled*". The menu bar includes File, Edit, Text, Go, Cell, Tools, and Debug. The code editor contains the following text:

```
1     eps = 1;  
2     while (1+eps) > 1  
3         eps = eps/2;  
4     end  
5     eps = eps*2;|
```

The status bar at the bottom indicates the file name "cript", the current position "Ln 5 Col 13", and the view mode "OVR".



The image shows a MATLAB Editor window titled "Editor - Untitled*" with a split view. The menu bar includes File, Edit, Text, Go, Cell, Tools, and Debug. The code editor contains the following text:

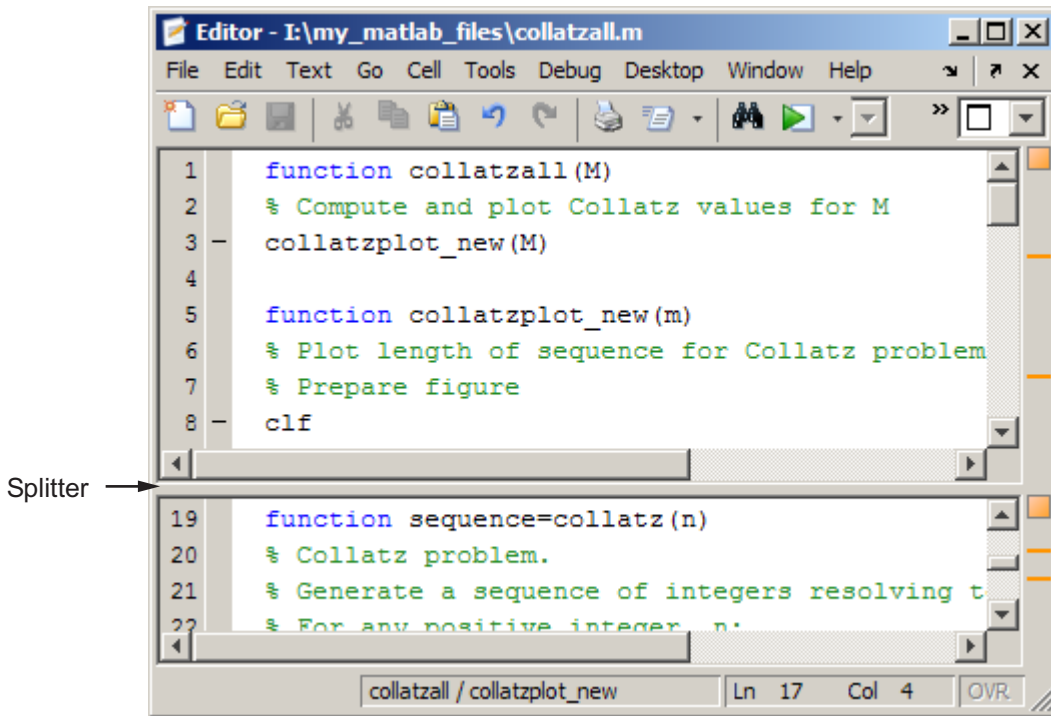
```
1     eps = 1;  
2     eps = eps/2;  
3     end  
4     eps = eps*2;
```

The status bar at the bottom indicates the file name "cript", the current position "Ln 2 Col 1", and the view mode "OVR".

Displaying Two Parts of a File Simultaneously

You can simultaneously display two different parts of a file in the Editor by splitting the screen display, as shown in the image that follows. This feature makes it easy to compare different lines in a file or to copy and paste from one part of a file to another.

See also “Positioning Documents” on page 2-24 for instructions on displaying multiple documents simultaneously.

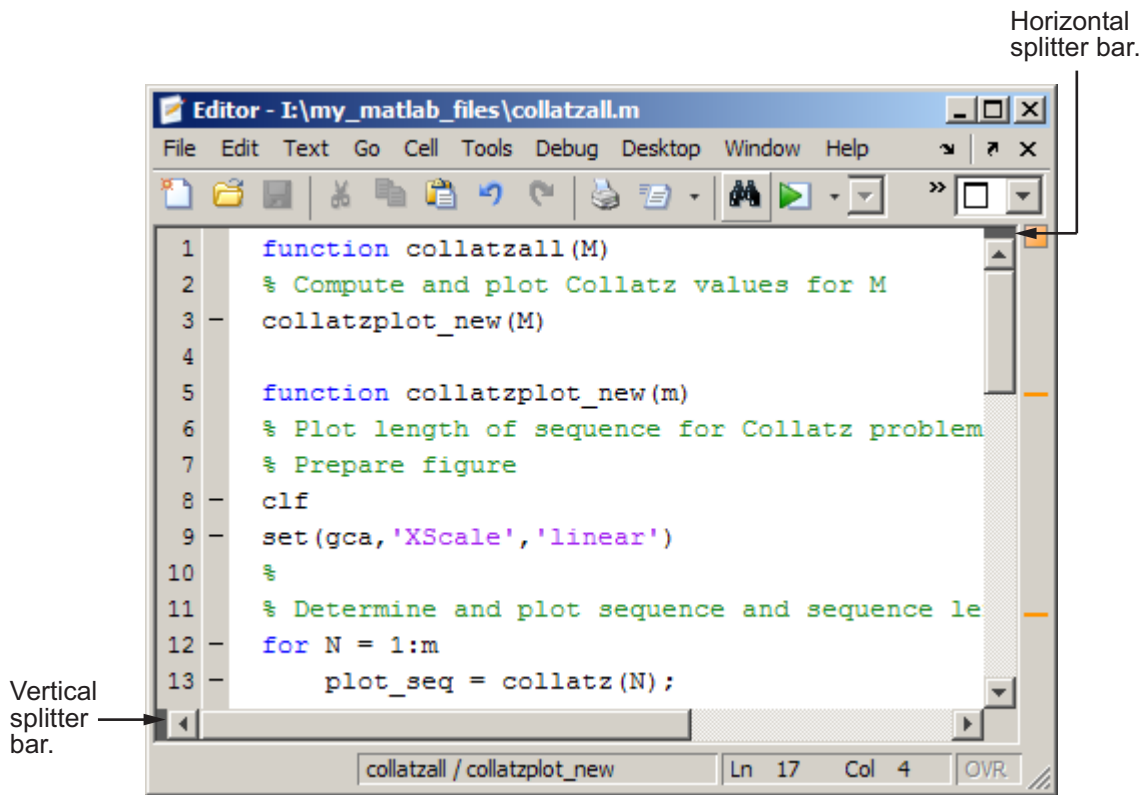


Splitting the Screen Display

The following table describes the various ways you can split the Editor and manipulate the split-screen views. When you open a document, it opens unsplit, regardless of its split status it had when you closed it.

Operation	Instructions
Split the screen horizontally.	<p>Do either of the following</p> <ul style="list-style-type: none"> • Select Window > Split Screen > Top/Bottom. • If there is a vertical scroll bar, as shown in the illustration that follows, drag the splitter bar down.

Operation	Instructions
Split the screen vertically.	<p>Do either of the following:</p> <ul style="list-style-type: none"> • Select Window > Split Screen > Left/Right. • If there is a horizontal scroll bar, as shown in the illustration that follows, drag the splitter bar from the left of the scroll bar.
Specify the active view.	<p>Do either of the following:</p> <ul style="list-style-type: none"> • Select Window > Split Screen > Switch Focus. • Click in the view you want to make active. <p>Updates you make to the document in the active view are also visible in the other view.</p>
Resize the views.	Drag the splitter.
Remove the splitter	<p>Do one of the following:</p> <ul style="list-style-type: none"> • Drag the splitter to an edge of the window. • Double-click the splitter. • Select Window > Split > Screen > Off.




Navigating an Open File in the Editor



In this section...
“Navigating to a Specific Location” on page 9-71
“Using Bookmarks” on page 9-75
“Navigating Backward and Forward in Files” on page 9-75
“Opening a File or Variable from Within a File” on page 9-76


Note See also “Finding Text in Files” on page 9-78.







Navigating to a Specific Location

This table summarizes the steps for navigating to a specific location within a file open in the Editor. In some cases, different sets of steps are available for navigating to a particular location. Choose the set that works best with your workflow.

Go To	Steps	Notes
Line Number	<ol style="list-style-type: none"> 1 Select Go > Go To 2 Specify the line to which you want to navigate. 	None
Function definition	<ol style="list-style-type: none"> 1 Click the Show Functions button  on the Editor toolbar. 2 From the list that appears, select the subfunction or nested function to which you want to navigate. <p>Includes subfunctions and nested functions</p>	<p>Includes subfunctions and nested functions</p> <p>For both class and function files, the functions list in alphabetical order—except that in function files, the name of the main function always appears at the top of the list.</p>

Go To	Steps	Notes
	<p>1 Select Go > Go To</p> <p>2 Select the subfunction or nested function to which you want to navigate.</p> <p>Includes subfunctions and nested functions</p>	<p>Functions list in alphabetical order. To order them by their position in the file, click the Line column heading.</p> <p>The list shows lines in the current file that begin with a function statement. The list does not include functions that the file calls.</p>
	<p>1 In the Current Folder browser, click the name of the file open in the Editor.</p> <p>2 Click the up arrow  at the bottom of Current Folder browser to open the detail panel.</p> <p>3 In the detail panel, double-click the function icon  corresponding to the title of the function or subfunction to which you want to navigate.</p>	<p>Functions list in order of appearance within your file.</p>

Go To	Steps	Notes
Function reference	<ol style="list-style-type: none"> 1 Click in any instance of the function name. 2 Press Alt +Up or Alt+Down to go to the next or previous function reference, respectively. 	<p>Alt +Up and Alt+Down are the default keyboard shortcuts for the actions Go to Previous Underline or Highlight and Go to Next Underline or Highlight, respectively.</p> <p>For more information, see “Performing Desktop Actions Using Keyboard Shortcuts” on page 2-69.</p>
Variable reference	<ol style="list-style-type: none"> 1 Click in any instance of the variable name. 2 Press Alt +Up or Alt+Down to go to the next or previous variable reference, respectively. 	
Code Analyzer Message	<p>Press Alt +Up or Alt+Down to go to the next or previous code analyzer message, respectively.</p>	
Code Cell	<ol style="list-style-type: none"> 1 Click the Show Cell Titles button  on the Editor Cell Mode toolbar. 2 From the list that appears, select the title of the code cell to which you want to navigate. 	<p>For more information, see “Defining Code Cells” on page 9-178</p>
	<ol style="list-style-type: none"> 1 Select Go > Go To 2 Select the title of the code cell to which you want to navigate. 	
	<ol style="list-style-type: none"> 1 In the Current Folder browser, click the name of the file that is open in the Editor. 	


Go To	Steps	Notes
	<ol style="list-style-type: none"> 2 Click the up arrow  at the bottom of Current Folder browser to open the detail panel. 3 In the detail panel, double-click the cell icon  corresponding to the title of the cell to which you want to navigate. 	
Property	<ol style="list-style-type: none"> 1 In the Current Folder browser, click the name of the file that is open in the Editor. 2 Click the up arrow  at the bottom of Current Folder browser to open the detail panel. 3 On the detail panel, double-click the property icon  corresponding to the name of the property to which you want to navigate. 	For more information, see “Properties — Storing Class Data”
Method	<ol style="list-style-type: none"> 1 In the Current Folder browser, click the name of the file that is open in the Editor. 2 Click the up arrow  at the bottom of Current Folder browser to open the detail panel. 3 In the detail panel, double-click the icon  corresponding to the name of the method to which you want to navigate. 	For more information, see “Methods — Defining Class Operations”
Bookmark	Select Go > Next Bookmark , or Go > Previous Bookmark .	For information on setting and clearing bookmarks, see “Using Bookmarks” on page 9-75.

Using Bookmarks

You can set a bookmark at any line in a file in the Editor so you can quickly navigate to the bookmarked line. This is particularly useful in long files. For example, suppose while working on a line, you want 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 use the bookmark to return.

To set a bookmark:

- 1 Position the cursor anywhere on the line.
- 2 Select **Go > Set/Clear Bookmark**.

A bookmark icon  appears to the left of the line.

To clear a bookmark, position the cursor anywhere on the line and select **Go > Select/Clear Bookmark**.

MATLAB does not maintain bookmarks after you close a file.

Navigating Backward and Forward in Files

To access lines in a file in the same sequence that you previously navigated or edited them, do one of the following:

- Select **Go > Back** or click .
- Select **Go > Forward** or click .

Interrupting the Sequence of Go Back and Go Forward

The **Go > Back** or **Go > Forward** feature sequence is interrupted if you:

- 1 Select **Go > Back**.
- 2 Select **Go > Forward**.
- 3 Edit a line or navigate to another line using the list of features described in “Navigating to a Specific Location” on page 9-71.

You can still go to the lines preceding the interruption point in the sequence, but you cannot go to any lines after that point. Any lines you edit or navigate to after interrupting the sequence are added to the sequence after the interruption point.

For example:

- 1 Open a file.
- 2 Edit line 2, line 4, and line 6.
- 3 Use **Go > Back** to return to line 4, and then to return to line 2.
- 4 Use **Go > Forward** to return to lines 4 and 6.
- 5 Use **Go > Back** to return to line 1.
- 6 Edit at 3.

This interrupts the sequence. You can no longer use **Go > Forward** to return to lines 4 and 6. You can, however, use **Go > Back** to return to line 1.

Opening a File or Variable from Within a File

You can open a subfunction, function, file, variable, or Simulink model from within a file in the Editor. Position the cursor on the name, and then right-click and select **Open selection** from the context menu. Based on what the item is, the Editor performs a different action, as described in this table.

Item	Action
Subfunction	Navigates to the subfunction within the current file, if that file is a MATLAB code file. If no subfunction by that name exists in the current file, the Editor runs the <code>open</code> function on the selection, which opens the selection in the appropriate tool.
Text file	Opens in the Editor.
Figure file (.fig)	Opens in a figure window.

Item	Action
MATLAB variable that is in the current workspace	Opens in the Variable Editor.
Model	Opens in Simulink.
Other	If the selection is some other type, Open selection looks for a matching file in a private folder in the current folder and performs the appropriate action.

Finding Text in Files

In this section...

“Finding Any Text in the Current File” on page 9-78

“Finding and Replacing Functions or Variables in the Current File” on page 9-78

“Finding and Replacing Any Text” on page 9-80

“Finding Text in Multiple File Names or Files” on page 9-81

“Function Alternative for Finding Text” on page 9-82

“Performing an Incremental Search in the Editor” on page 9-82

Finding Any Text in the Current File

- 1 Within the current file, select the text you want to find.
- 2 From the **Edit** menu, select **Find Selection**.

The next occurrence of that text is selected.

- 3 Select **Find Selection** again (or **Find Next**) to continue finding more occurrences of the text.

To find the previous occurrence of selected text (find backwards) in the current file, select **Find Previous** from the **Edit** menu. The previous occurrence of the text is selected. Repeat to continue finding the previous occurrences of the text.

Finding and Replacing Functions or Variables in the Current File

To search for references to a particular function or variable, use the function and variable highlighting feature. This feature is more efficient than using the text finding tools. Function and variable highlighting indicates only references to the function or variable, not other occurrences. For instance it does not find references to the function or variable in comments. Furthermore, variable highlighting only includes references to the *same* variable. That is, if

two variables use the same name, but are in different scopes, highlighting one does not cause the other to highlight.

- 1** Select **File > Preferences > Colors > Programming Tools**.
- 2** Under **Variable and function highlighting colors**, select **Automatically highlight** and **Nonlocal variables**, and then click **Apply**.
- 3** In a file open in the Editor, click an instance of the variable you want to find throughout the file.

MATLAB indicates all occurrences of that variable within the file by:

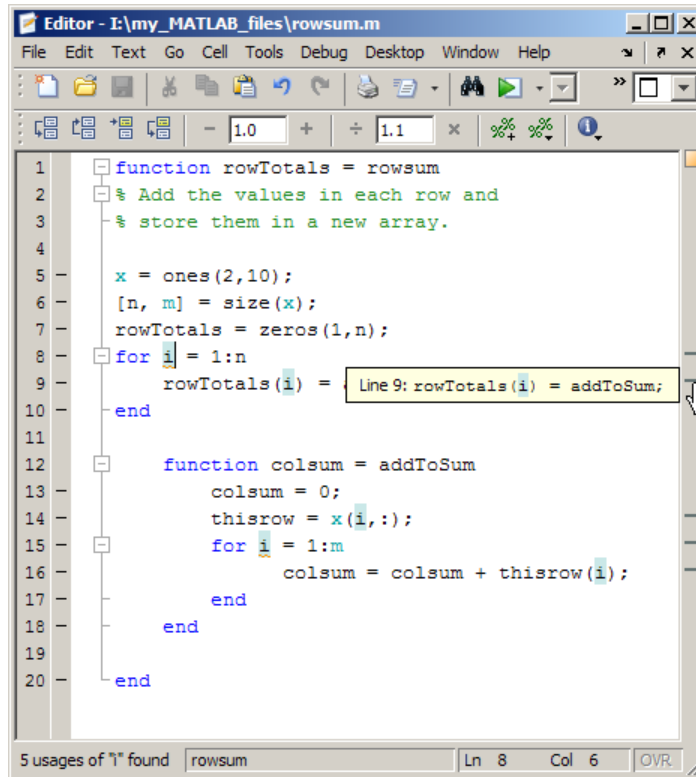
- Highlighting them in teal blue (by default) throughout the file.
- Adding a marker for each in the indicator bar

If a code analyzer indicator and a variable indicator appear on the same line in a file, the marker for the variable takes precedence.

- Specifying the number of usages it finds in the status bar
- 4** Hover over a grey maker in the indicator bar to see the line it represents.
 - 5** Click a grey marker in the indicator bar to navigate to that occurrence of the variable.

Replace an instance of a function or variable by editing the occurrence at a line to which you have navigated.

The following image shows an example of how the Editor looks with variable highlighting enabled. In this image, the variable `i` appears highlighted in sky blue, the indicator bar contains three variable markers, and the status bar (on the lower left edge of the tool) indicates 5 usages of `i` found.



```
1 function rowTotals = rowsum
2 % Add the values in each row and
3 % store them in a new array.
4
5 x = ones(2,10);
6 [n, m] = size(x);
7 rowTotals = zeros(1,n);
8 for i = 1:n
9     rowTotals(i) = addToSum;
10 end
11
12 function colsum = addToSum
13     colsum = 0;
14     thisrow = x(i,:);
15     for i = 1:m
16         colsum = colsum + thisrow(i);
17     end
18 end
19
20 end
```

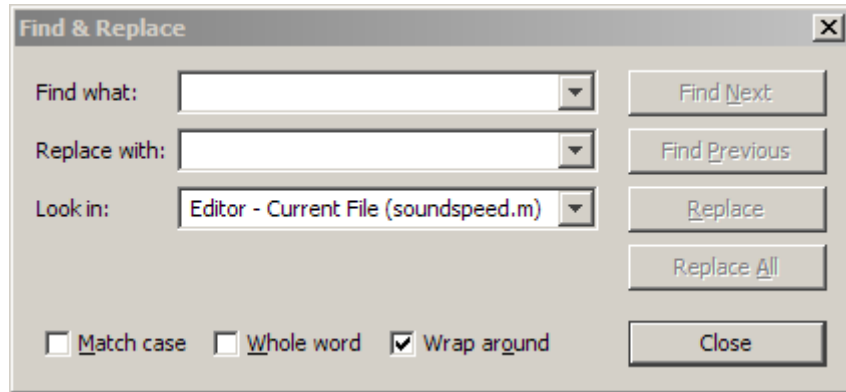
5 usages of "*" found | rowsum | Ln 8 Col 6 | OVR

Finding and Replacing Any Text

To search for, and optionally replace specified text within a file:

- 1 Select **Edit > Find and Replace**.

The Find and Replace dialog box opens.



2 Complete the resulting Find & Replace dialog box:

- Type the text you want to find in the **Find what** field.
- Optionally, type the text that you want to use instead of the found text in the **Replace with** field.
- Optionally, search for the specified text in other MATLAB desktop tools by changing the selection in the **Look in** field.

3 Click **Find Next**, **Find Previous**, **Replace**, or **Replace All**.

4 Select **Wrap around** to have MATLAB continue the search after reaching the beginning or end of the file.

When a search for **Find Next** reaches the end of the file, or when a search for **Find Previous** reaches the beginning of the file, the MATLAB software beeps.

Use keyboard shortcuts to continue finding the specified text even after closing the Find & Replace dialog box. You can go to another file and find the specified text in it. For details on keyboard shortcuts, see “Displaying Keyboard Shortcuts” on page 2-75.

Finding Text in Multiple File Names or Files

To find folders and file names that include specified text, or whose contents contain specified text, select **Edit > Find Files**. For details, see “Finding Files and Folders” on page 7-27.

Function Alternative for Finding Text

Use `lookfor` to search for the specified text in the first line of help for all files with the `.m` extension on the search path.

Performing an Incremental Search in the Editor

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. In the Editor, incremental search uses the same controls as incremental search in the Command Window. For details, see “Using Incremental Search in the Command Window” on page 3-53.

Saving, Printing, and Closing Files in the Editor

In this section...
“Saving Files” on page 9-83
“Printing Files” on page 9-85
“Closing Files” on page 9-86

Saving Files

After you modify a file, an asterisk (*) follows the file name in the title bar of the Editor. This asterisk indicates that there are unsaved changes to the file.

You can perform four different types of save operations, which have various effects, as described in this table.

Save Option	Steps
Save file to disk File stays open in the Editor.	Select File > Save .
Rename file and make active Renames the file, saves it to disk, and makes it the active Editor document. The original file remains unchanged on disk.	<ol style="list-style-type: none"> 1 Select File > Save As. 2 Specify a new name, type, or both for the file, and then click Save.
Save backup copy Saves file to disk under new name. Original file remains open and unsaved.	<ol style="list-style-type: none"> 1 Select File > Save Backup. MATLAB opens the Select File for Backup dialog box. 2 Specify a name and type for the backup file, and then click Save.

Save Option	Steps
<p>Save changes to open files</p> <p>Saves changes to all open files using current file names. Prompts you to name unnamed files, and then saves them.</p> <p>All of the files remain open.</p>	<ol style="list-style-type: none"> 1 Select File > Save All. MATLAB opens the Select File for Save As dialog box for the first unnamed file. 2 Specify a name and type for the unnamed file, and then click Save. 3 Repeat step 2 until all unnamed files are saved.

Recommendations on Saving Files

MathWorks recommends that you save files you create and files from MathWorks that you edit to a folder that is not in the *matlabroot*/toolbox folder tree. If you keep your files in *matlabroot*/toolbox folders, they can be overwritten when you install a new version of MATLAB software.

At the beginning of each MATLAB session, MATLAB loads and caches in memory the locations of files in the *matlabroot*/toolbox folder tree. Therefore, if you:

- Save files to *matlabroot*/toolbox folders using an external editor, run `rehash toolbox` before you use the files in the current session.
- Add or remove files from *matlabroot*/toolbox folders using file system operations, run `rehash toolbox` before you use the files in the current session.
- Modify existing files in *matlabroot*/toolbox folders using an external editor, run `clear function-name` before you use these files in the current session.

For more information, see `rehash` or “Toolbox Path Caching in the MATLAB Program” on page 1-19.

Autosaving Files

When you modify a file in the Editor, the Editor saves a copy of the file using the same file name but with an `.asv` extension every 5 minutes. The autosave version is useful if you have system problems and lose changes you made to your file. In that event, you can open the autosave version, `filename.asv`, and then save it as `filename.m` to use the last good version of `filename`. For example, if you edit `filename.m` and do not save it for 5 minutes, MATLAB saves the file including the unsaved changes, to `filename.asv`.

You can set autosave preferences to:

- Turn the autosave feature off or on.
- Specify the number of minutes between automatic saves.
- Specify the file extension and location for autosave files.


For details, see “Setting Autosave Preferences” on page 9-35.

If you edit a file in a read-only folder and the autosave **Location** preference is the source file folder, then the Editor does not create an autosave copy of the file.

Deleting Autosave Files. By default, autosave files do not delete automatically when you delete the source file. It is best to keep autosave-to-file relationships clear and current. Therefore, when you rename or remove a file, consider deleting or renaming the corresponding autosave file.

There is a preference to **Automatically delete autosave files**. With this preference selected, when you close a file in the Editor, MATLAB automatically deletes the corresponding autosave file.

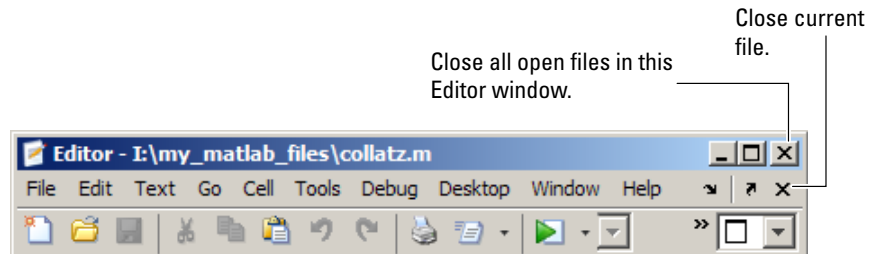
Printing Files

To print an entire file, select **File > Print**, or click the Print button  on the toolbar. To print the current selection, select **File > Print Selection**. Complete the standard print dialog box that appears.

Specify printing options for the Editor by selecting **File > Page Setup**. For example, you can specify printing with a header. For more information, see “Printing and Page Setup Options for Desktop Tools” on page 2-115.

Closing Files

To close the current file, select **Close *filename*** from the **File** menu, or click the Close box Close button in the Editor menu bar. This is different from the Close button in the title bar of the Editor, which closes all open files in that Editor window.



To close all files within the Editor, select **Window > Close Editor Documents**. This menu option does not close any files undocked from the Editor. The Editor remains open with no files in it.

If each file is open in a separate window, close all the files at once using **Close All Documents** in the **Window** menu. This closes desktop documents of all types, including Variable Editor documents.

When you close a file that has unsaved changes, you are prompted to save the file. If you do not want to be prompted, hold **Ctrl** and click the Close button. The prompt does not appear and the document closes without saving any unsaved changes.

Running MATLAB Files in the Editor

In this section...

- “Running Files with No Input Arguments in the Editor” on page 9-87
- “Using Run Configurations to Run Files with Input Arguments in the Editor” on page 9-88
- “Create and Use a Run Configuration” on page 9-88
- “Create and Execute Multiple Run Configurations for a File” on page 9-93
- “About the run_configurations.m File” on page 9-96
- “Find Configurations” on page 9-96
- “Remove Configurations” on page 9-98
- “Reassociate and Rename Configurations” on page 9-99
- “Other Ways to Run Files from the Editor” on page 9-103

Running Files with No Input Arguments in the Editor

To run a MATLAB script or function file that is open in the Editor and requires no input argument values, do one of the following:

- Click the Run button  on the toolbar.

The button’s tooltip includes the name of the file that will run, which is useful when you have multiple files open. If the file has unsaved changes, the Editor automatically saves the changes before running it.

- Select **Debug > Run *filename***.

If the file is not in the current folder or a folder on the search path, then a dialog box appears with options that enable you to run the file. You can either change the current folder to the folder containing the file, or you can add the folder containing the file to the search path.

- Select **Debug > Save File and Run *filename***.

If the file is a script, you can view the value of a variable in the file, using a *data tip*, which is like a tooltip for data. For details, see “Viewing Values as Data Tips in the Editor” on page 9-153.

Using Run Configurations to Run Files with Input Arguments in the Editor

You can provide values for MATLAB function input arguments by using a run configuration. Then, run that configuration to use the assigned values. When you are editing a function file, use a run configuration as an alternative to running the function in the MATLAB Command Window. You can associate multiple run configurations with a function file to assign different input values. MATLAB saves the run configurations between sessions to a file named `run_configurations.m`. (For details, see “About the `run_configurations.m` File” on page 9-96.)

Consider the function `collatzplot_new.m`, which computes and plots the Collatz sequence for any given positive integer. This function requires you to specify the integer as an input value. You cannot simply run `collatzplot_new.m` in the Editor because the input value is not defined. One way to specify the input value is to run the MATLAB function file in the Command Window. Run configurations allow you to run `collatzplot_new(specific value)` in the Editor.

You can also use run configurations to provide preparatory or setup information before running a MATLAB file, whether it takes input arguments or not.

Note Run configurations use the base MATLAB workspace. Therefore, a value that you assign to a variable in a run configuration overwrites the value for that variable (assuming that it currently exists) in the base workspace.

Create and Use a Run Configuration

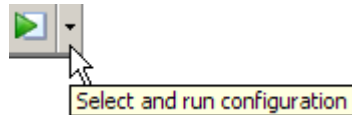
Follow these steps to create and use a run configuration for a file in the Editor. These steps specify Editor toolbar buttons, but you can also use equivalent options in the **Debug** menu.

- 1 Open the file you want to run in the Editor. For example, open `collatzplot_new.m` by running the following command:

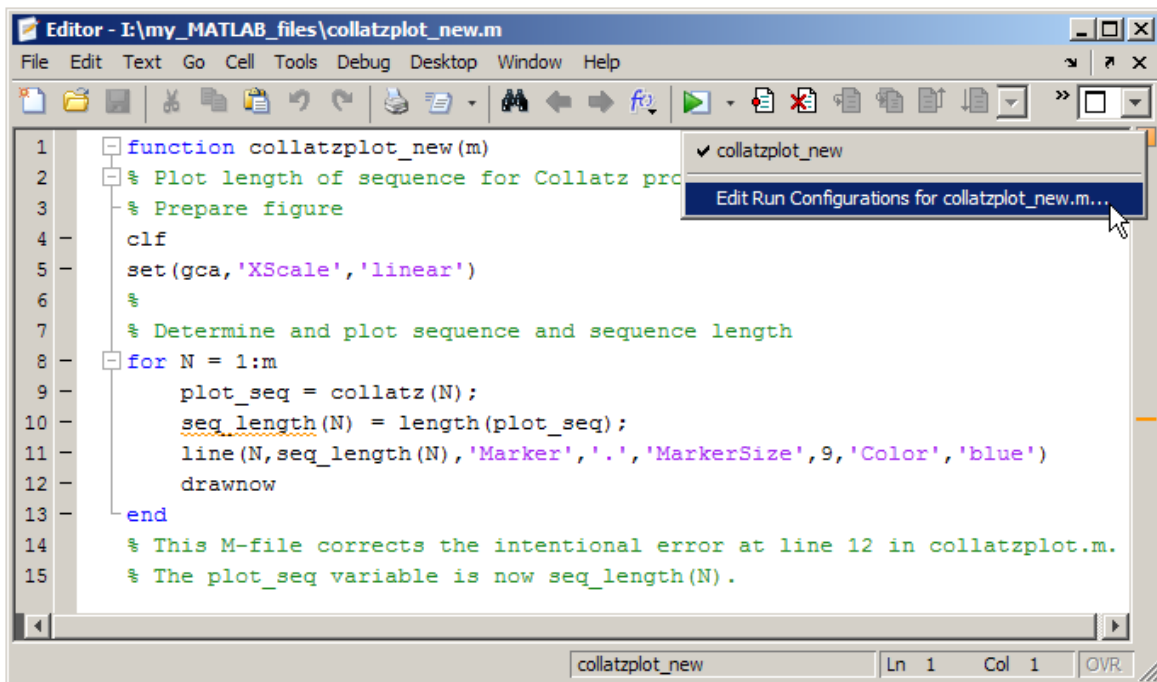
```
edit(fullfile(matlabroot, 'help', 'techdoc', ...  
            'matlab_env', 'examples', 'collatzplot_new.m'))
```

To work with `collatzplot_new.m` on your system, save the file to a folder for which you have write permission. In the example, the file is saved to `I:\my_matlab_files\my_mfiles\collatzplot_new.m`.

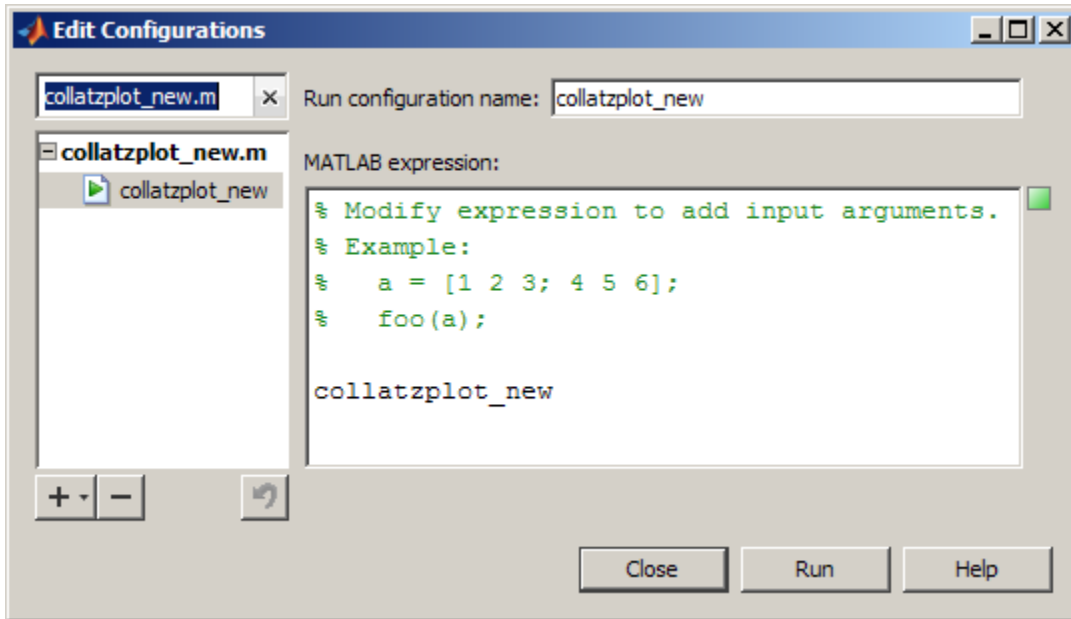
- 2 Click the down arrow on the Run button in the Editor toolbar,



and then select **Edit Run Configurations for *filename***, where *filename* in this example is `collatzplot_new.m`.

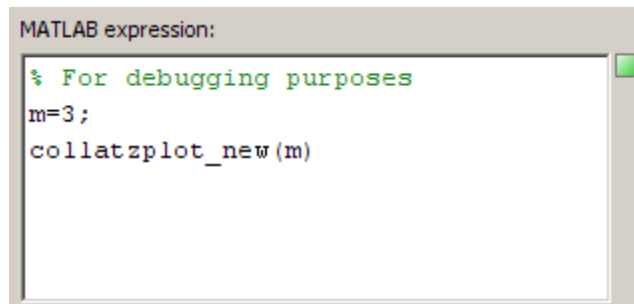


The Edit Configurations dialog box opens with a default run configuration template for `collatzplot_new.m`.



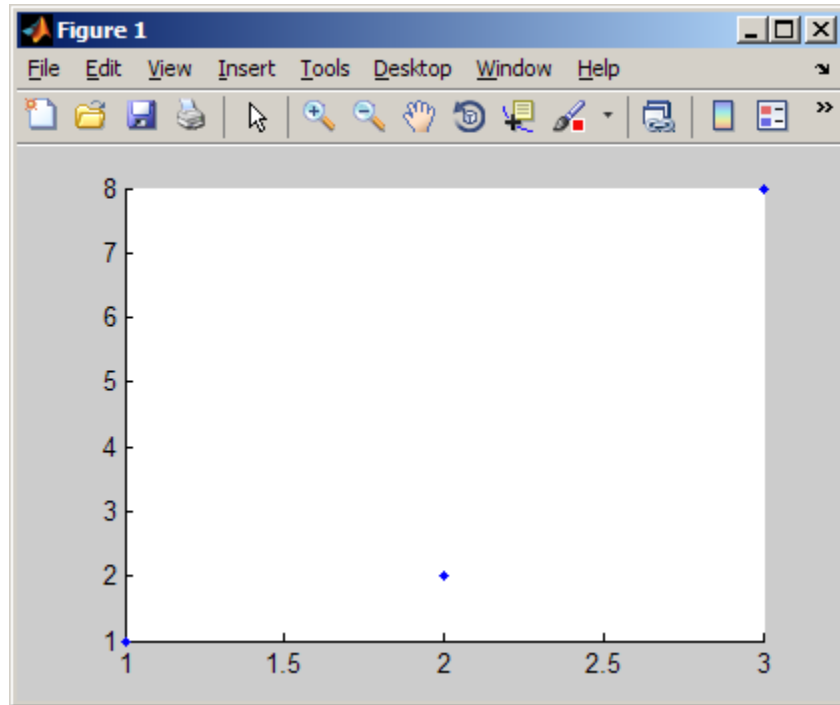
- 3 In the **MATLAB expression** area of the dialog box, enter MATLAB statements that you want to run. Delete the existing comments or replace them with comments relevant to your run configuration. To undo and redo, use the keyboard shortcuts for your platform, such as **Ctrl+Z** and **Ctrl+Y** for Microsoft Windows platforms.

In this example, set m equal to 3, which is a small value useful for debugging purposes. Complete the statement to run `collatzplot_new(m)`.



The **MATLAB expression** area provides syntax highlighting and shows M-Lint messages, like the Editor.

- 4 To ensure that your run configuration executes as expected, click **Run** to execute the statements in the **MATLAB expression** field. In this example, `collatzplot_new(3)` runs, and a Figure window displays the plot.




- 5 You can modify the statements in the **MATLAB expression** area of the dialog box and click **Run** to see the results of the changes. You can also modify the code file and save the changes while the Edit Configurations dialog box is open. Click **Run** to see the results of the changes you made to the file.
- 6 You can assign a name using the **Run configuration name** field in the Edit Configurations dialog box. By default, the run configuration name is the file name. If you expect to create multiple run configurations for a

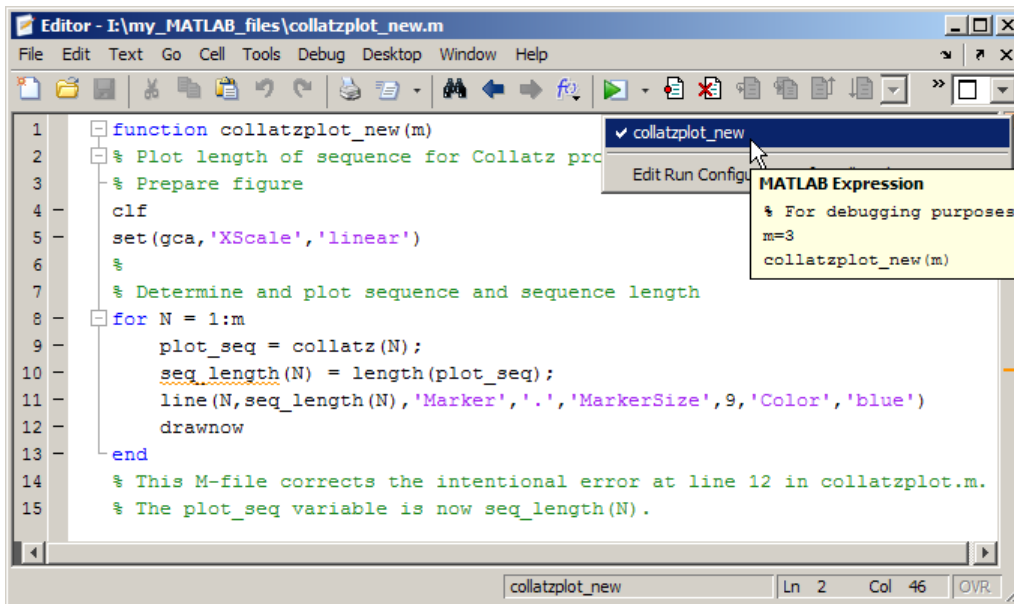
file, assign each a name that helps you identify the configuration. In this example, name the run configuration `collatzplot_new_test`.

MATLAB automatically saves the run configuration and its association with the file in the `run_configurations.m` file in your preferences folder.

For more information, see “About the `run_configurations.m` File” on page 9-96.

- 7 To close the Edit Configurations dialog box, click **Close**.
- 8 After creating a run configuration, you can view and use the configuration without opening the Edit Configurations dialog box.

In the Editor toolbar, click the down arrow on the Run button  and position the mouse pointer on a run configuration name. The MATLAB desktop displays a tooltip showing the **MATLAB Expression** associated with the run configuration, so you can see what will run.



- 9 To use the run configuration, select the run configuration name. MATLAB runs the expression you specified in the run configuration. For example,

select `collatzplot_new_test`, and MATLAB run `collatzplot_new(3)`, as specified in step 3. You can modify the file, save it, and then execute the run configuration from the toolbar to see the effects of the modifications.

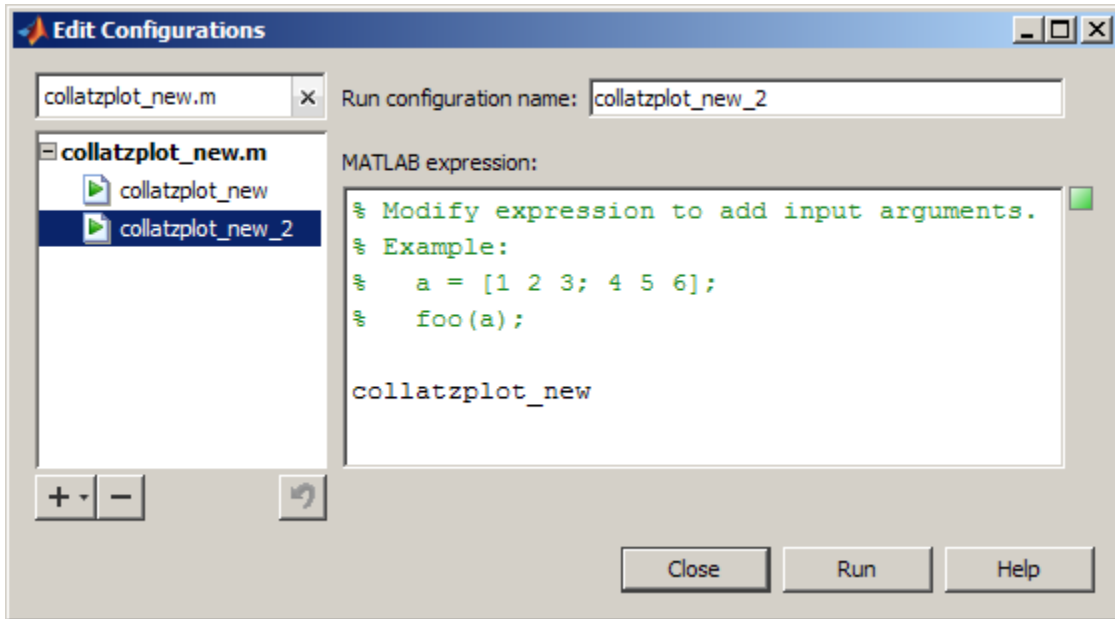
Create and Execute Multiple Run Configurations for a File

You can create multiple run configurations for a given file, allowing you to run with different values for input arguments, each for a different purpose. Create a named run configuration for each purpose, all associated with the same file. Then any time you open that file, choose and execute the run configuration you want. For example, for `collatzplot_new(m)` you might use three values for `m` and have three run configurations:

- Small value, for example, 3, for debugging and testing
 - Realistic value, for example, 200 or more, for a specific project
 - Random value to observe changes
- 1 Open the Edit Configurations dialog box, and then do the following:
 - a Select the file to which you want to add a run configuration, or select a configuration already associated with that file.
 - b Click the Add button **+** (under the list of configurations), and then click **Run Configuration**.

MATLAB creates a new default run configuration template, in this example, `collatzplot_new_2`.

The example shows `collatzplot_new_2` and its default expression, as well as one previously created run configuration associated with `collatzplot_new.m`, `collatzplot_new`.



- 2 In the Edit Configurations dialog box, modify, run, and name the new run configurations as you did for the initial run configuration, `collatzplot_new`, as described in “Create and Use a Run Configuration” on page 9-88.

For example, rename `collatzplot_new_2` to `collatzplot_new_largevalue`, and replace the default template expression with:

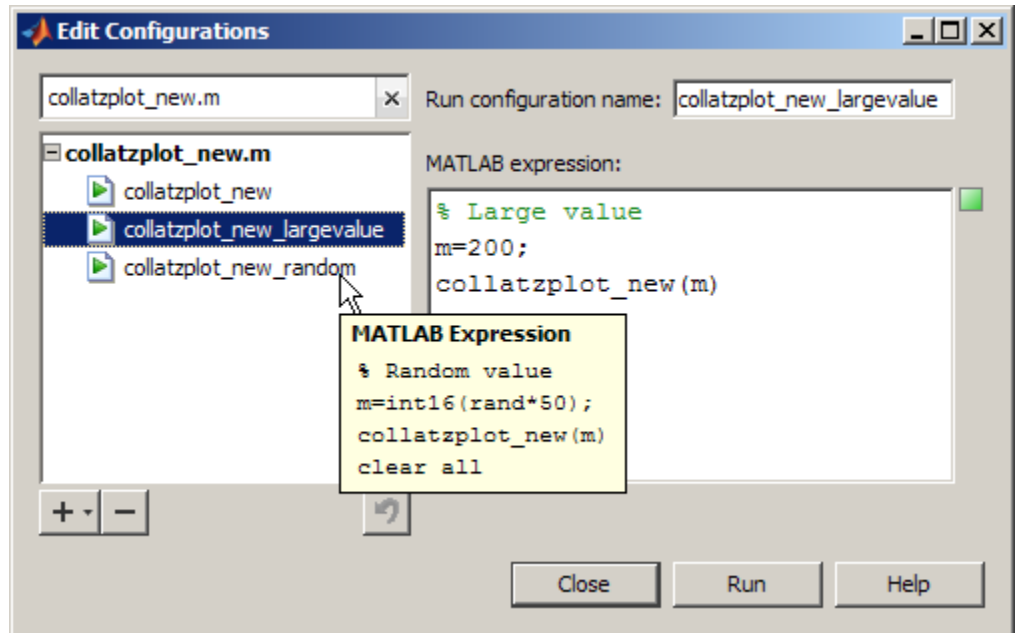
```
% Large value
m=200;
collatzplot_new(m)
```

To create another run configuration, click the down arrow next to the Add button **+**, again, and then click **Run Configuration**. Rename `collatzplot_new_2` to `collatzplot_new_random` and replace the default template expression with:

```
% Random value
m=int16(rand*50);
collatzplot_new(m)
```

```
clear all
```

- 3 Select a run configuration in the listing to see and modify its expression, or to rename the configuration. Click the expanders next to a file name (plus + and minus - signs on Windows platforms) to see or hide all the configurations associated with that file.
- 4 To get a quick view of the expression in a configuration, position the mouse pointer on the name of a configuration without selecting it. In this example, `collatzplot_new_largevalue` is selected and you can edit its expression or name. The pointer is positioned on `collatzplot_new_2` and you can see the statements in it.



- 5 To close the Edit Configurations dialog box, click **Close**. MATLAB saves the configurations and their associations with the file in `run_configurations.m` in your preferences folder.

For more information, see “About the `run_configurations.m` File” on page 9-96.

About the `run_configurations.m` File

When you create one or more run configurations using the Edit Configurations dialog box, the Editor creates or updates the `run_configurations.m` file in your preferences folder. (MATLAB returns the preferences folder when you run `prefdir`.) The `run_configurations.m` file is a text file that you can view and use to evaluate MATLAB code files.

Although you can port this file from the preferences folder on one system to another, there can only be one `run_configurations.m` file on a system. Therefore, only do this if you have not already created configurations on the second system. Also, because this file might contain references to file paths, ensure that the MATLAB file and paths it specifies exist on the second system.

MathWorks recommends that you do not update this file in the Editor or a text editor. Changes you make using tools other than the Edit Configurations dialog box might be overwritten.


Each time you change a run configuration using the Edit Configurations dialog box, MATLAB updates the `run_configurations.m` file as well as the `publish_configurations.m` file. See “About the `publish_configurations.m` File” on page 11-104 for more information about that file.

Find Configurations

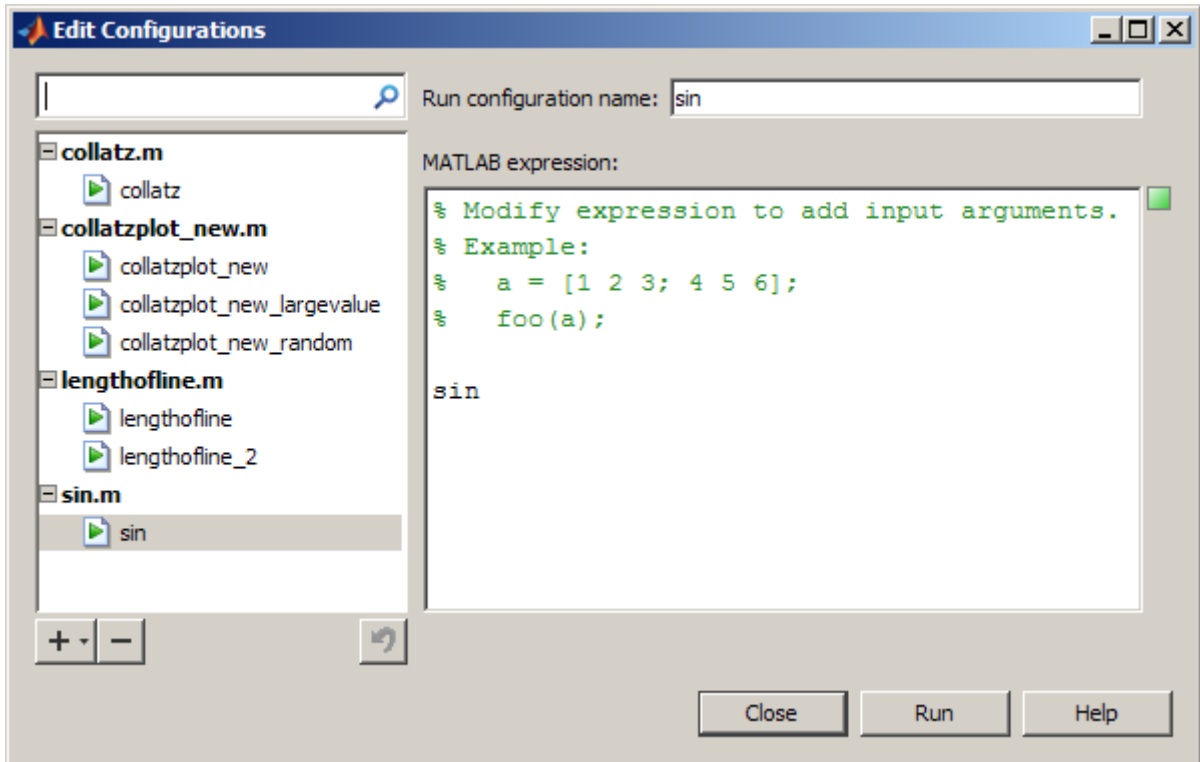
Follow these steps to find run or publish configurations. (For information on publish configurations, see “Specifying Output Preferences for Publishing” on page 11-64.)

- 1 Open any MATLAB code file in the Editor. For example, open the MATLAB function file `sin.m`.
- 2 Open the Edit Configurations dialog box. If none exists, MATLAB automatically creates a default configuration for `sin.m`.

In the left pane, MATLAB lists all configurations currently defined for `sin.m`.

- 3 Click the Clear search button within the filter field  to clear the filter field.

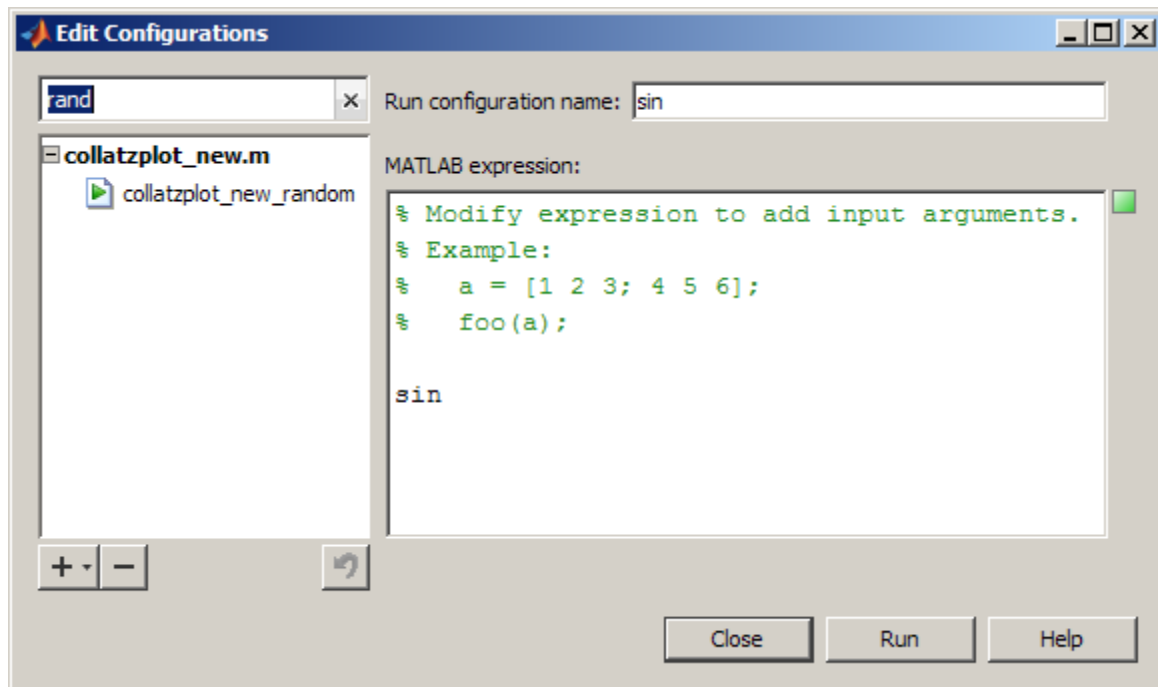
In the left pane, MATLAB lists all files with configurations.



- 4 Type a term in the filter field to find a file or configuration by name.

MATLAB displays only those files whose names contain the term, or whose associated configurations contain the term in their name. As you type, MATLAB filters out files and configurations that do not contain the term.

For example, type rand. In this example, only one file, collatzplot_new.m, has a configuration that contains the term rand.





- 5 If you cannot view the entire name of a configuration, drag the separator bar to the right of the list, making the left pane wider.
- 6 To see the expression in that configuration, select the configuration, or position the mouse pointer over the name.
- 7 As you type additional letters in the filter field, fewer files remain in the list of results. Use the **Backspace** key to modify the term. If there are no files or configurations containing the term, the list is empty.

Remove Configurations

If you no longer need a run or publish configuration because you do not use it or because you deleted the file with which it is associated, consider deleting the configuration. (For information on publish configurations, see “Specifying Output Preferences for Publishing” on page 11-64.)

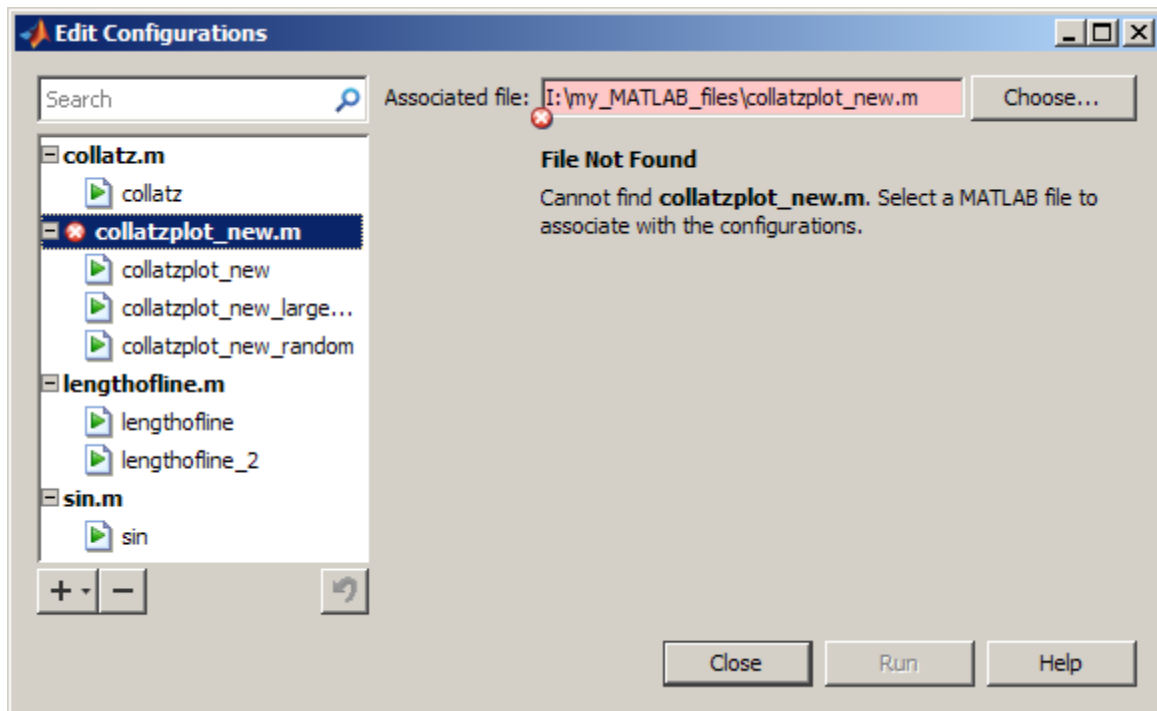
- 1 Open any MATLAB code file in the Editor.

- 2 Open the Edit Configurations dialog box.
- 3 Do one of the following in the pane on the left:
 - If you want to remove a single configuration, select that configuration.
 - If you want to remove all the run and publish configurations for the file, select the file.
- 4 Click the Remove button .
- 5 To undo the last deletion, click the Undo button . You cannot undo the last deletion after you close this dialog box.

Reassociate and Rename Configurations

Each run and publish configuration is associated with a specific file. If you move or rename a file that has configurations, redefine the association. If you delete a file, consider deleting the associated configurations, or associate them with a different file. You might also need to modify the statements in the configurations so they will run.

When MATLAB cannot associate a configuration with a file, the Edit Configurations dialog box displays the file name in red, displays a **File Not Found** message, and enables you to find the file to which you want to associate the configuration. In this example, MATLAB cannot find the file `collatzplot_new.m`, which has three configurations associated with it. For this example, `collatzplot_new.m` had been renamed to `collatzplot_fixed.m`, so the configurations associated with `collatzplot_new.m` need to be reassociated with `collatzplot_fixed.m`.

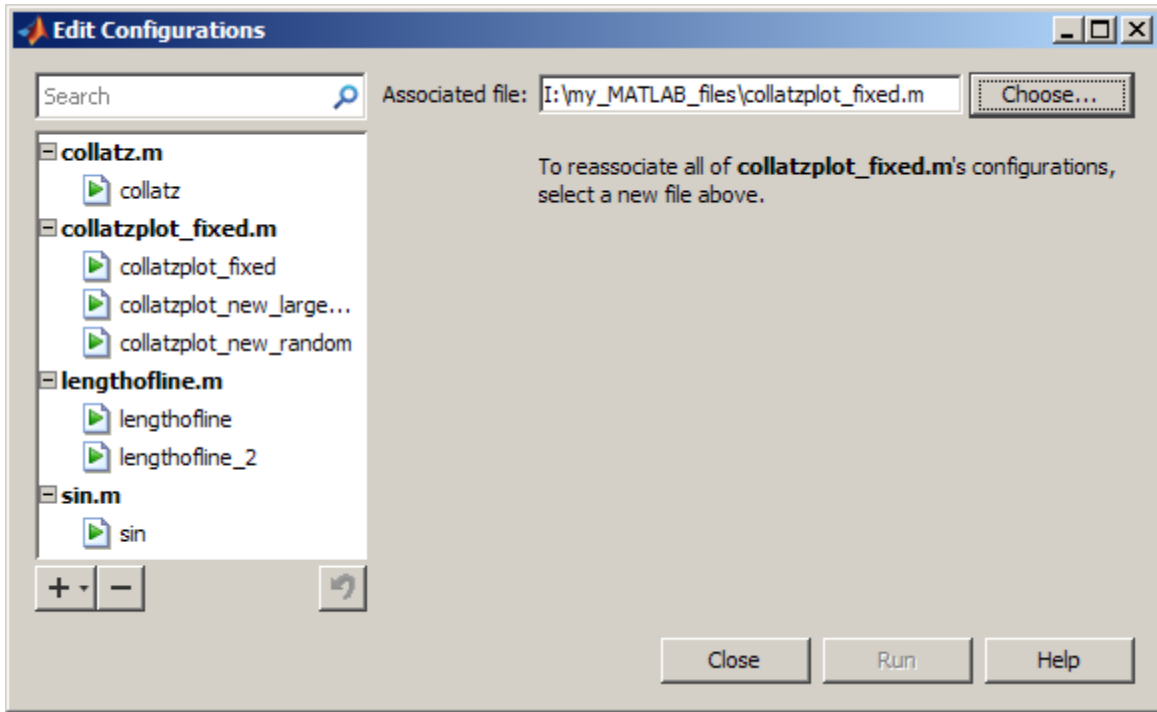


To reassociate a configuration:

- 1 In the left pane, select the file. The **Associated file** field displays the full path to the file that was associated with the configurations. Click **Choose**.
- 2 In the resulting Open dialog box, navigate to and select the file with which you now want to reassociate the configurations. Click **Open**.

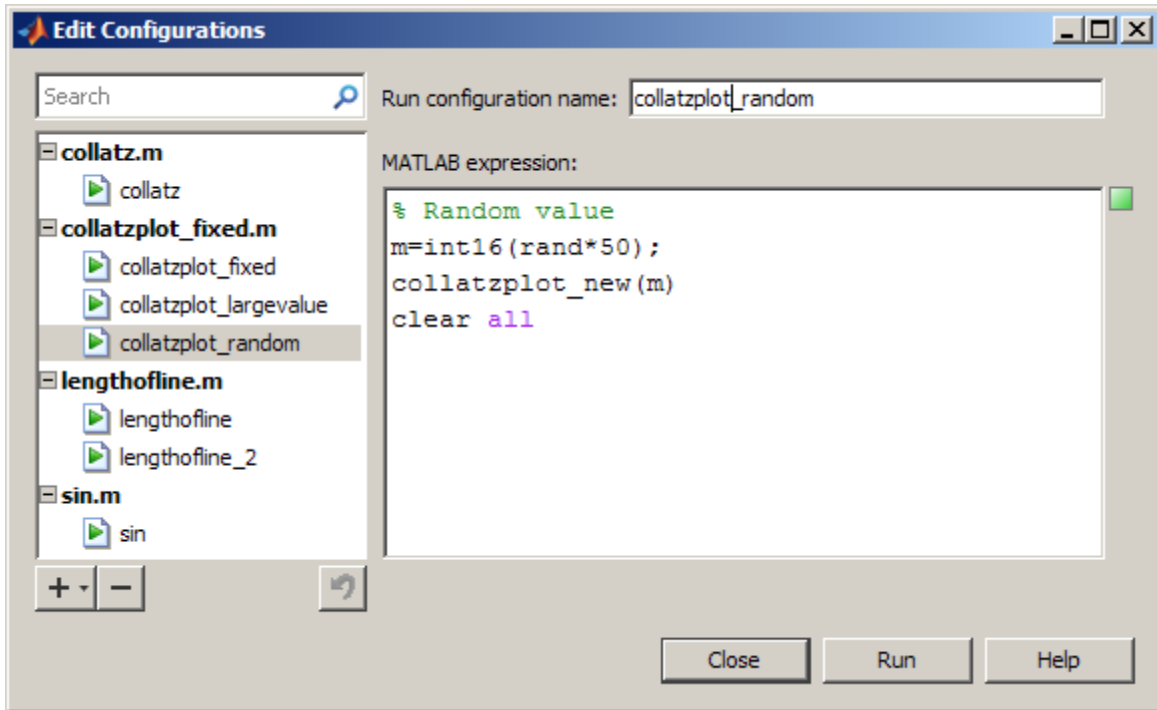
In this example, you want to reassociate the configurations with `collatzplot_fixed.m`; select `collatzplot_fixed.m`, and then click **Open**.

In the Edit Configurations dialog box, the **Associated file** value reflects the change you made and the **File Not Found** message no longer appears.

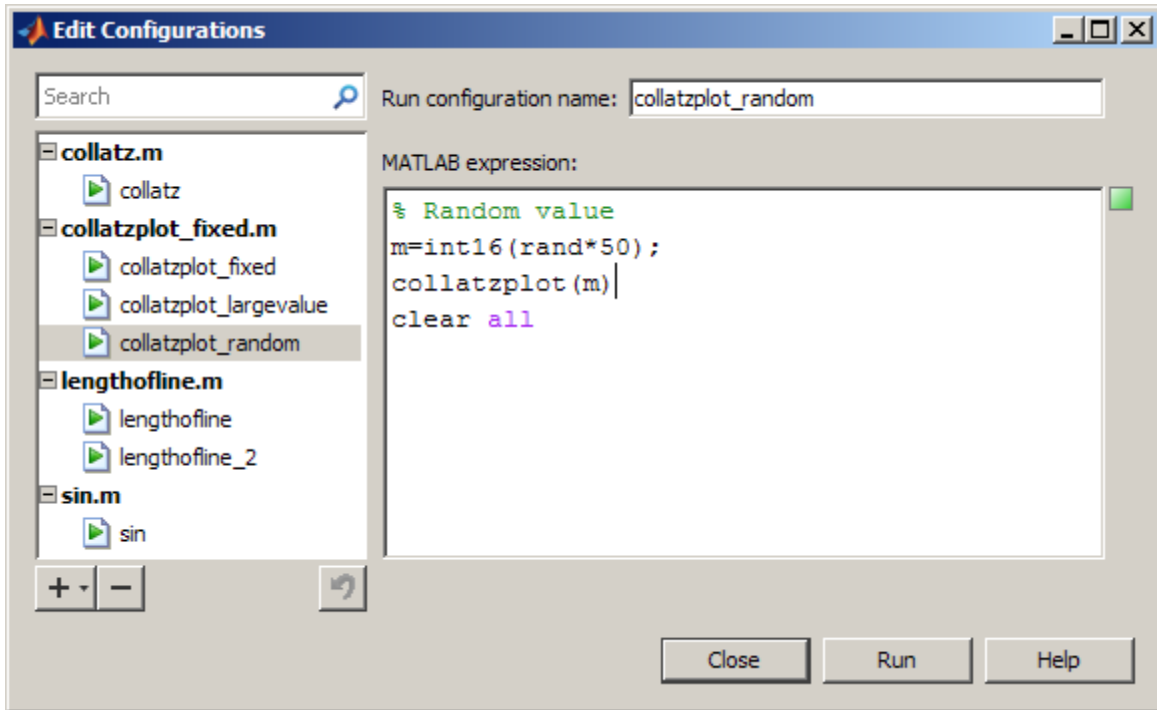


- 3** Consider renaming the configurations to be consistent with the new file name, or at least to not reflect the former file name. To do so, select a configuration from the list in the left pane. In the right pane, edit the value for the configuration name. Depending on the type of configuration that you are renaming, the field label is either **Run configuration name** or **Publish configuration name**. Repeat this step for all run and publish configurations associated with the file.

In this example, remove `collatzplot_new` from the start of each run configuration name.



- 4 For a file name change, you might need to modify the configuration statements to run correctly. For this example, modify the `collatzplot_new(m)` statement in each configuration to use `collatzplot(m)`.



Other Ways to Run Files from the Editor

- For additional information about running files while debugging, see “Running a File with Breakpoints” on page 9-148 .
- While debugging, you can execute sections of a file even though there are changes. See “Running Sections in MATLAB Files That Have Unsaved Changes” on page 9-165.
- You can execute files one section at a time and quickly modify values incrementally using the toolbar. For more information, see “Evaluating Subsections of Files Using Code Cells” on page 9-175.

Finding Errors, Debugging, and Correcting MATLAB Files

This section introduces general techniques for finding errors and using the automatic code analysis to detect possible areas for improvement in MATLAB code. It then illustrates the MATLAB debugger features in the Editor, as well as equivalent Command Window debugging functions, using a simple example.

There are two kinds of errors:

- Syntax errors — For example, misspelling a function name or omitting a parenthesis.
- Run-time errors — These errors are usually algorithmic in nature. For example, you might modify the wrong variable or code a calculation incorrectly. Run-time errors are usually apparent when a file produces unexpected results. Run-time errors are difficult to track down because the function’s local workspace is lost when the error forces a return to the MATLAB base workspace. The process of isolating and fixing these run-time problems is referred to as *debugging*.

In addition to finding and fixing problems with your code, 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 MATLAB code.

Technique or Tool	Description	For More Information
Syntax highlighting and delimiter matching	Syntax highlighting helps you identify unterminated strings in a file before you run the file. Delimiter matching helps you correctly match pairs of parentheses, brackets, braces, and keywords.	“Syntax Highlighting” on page 9-53 “Matching Delimiters (Parentheses)” on page 3-24
Nonlocal variable highlighting	Nonlocal variable highlighting helps you identify sources of coding errors that are due to errors in variable reuse across nested functions.	“Determining Scope and Usage of Functions and Variables” on page 9-135

Technique or Tool	Description	For More Information
Error messages	<p>When you run a file with a syntax error, MATLAB software will most likely detect it and display an error message in the Command Window describing the error and showing its line number in the file. Click the underlined portion of the error message, or position the cursor within the message and then press Ctrl+Enter. (This is the default keyboard shortcut for the Display Hyperlinked Error action on Windows.) The offending file opens in the Editor, scrolled to the line containing the error.</p>	“Displaying Keyboard Shortcuts” on page 2-75
Code analysis and M-Lint messages	<p>Use code analysis and M-Lint messages to help you verify the integrity of your MATLAB code and learn about potential improvements. Access messages automatically while you work in a file in the Editor, or run a Code Analyzer Report for an existing file.</p> <p>To evaluate the McCabe complexity (also known as the cyclomatic complexity) of a file, use the <code>mlint</code> function with the <code>-cyc</code> option.</p>	“Preventing and Identifying Coding Problems” on page 9-107 and the reference page for the <code>mlint</code> function
Editor, graphical debugger, and MATLAB debugging functions	<p>The MATLAB Editor, graphical debugger, and MATLAB debugging functions are useful for correcting run-time problems. They enable you to access function workspaces and examine or change the values they contain. You can set and clear <i>breakpoints</i>, indicators that temporarily halt execution in a file. While stopped at a breakpoint, you can change workspace contexts, view the function call stack, and execute the lines in a file one by one.</p>	“Debugging Process and Features” on page 9-141

Technique or Tool	Description	For More Information
Other debugging techniques	<ul style="list-style-type: none"> • Add keyboard statements to the MATLAB code file—keyboard statements stop file execution at the point where they appear and enable you to examine and change the function’s local workspace. A special K>> prompt indicates this mode. Resume function execution by typing return and pressing the Enter key. • Remove selected semicolons from the statements in your MATLAB code file—semicolons disable the display of output in the file. By removing the semicolons, you instruct MATLAB to display these results on your screen as the file executes. • List dependent functions—use the depfun function to see the dependent functions. 	Reference pages for keyboard and depfun function
Cells	In the Editor, isolate sections of a MATLAB code file, called cells, so you can easily modify and run a single section.	“Evaluating Subsections of Files Using Code Cells” on page 9-175
Profiler	Use the Profiler to help you improve performance and detect problems in your MATLAB code files. Access the Profiler from the Editor by selecting Tools > Open Profiler .	“Profiling for Improving Performance” on page 10-27
Reports	Reports help you polish and package MATLAB code files before providing them to others to use. Access all of the reports from the Current Folder browser.	“Using MATLAB Reports” on page 10-2

Preventing and Identifying Coding Problems

In this section...

“Ways to Prevent and Check for Coding Problems” on page 9-107

“Code Analysis Options” on page 9-107

“Determining Scope and Usage of Functions and Variables” on page 9-135

Ways to Prevent and Check for Coding Problems

MATLAB provides features to help you avoid potential problems as you write your code and correct problems you introduce. In particular, you can:

- Highlight or provide a report on coding errors, inefficiencies, and potential problems

MATLAB can check your code for problems and recommend modifications to maximize performance and maintainability through messages, sometimes referred to as M-Lint messages. For details, see “Code Analysis Options” on page 9-107

- Highlight function and variable usage in your file


MATLAB can highlight function and variable usage throughout your file to help you track their scope and usage. This can help you avoid or fix coding problems that are the result of variable scoping mistakes. For details, see “Determining Scope and Usage of Functions and Variables” on page 9-135.

Code Analysis Options

You can check for coding problems three different ways, all of which report the same messages:

- Continuously check code in the Editor while you work.

View M-Lint messages and modify your file based on the messages. The messages update automatically and continuously so you can see if your changes addressed the issues noted in the messages. Some messages offer extended information, automatic code correction, or both. For details about using the continuous checking and correction interface in the Editor, see “Automatically Analyzing Code in the Editor” on page 9-108.

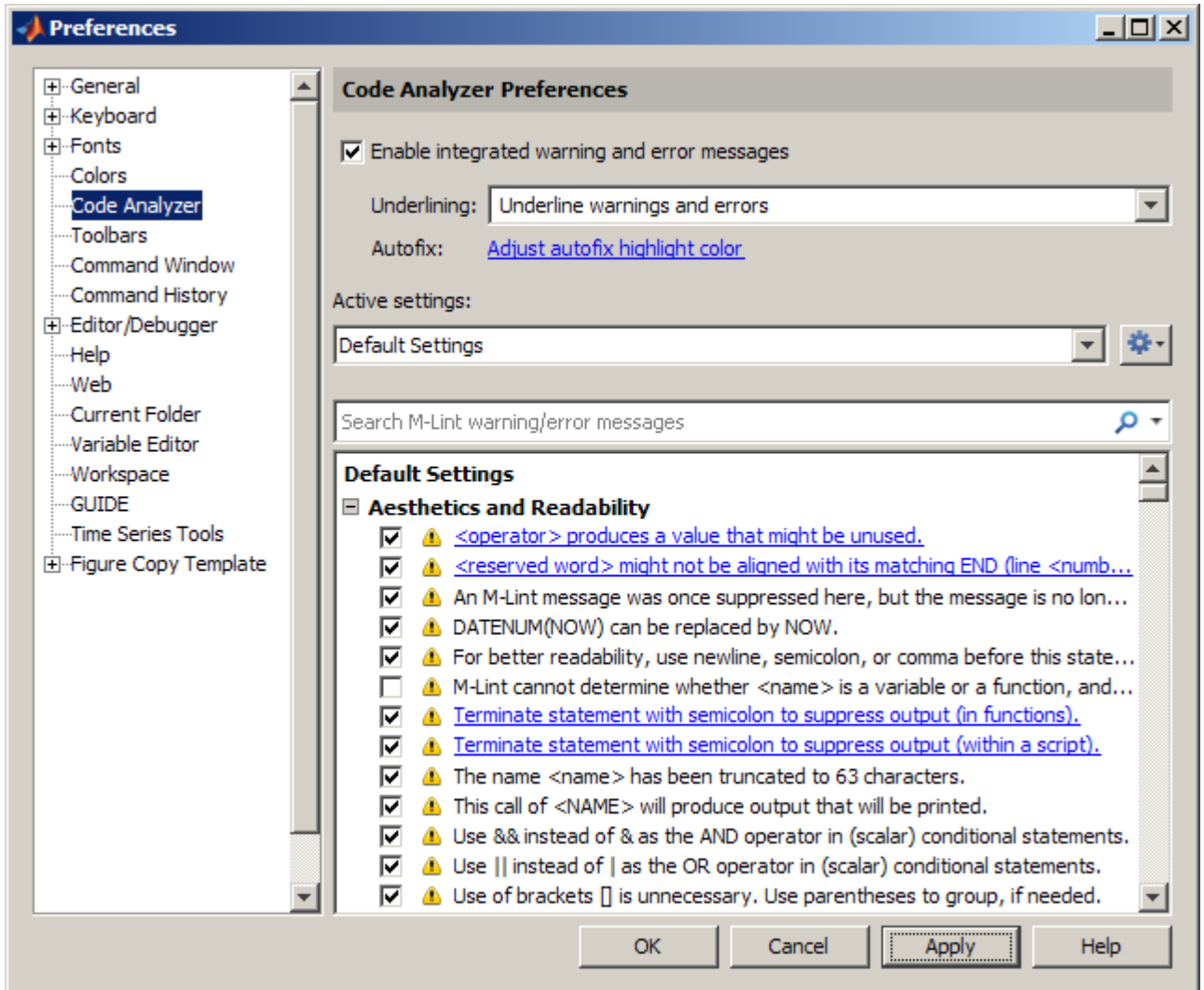
- Run a report for an existing MATLAB code file:
 - 1** From a file in the Editor, select **Tools > Code Analyzer > Show Code Analyzer Report**.
 - 2** Modify your file based on the M-Lint messages in the report.
 - 3** Save the file.
 - 4** Rerun the report to see if your changes addressed the issues noted in the messages.
- Run a report for all files in a folder:
 - 1** In the Current Folder browser, click the Actions button .
 - 2** Select **Reports > Code Analyzer Report**.
 - 3** Modify your files based on the messages in the report.

For details, see “Using the Code Analyzer Report” on page 10-22.
 - 4** Save the file.
 - 5** Rerun the report to see if your changes addressed the issues noted in the messages.

Automatically Analyzing Code in the Editor

To use continuous code checking in a MATLAB code file in the Editor:

- 1** Select **File > Preferences > Code Analyzer**, and then select the **Enable integrated warning and error messages** check box.
- 2** Set the **Underlining** option to **Underline warnings and errors**, and then click **OK**.



3 Open a MATLAB code file in the Editor. This example uses the sample file `lengthofline.m` that ships with the MATLAB software:

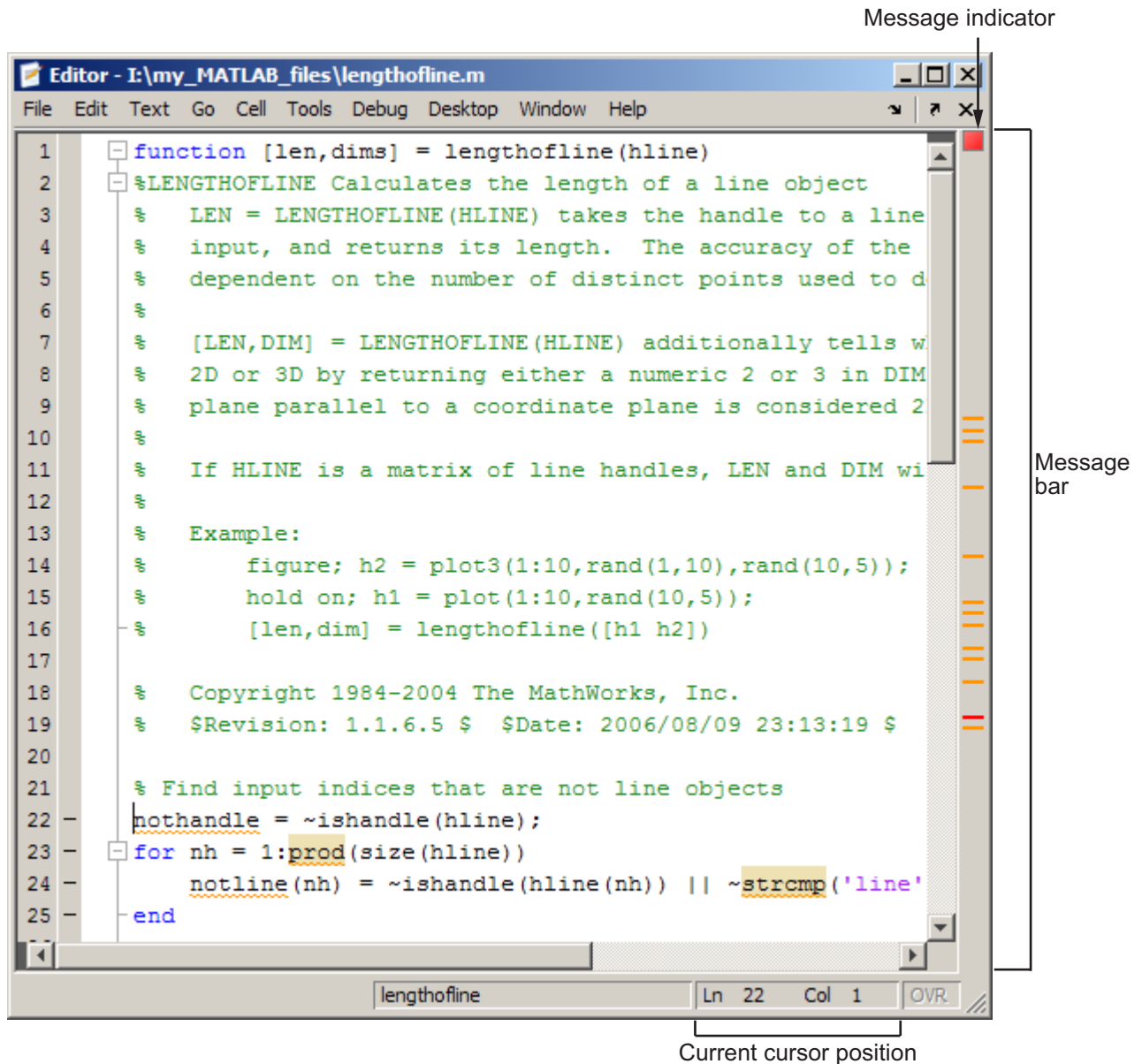
a Open the example file:

```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...
```

```
'examples', 'lengthofline.m'))
```

- b** Save the example file to a folder to which you have write access. For the example, `lengthofline.m` is saved to `I:\my_MATLAB_files`.
- 4** Examine the message indicator at the top right edge of the window to see the M-Lint messages reported for the file:
 - **Red** indicates syntax errors were detected. Another way to detect some of these errors is using syntax highlighting to identify unterminated strings, and delimiter matching to identify unmatched keywords, parentheses, braces, and brackets.
 - **Orange** indicates warnings or opportunities for improvement, but no errors, were detected.
 - **Green** indicates no errors, warnings, or opportunities for improvement were detected.

In this example, the indicator is red, meaning that there is at least one error in the file.



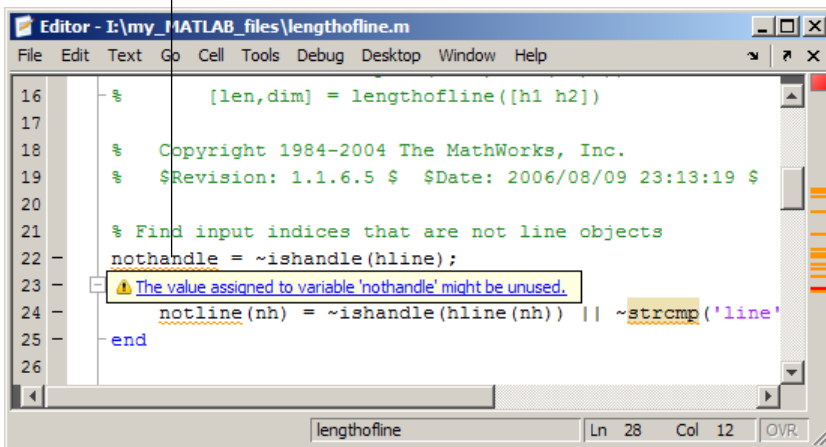
- 5 Click the message indicator to go to the next code fragment containing a message. The next code fragment is relative to the current cursor position, viewable in the status bar.

In the `lengthofline` example, the first message is at line 22. The cursor moves to the beginning of line 22.

The code fragment for which there is a message is underlined in either red for errors or orange for warnings and improvement opportunities.

- 6 View the message by moving the mouse pointer within the underlined fragment. The message appears with a yellow highlighted background.

Position cursor within orange underlined code fragment to display the related message.

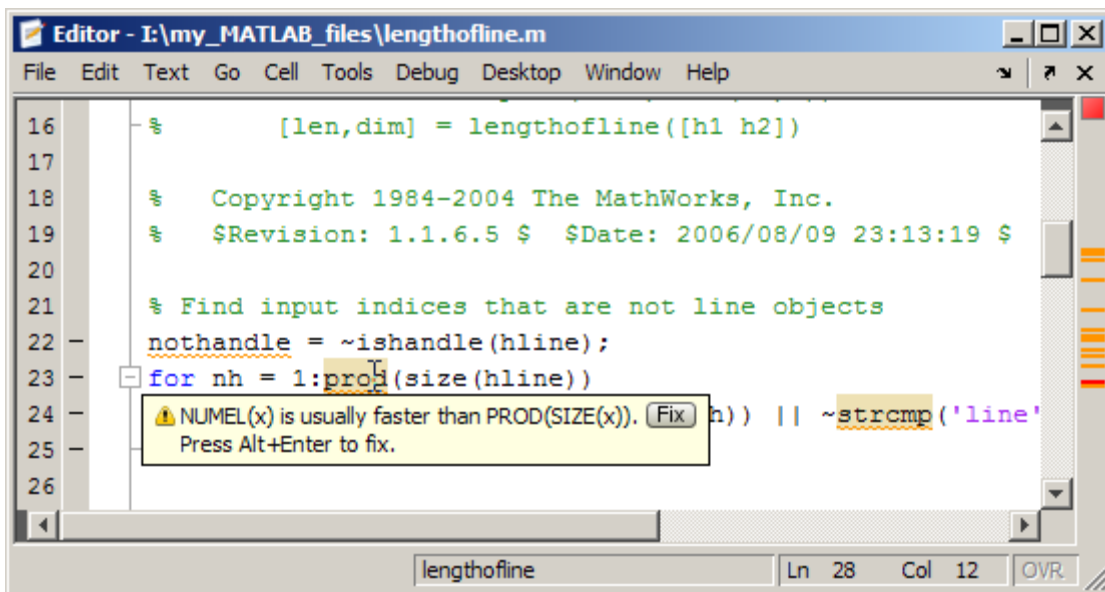


This message means that in line 22, `nothandle` is assigned a value, but is probably not used anywhere after that in the file. The line might be extraneous and you could delete it. But it might be that you actually intended to use the variable, as shown in step 7. Notice that the message is a link to additional information. Message links are discussed in steps 9 and 10.

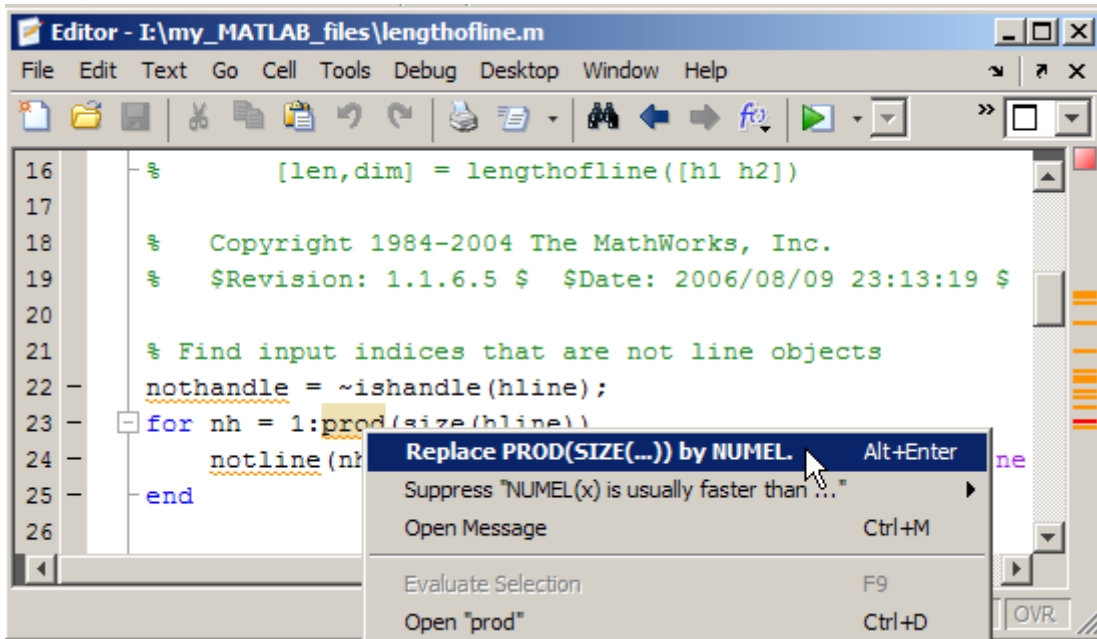
- 7 Modify your code as needed. The message indicator and underlining automatically update to reflect the changes you make, even if you do not save the file.

In this example, the intention was to use `nothandle` as a performance improvement by determining the value before the loop. Changing `~ishandle(hline(nh))` in line 24 to `nothandle(nh)` means that there is no longer a message associated with line 22.

- 8 In `lengthofline`, line 23, `prod` is underlined because there is a warning message, and it is highlighted because an automatic fix is available. When you view the message, it provides a button to apply the automatic fix. It also indicates the keyboard controls (**Alt+Enter**) to apply the fix.



To view what the automated fix is, right-click the highlighted code (for a single-button mouse, press **Ctrl+click**). The first item in the context menu indicates the automatic fix that MATLAB can perform. Select it and MATLAB automatically corrects the code. In this example, it replaces `prod(size(hline))` with `numel(hline)`.

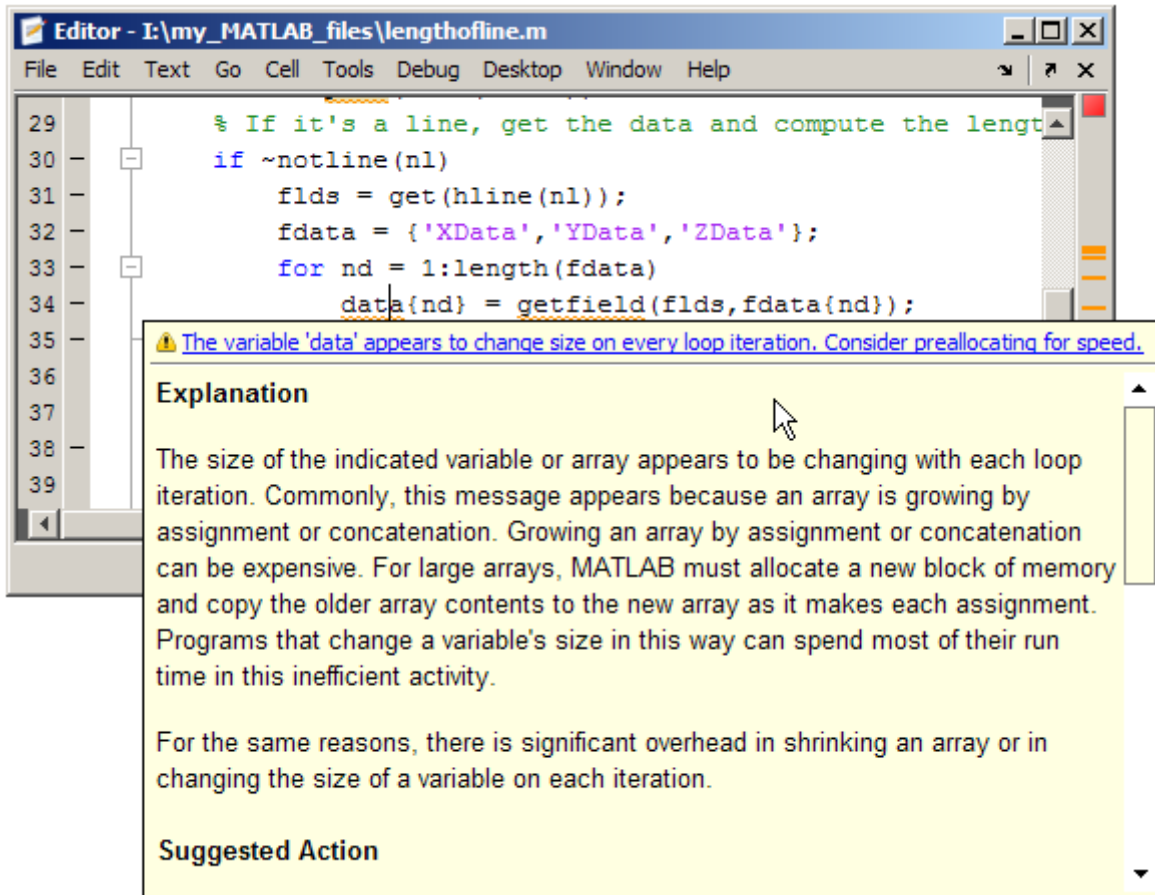


After you apply the fix, MATLAB removes the underline from `prod` in line 23.

- 9 Move the mouse pointer over the underlined code fragment `data` on line 34.

When you view this message, notice that it appears as a link. The link indicates that there is more information about the message. Not all messages have additional information.

- 10 Click the link in the message. The window expands to display an explanation and user action.



- 11 Click the message indicator to go to the next message, or use the indicator bar.

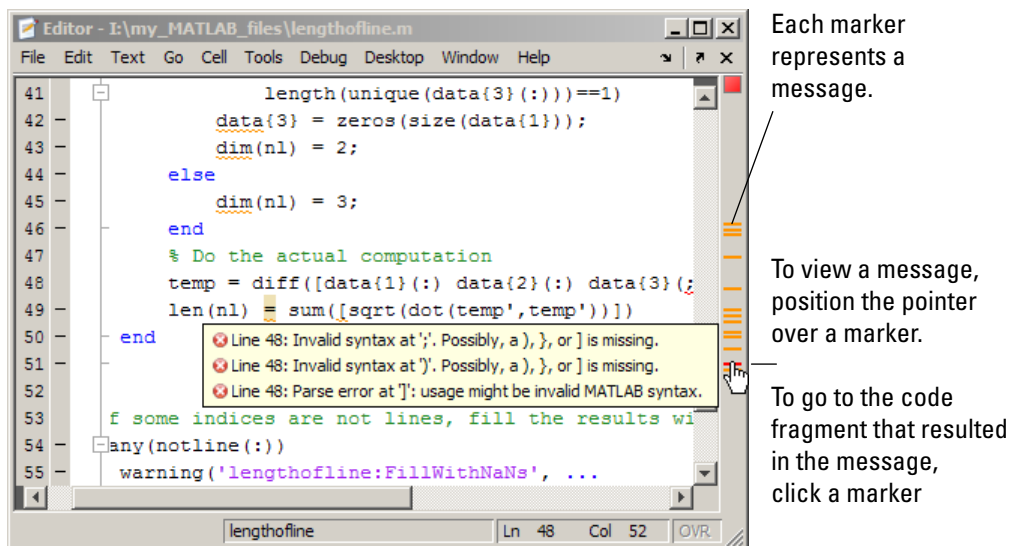
Each marker in the bar represents one or more lines that have associated messages.

- Position the mouse pointer at a marker in the indicator bar to view the message. For example, to see an error in `lengthofline`, position the pointer at a red (error) marker in the indicator bar. There is only one error in the file and with the pointer positioned over it, the associated messages appears. (There can be multiple messages per line.) Click the

marker to go to the first code fragment in the line that resulted in a message. For the example, click the red marker, which takes you to the first suspect code fragment in line 48.

```
temp = diff([data{1}(:) data{2}(:) data{3}(;)]);
```

Multiple messages can represent a single problem or multiple problems. Addressing one might address all of them, or after addressing one, the other messages might change or what you need to do might become clearer.



- b** Modify the code to address the problem noted in the message—the message indicators update automatically.

In the example, the message suggests a delimiter imbalance. You can check that by following these steps:

- i** Select **File > Preferences > Keyboard > Delimiter Matching**, and then select **Match on arrow key**, if it is not already selected.
- ii** Move the arrow key over each of the delimiters to see if MATLAB indicates a mismatch.

In the example, it might appear that there are no mismatched delimiters. However, code analysis detects the semicolon in parentheses: `data{3} (;)`, and interprets it as the end of a statement. The message reports that the two statements on line 48 each have a delimiter imbalance.

iii In line 48, change `data{3} (;)` to `data{3} (:)`.

Now, the underline no longer appears in line 48. The single change addresses the issues in both of the messages for line 48.

Because the change removed the only error in the file, the message indicator at the top of the bar changes from red to orange, indicating that only warnings and potential improvements remain.

If there are multiple messages associated with a line, there might be multiple underlined code fragments that are adjacent, as in the previous step. It can be difficult to display the message of interest. In such a case, it is easier to view the messages through the marker on the indicator bar than moving the arrow over each delimiter.

After modifying the code to address all the messages, or disabling designated messages, the message indicator becomes green. The example file with all messages addressed has been saved as `lengthofline2.m`. Open the example file with the command:

```
open(fullfile(matlabroot, 'help', 'techdoc', ...  
            'matlab_env', 'examples', 'lengthofline2.m'))
```

Suppressing Message Indicators and Messages

Depending on the stage at which you are in completing a MATLAB file, you might want to restrict the code underlining. You can do this by using the Code Analyzer preference referred to in step 1, in “Preventing and Identifying Coding Problems” on page 9-107. For example, when first coding, you might prefer to underline only errors because warnings would be distracting. For details, click the **Help** button in the Preferences dialog box.

Code analysis does not provide perfect information about every situation and in some cases, you might not want to change the code based on a message. If

you do not want to change the code, and you do not want to see the indicator and M-Lint message for that line, suppress them. For the `lengthofline` example, in line 49, the first message is `Terminate statement with semicolon to suppress output (in functions)`. Adding a semicolon to the end of a statement suppresses output and is a common practice. Code analysis alerts you to lines that produce output, but lack the terminating semicolon. If you want to view output from line 49, do not add the semicolon as the message suggests.

There are a few different ways to suppress (turn off) the indicators for warning and error messages:

- “Suppress an Instance of a Message in the Current File” on page 9-118
- “Suppress All Instances of a Message in the Current File” on page 9-119
- “Suppress All Instances of a Message in All Files” on page 9-120
- “Specify Default Message Settings” on page 9-120
- “Specify Nondefault Message Settings” on page 9-121
- “Understanding Code Containing Suppressed Messages” on page 9-122

You cannot suppress error messages such as syntax errors. Therefore, instructions on suppressing messages do not apply to those types of messages.

Suppress an Instance of a Message in the Current File. You can suppress a specific instance of an M-Lint message in the current file. For example, using the code presented in “Preventing and Identifying Coding Problems” on page 9-107, follow these steps:

- 1** In line 49, right-click at the first underline (for a single-button mouse, press **Ctrl+click**).
- 2** From the context menu, select **Suppress 'Terminate statement with semicolon...' > On This Line**.

The comment `%#ok<NOPRT>` appears at the end of the line, which instructs MATLAB not to check for a terminating semicolon at that line. The underline and mark in the indicator bar for that message disappear.

- 3 If there are two messages on a line that you do not want to display, right-click separately at each underline and select the appropriate entry from the context menu.

The `%#ok` syntax expands. For the example, in the code presented in “Preventing and Identifying Coding Problems” on page 9-107, ignoring both messages for line 49 adds `%#ok<NBRAK,NOPRT>`.

Even if Code Analyzer preferences are set to enable this message, the specific instance of the message suppressed in this way does not appear because the `%#ok` takes precedence over the preference setting. If you later decide you want to check for a terminating semicolon at that line, delete the `%#ok<NOPRT>` string from the line.

For more information about `%#ok`, see the `mlint` function reference page.

Suppress All Instances of a Message in the Current File. You can suppress all instances of a specific M-Lint message in the current file. For example, using the code presented in “Preventing and Identifying Coding Problems” on page 9-107, follow these steps:

- 1 In line 49, right-click at the first underline (for a single-button mouse, press **Ctrl**+click).
- 2 From the context menu, select **Suppress 'Terminate statement with semicolon...' > In This File**.

The comment `%#ok<*NOPRT>` appears at the end of the line, which instructs MATLAB to not check for a terminating semicolon throughout the file. All underlines, as well as marks in the message indicator bar that correspond to this message disappear.

If there are two messages on a line that you do not want to display anywhere in the current file, right-click separately at each underline, and then select the appropriate entry from the context menu. The `%#ok` syntax expands. For the example, in the code presented in “Preventing and Identifying Coding Problems” on page 9-107, ignoring both messages for line 49 adds `%#ok<*NBRAK,*NOPRT>`.

Even if Code Analyzer preferences are set to enable this message, the message does not appear because the `%#ok` takes precedence over the preference setting. If you later decide you want to check for a terminating semicolon in the file, delete the `%#ok<*NOPRT>` string from the line.

For more information about `%#ok`, see the `mLint` function reference page.

Suppress All Instances of a Message in All Files. You can disable all instances of an M-Lint message in all files. For example, using the code presented in “Preventing and Identifying Coding Problems” on page 9-107, follow these steps:

- 1** In line 49, right-click at the first underline (for a single-button mouse, press **Ctrl+click**).
- 2** From the context menu, select **Suppress 'Terminate statement with semicolon...' > In All Files**.

This modifies the Code Analyzer preference setting. For more information about Code Analyzer preferences, including how to restore MATLAB default settings, select **File > Preferences > Code Analyzer**, and then click **Help**.

Specify Default Message Settings. You can specify that you want certain M-Lint messages to be disabled by default when you open any MATLAB file. Typically, you do this if you find that you do not want certain messages or categories of messages enabled for all or most of your MATLAB files.

Follow these steps:


- 1** Select **File > Preferences > Code Analyzer**.

The Preferences dialog box opens and displays the Code Analyzer preferences pane.

- 2** Disable specific M-Lint messages, or categories of messages.

The **Active Settings** field now contains the value `Default Settings (modified)`.

- 3** Click **Apply**.

Now, each file you open uses the modified default settings. If you want to restore the factory-installed default settings, decide if you want to save the current settings to a file, as described in “Specify Nondefault Message Settings” on page 9-121. Then, click the Actions button , and select **Restore Defaults**.


Specify Nondefault Message Settings. You can specify that you want certain M-Lint messages enabled or disabled, and then save those settings to a file. When you want to use a settings file with a particular file, you select it from the Code Analyzer preferences pane. That setting file remains in effect until you select another settings file. Typically, you change the settings file when you have a subset of files for which you want to use a particular settings file.

Follow these steps:

1 Select File > Preferences > Code Analyzer

The Preferences dialog box opens and displays the Code Analyzer preferences pane.

2 Enable or disable specific messages, or categories of messages.

3 Click the Actions button , select Save as, and then save the settings to a txt file.

4 Click OK.

You can reuse these settings for any MATLAB file, or provide the settings file to another user.

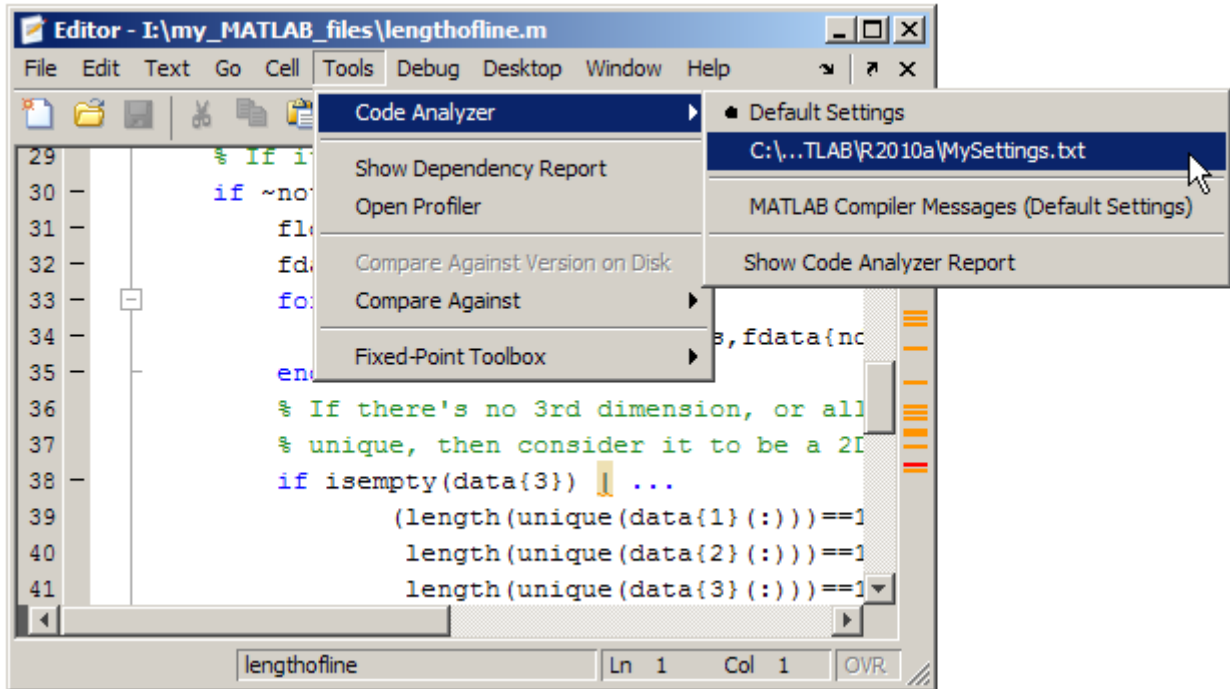
To use the saved settings:

1 In the Editor, select Tools > Code Analyzer.

The currently selected setting choice displays, preceded by a bullet point. In the image shown here, the Default Settings are currently selected.

2 Choose from any of the settings files, such as the My_Settings example, as shown here.

The settings you choose are in effect for all MATLAB files until you select another set of Code Analyzer settings.



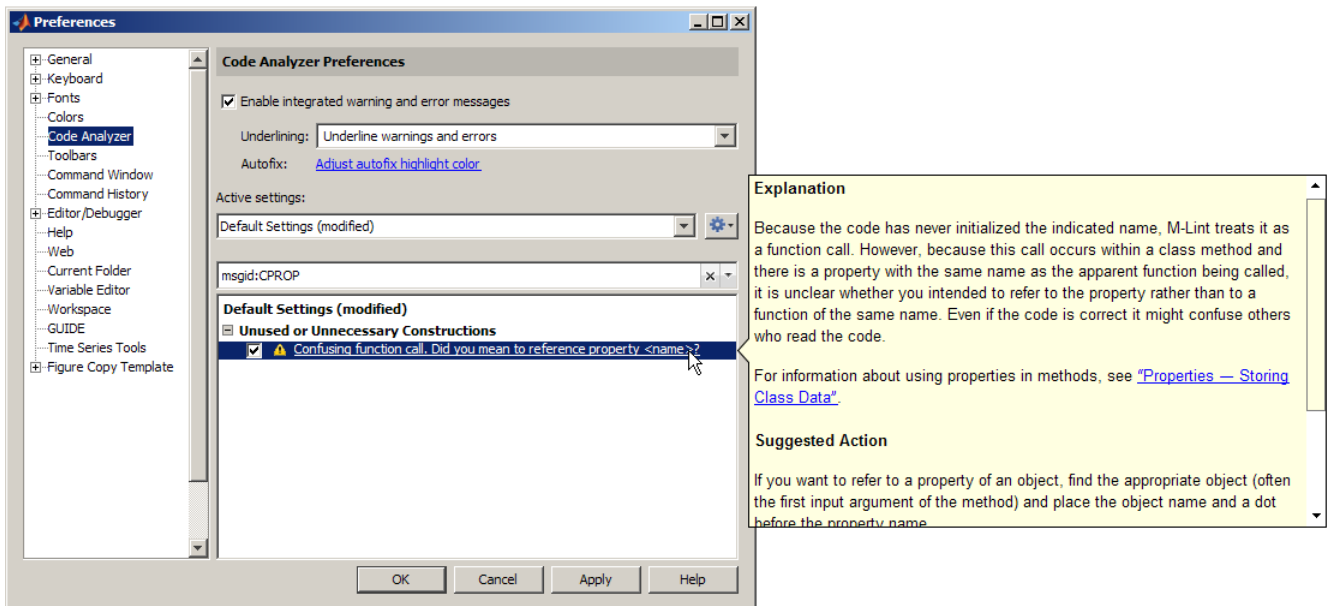
Understanding Code Containing Suppressed Messages. If you receive code that contains suppressed M-Lint messages, you might want to review those messages without the need to unsuppress them first. A message might be in a suppressed state for any of the following reasons:


- One or more `%ok<message-ID>` pragmas are on a line of code that elicits a message specified by `<message-ID>`.
- One or more `%ok<*message-ID>` pragmas are in a file that elicits a message specified by `<message-ID>`.
- It is cleared in the Code Analyzer preferences pane.
- It is disabled by default.


To determine the reasons why some messages are suppressed:

- 1 Search the file for the `%#ok` pragma and create a list of all the message IDs associated with that pragma.
- 2 Open the Code Analyzer preferences dialog box by selecting **File > Preferences > Code Analyzer**.
- 3 In the filter field, type `msgid:` followed by one of the message IDs, if any, you found in step 1.

The message list now contains only the message that corresponds to that ID. If the message is a hyperlink, click it to see an explanation and suggested action for the message. This can provide insight into why the message is suppressed or disabled. The following image shows how the Preferences dialog box appears when you enter `msgid:CPROP` in the filter field.



- 4 Click the  button to clear the filter field, and then repeat step 3 for each message ID you found in step 1.

- 5 Click the  button to clear the filter field.
- 6 Filter the messages to display those that are disabled by default and disabled in the Preferences pane by clicking the down arrow to the right of the filter field. Then, click **Show Disabled Messages**.
- 7 Review the message associated with each message ID to understand why it is suppressed in the code or disabled in Preferences.

Setting Code Analyzer Preferences

Use Code Analyzer preferences to adjust how M-Lint messages appear. These preferences apply to M-Lint messages in the Editor, the Embedded MATLAB® Editor (if you have products which use that tool), and the Code Analyzer Report, with a few exceptions. For more information, see “Preventing and Identifying Coding Problems” on page 9-107.

This section contains information about the following topics:

- Enable Integrated Warning and Error Messages
- Restricting Underlining in the Editor
- Changing the Color that the Code Analyzer Uses to Indicate an Automatic Fix Is Available
- Choosing the Messages to Display in Your Code
- Filtering the Messages Displayed in the Preferences Pane
- Saving Your Preferred Settings
- Using Your Saved Settings
- Using Default Settings
- “Enabling MATLAB Compiler Deployment Messages” on page 9-132

Enable Integrated Warning and Error Messages. Select this check box if you want the Editor to show M-Lint messages in the file. This preference does *not* apply to the Code Analyzer Report. When you select this preference, the Editor provides visual cues that alert you to potential errors, problems, and opportunities for improvement in your code. These visual cues take the form of underlines and a message indicator bar. From these cues, you can view a message for each line of a file that code analysis indicates you might be able to improve. For example, a common message is that a variable is defined but never used in the file.

Underlining

Restrict the underlining notification using the associated drop-down list. The list is available only when you select **Enable integrated warning and error messages**. Options are:

- Underline warnings and errors
- Underline errors only
- No underlines

Underlining for errors is red and for warnings is orange. For all of the underlining options, the Editor provides the marks for errors and warnings in the indicator bar.

You might choose a different option at different stages in your workflow. For example, when first coding, you might prefer no underlines because they would appear as you enter a statement and might be distracting. Later, you might choose to underline only errors to help you debug your file. Finally, when tweaking an existing file, you might want to underline warnings and errors because the file is in a state that you can fix any issues you introduce.

Autofix

Click **Adjust autofix highlight color** to open the Colors Programming Tools Preferences dialog box. This dialog box enables you adjust the color that highlights errors and warnings that MATLAB can autofix. By default, this

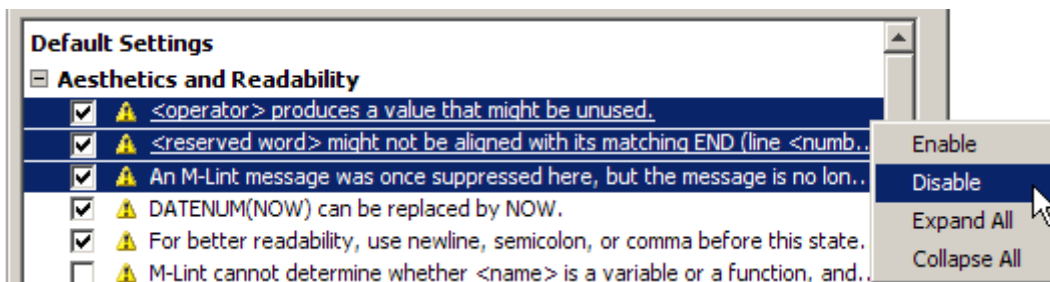
color is pale orange. Trigger autofix by clicking the **Fix** button in a code analyzer message. Autofixes are not available for all messages.

Active settings. When you have integrated warnings and error messages enabled, use Code Analyzer settings to show or hide specific M-Lint messages for your code. If you are new to using the code analysis or MATLAB, use the default settings. After you are familiar with code analysis and MATLAB, consider suppressing the display of certain M-Lint messages.

To suppress a message on a line-by-line or file-by-file basis, see “Suppressing Message Indicators and Messages” on page 9-117, or the `mlint` function.

To suppress messages in multiple files, it is more convenient to disable the Code Analyzer preference settings, as described here:

- 1 Use the filter field to filter the messages displayed in the Active settings table. For details, see [Filtering Messages](#) on page 127.
- 2 Click the link for a given message (if available) to get more information about that message, including an explanation and suggested action.
- 3 In the Active settings table, select check boxes or clear check boxes to enable or disable messages, respectively. To enable or disable a set of messages simultaneously, highlight the messages in the Active settings table, right-click, and then select **Enable** or **Disable**.



Enabled messages display and disabled messages do not display in the Editor.

- 4 Click **Apply** or **OK** to save the changes.

As with all preferences, MATLAB retains the settings for your next session.


Note The **MATLAB Compiler (deployment) messages** category is the only one that you can enable or disable by category. The Code Analyzer preferences pane only displays the **MATLAB Compiler (deployment) messages** category if you have MATLAB® Compiler™ installed.



Filtering Messages

You can filter the list of M-Lint messages in the Preferences dialog box to display only those messages that are currently of interest to you. Use any combination of the methods that the following table presents.

Note If you do not have the MATLAB Compiler installed, the Code Analyzer preferences pane does not display the **MATLAB Compiler (deployment) messages** category.

To see a list of messages that:	Perform this action:	Example Scenario
Contain a given string in the: <ul style="list-style-type: none"> • Short message • Extended message • Message category • Message ID 	Type the string in the filter field.	You recall seeing a message containing a certain string that you want to review, but you cannot remember the exact message text. For example, type <code>com</code> in the filter field to display those messages that contain that string in the short message, extended message, or message ID.
Correspond to a given message ID	Type <code>msgid:</code> followed by the message ID in the filter field.	You are reviewing the code that someone else wrote and you want to see the message that

To see a list of messages that:	Perform this action:	Example Scenario
		<p>corresponds to a suppressed one using the <code>%#ok<AGROW></code> pragma.</p> <p>Type <code>msgid:agrow</code> in the filter field. The message corresponding to <code>AGROW</code> is a link. Click it for more information about the message.</p> <p>Not all M-Lint messages have additional information. These messages do not appear as links.</p>
You can set using Code Analyzer Preferences	Click the down arrow to the right of the search field, and then click Show All .	You want to see the complete list of messages after you have filtered the messages on a given string or filter menu option.
Are different from the default setting (of enabled or disabled)	<p>Click the down arrow to the right of the search field, and then click Show Messages Modified from Default.</p> <p>A gray dot precedes a message with a setting different from the default. For example:</p> <p><input checked="" type="checkbox"/> <input type="checkbox"/>  DATENUM(NOW)</p>	A coworker gave you a settings file and you want to review each message that the coworker changed from its default setting.
Are in a given category	Click the down arrow to the right of the filter field, click Show Messages in Category , and then click the category you want.	<p>You want to review messages that describe coding practices that make it difficult for others to use your code.</p> <p>Click the down arrow to the right of the filter field, select Show Messages in Category, and then select Aesthetics and Readability.</p>

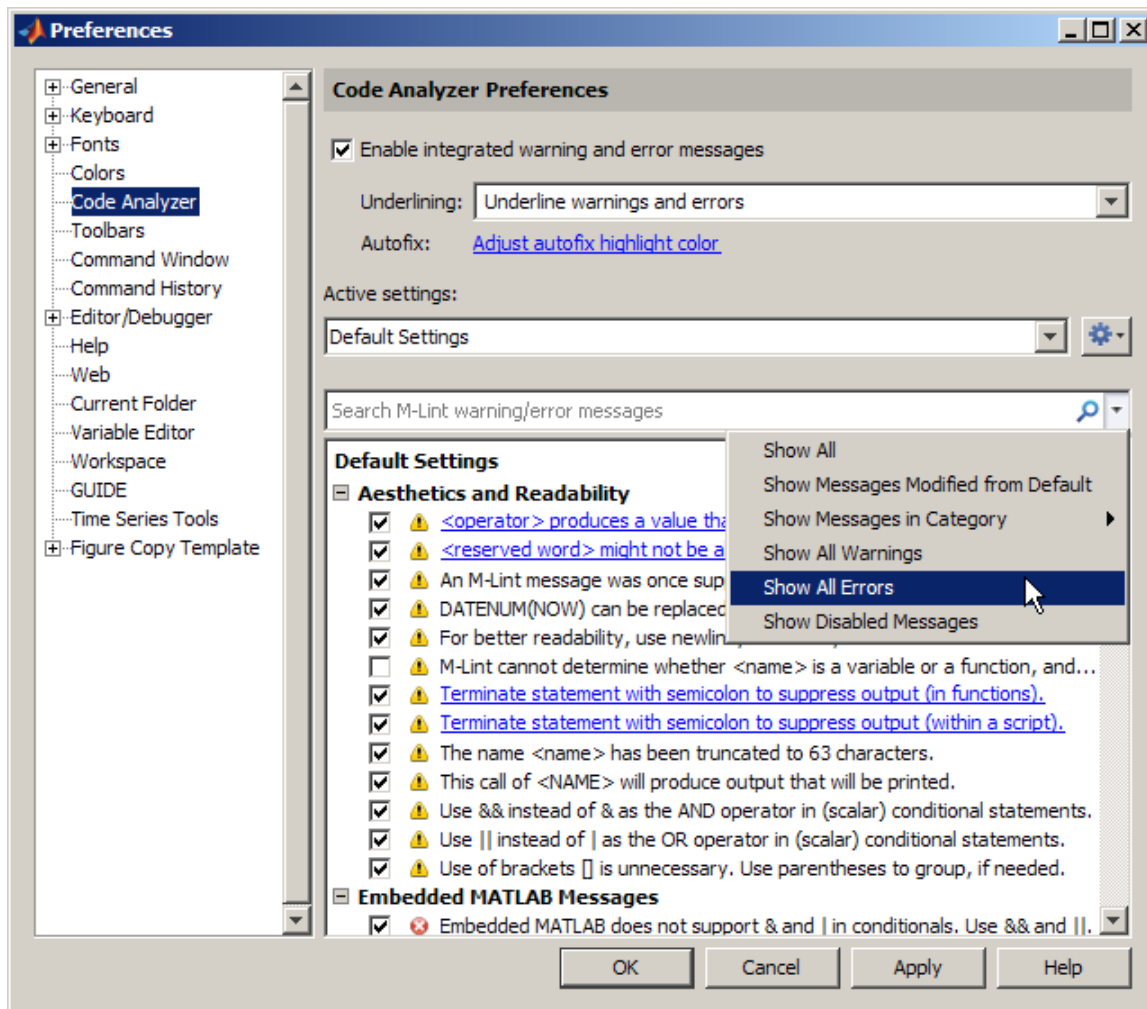
To see a list of messages that:	Perform this action:	Example Scenario
		Click the messages that appear as links for more information. Not all messages appear as links.
Are warnings	Click the down arrow to the right of the filter field, and then select Show All Warnings . An exclamation point in a yellow triangle  indicates a warning message.	You recall previous warnings that your code generated, but you cannot remember enough details to use the filter field to find it. You want to skim all the warning messages to find a particular one of interest.
Are errors	Click the down arrow to the right of the filter field, and then select Show All Errors . By default, an X in a red dot indicates an error message,  .	<p>You want to find a message that a script you worked on previously elicited. All you can recall is that it was an error and it involved parfor.</p> <p>Click the down arrow to the right of the filter field, and then select Show All Errors. Then, type a space and parfor in the filter field.</p> <p>The Code Analyzer preference pane displays only error messages that contain the word parfor.</p>
Are disabled	Click the down arrow to the right of the filter field, and then select Show Disabled Messages .	You want to see the messages that are disabled by default or that you have previously disabled.

Example of Filtering Messages

To display M-Lint error messages that contain the string comma and are disabled:


- 1 Click the arrow next to the filter field and select **Show All Errors**.

The filter field contains the string `severity:error`.




- 2 At the end of the string `severity:error`, press the **Space** key, and then type comma.
- 3 Click the arrow next to the filter field and select **Show Disabled Messages**.

The filter field now contains the string, `severity:error comma enabled:false`. Only the messages that fulfill those requirements appear in the Preferences pane.


To restore the list of all messages, click the Clear search button .

Saving Settings


If you are likely to use different settings at different times, or if you want to make these settings available to other users, click the Actions button , and then select **Save As**. MATLAB prompts you to provide the name of the txt file to which it will save the settings. The default location for settings is the MATLAB preferences folder (the folder returned when you run `prefdir`), although you can choose a different folder when saving.

Using Saved Settings

To use settings previously saved, select the settings txt file from the **Active settings** drop-down list. Or, select **Browse** from the **Active settings** drop-down list to locate the settings file. Then click **Apply** or **OK** to make the settings take effect. You can also access saved settings from within the Editor using **Tools > Code Analyzer**, or the indicator bar.

After selecting a settings file, you can modify the settings, but your changes automatically modify the txt file. If you want to retain the current settings in the txt file, create a copy of the settings file for modification. To do so, click the Actions button , select **Save As**, and then save the file to a different name. Make modifications to the newly named file.

Default Settings

The **Active settings** indicator shows **Default Settings** when you are using the default settings rather than settings from a txt file. The term (modified) appears when you modify the default settings, but have not yet saved the changes to a file. To undo any unsaved changes and return to the default settings, click the Actions button , and then select **Restore Defaults**. If

you think you will use the modified default settings in a future session, save the settings as described in Saving Settings on page 131.

Enabling MATLAB Compiler Deployment Messages. You can switch between showing or hiding Compiler deployment messages when you work on a file. Change the Code Analyzer preference for this message category. Your choice likely depends on whether you are working on a file to be deployed. When you change the preference, it also changes the setting in the Editor. The converse is also true—when you change the setting from the Editor, it effectively changes this preference. However, if the dialog box is open at the time you modify the setting in the Editor, you will not see the changes reflected in the Code Analyzer preferences dialog box. Whether you change the setting from the Editor or from the Code Analyzer preferences dialog box, it applies to the Editor and to the Code Analyzer Report.

To enable MATLAB Compiler deployment messages:

- 1 Select **File > Preferences > Code Analyzer**.
- 2 Click the down arrow next to the filter field, and then select **Show Messages in Category > MATLAB Compiler (deployment) messages**.
- 3 Click the **Enable Category** button.
- 4 Clear individual messages that you do not want to display for your code (if any).

The settings txt file, which you can create as described in Saving Settings on page 131, includes the status of this setting.

Understanding the Limitations of Code Analysis. Code analysis is a valuable tool, but there are some limitations:

- Sometimes, it fails to produce M-Lint messages where you expect them.
By design, code analysis attempts to minimize the number of incorrect messages it returns, even if this behavior allows some issues to go undetected.
- Sometimes, it produces messages that do not apply to your situation.

Links to additional information, when available in a message, can help you to make this determination. Error messages are almost always problems. However, many warnings are suggestions to look at something in the code that is unusual and therefore suspect, but might be correct in your case.

Suppress a warning message if you are certain that the message does not apply to your situation. If your reason for suppressing a message is subtle or obscure, include a comment giving the rationale. That way, those who read your code are aware of the situation.

For details, see “Suppressing Message Indicators and Messages” on page 9-117.

These sections describe code analysis limitations with respect to the following:

Distinguishing Function Names from Variable Names

Code analysis cannot always distinguish function names from variable names. For the following code, if the M-Lint message is enabled, code analysis returns the message, M-Lint cannot determine whether `xyz` is a variable or a function, and assumes it is a function. Code analysis cannot make a determination because `xyz` has no obvious value assigned to it. However, the program might have placed the value in the workspace in a way that code analysis cannot detect.

```
function y=foo(x)
    .
    .
    .
    y = xyz(x);
end
```

For example, in the following code, `xyz` can be a function, or can be a variable loaded from the MAT-file. Code analysis has no way of making a determination.

```
function y=foo(x)
    load abc.mat
    y = xyz(x);
end
```

Variables might also be undetected by code analysis when you use the `eval`, `evalc`, `evalin`, or `assignin` functions.

If code analysis mistakes a variable for a function, do one of the following:

- Initialize the variable so that code analysis does not treat it as a function.
- For the `load` function, specify the variable name explicitly in the `load` command line. For example:

```
function y=foo(x)
    load abc.mat xyz
    y = xyz(x);
end
```

Distinguishing Structures from Handle Objects

Code analysis cannot always distinguish structures from handle objects. In the following code, if `x` is a structure, you might expect an M-Lint message indicating that the code never uses the updated value of the structure. If `x` is a handle object, however, then this code can be correct.

```
function foo(x)
    x.a = 3;
end
```

Code analysis cannot determine whether `x` is a structure or a handle object. To minimize the number of incorrect messages, code analysis returns no message for the previous code, even though it might contain a subtle and serious bug.

Distinguishing Built-In Functions from Overloaded Functions

Code analysis does not use the MATLAB path information because it can be different, depending on whether you are editing or running the program. If some built-in functions are overloaded in a class or on the path, M-Lint messages might apply to the built-in function, but not to the overloaded function you are calling. In this case, suppress the message on the line where it appears or suppress it for the entire file.

For information on suppressing messages, see “Suppressing Message Indicators and Messages” on page 9-117.

Determining the Size or Shape of Variables

Code analysis has a limited ability to determine the type of variables and the shape of matrixes. Code analysis might produce M-Lint messages that are appropriate for the most common case, such as for vectors. However, these messages might be inappropriate for less common cases, such as for matrixes.

Analyzing Class Definitions with Superclasses

Because code analysis looks at one file at a time and does not use the path, it has no way to analyze superclasses. Therefore, the amount of checking that code analysis can provide for a class definition with superclasses is limited. In general, code analysis cannot always tell whether the class is a handle class. It makes an educated guess, but often cannot get enough information, to make a determination for certain.

Analyzing Methods

Most class methods must contain at least one argument that is an object of the same class as the method. But it does not always have to be the first argument. When it is, code analysis can determine that an argument is an object of the class you are defining, and it can do various checks. For example, it can check that the property and method names exist and are spelled correctly. However, when code analysis cannot determine that an object is an argument of the class you are defining, then it cannot provide these checks.

Determining Scope and Usage of Functions and Variables

Scoping issues can be the source of some coding problems. For instance, if you are unaware that nested functions share a particular variable, results of running your code might not be as you expect. Similarly, mistakes in usage of local, global, and persistent variables can cause unexpected results.

Code analysis does not always indicate scoping issues because sharing a variable across nested functions is not an error—it may be your intent. Use MATLAB function and variable highlighting features to identify when and where your code uses functions and variables. If you have an active Internet connection, you can watch the Variable and Function Highlighting video demo for an overview of the major features.

For conceptual information on nested functions and the various types of MATLAB variables, see “Variable Scope in Nested Functions” and “Types of Variables”.

Using Automatic Function and Variable Highlighting

By default, automatic function and local variable highlighting is enabled, as is nonlocal variable highlighting. To enable and disable highlighting or change the highlighting colors, select **File > Preferences > Colors > Programming tools**.

By default, the Editor:

- Highlights all instances of a function or local variable in sky blue when you place the cursor within a function or variable name. For instance:

```
collatz
```

- Displays nonlocal variable names in teal blue, regardless of the cursor location. For instance:

```
x
```

Example of Using Automatic Function and Variable Highlighting

Consider the code for a function rowsum:

```
function rowTotals = rowsum
% Add the values in each row and
% store them in a new array

x = ones(2,10);
[n, m] = size(x);
rowTotals = zeros(1,n);
```

```

for i = 1:n
    rowTotals(i) = addToSum;
end

function colsum = addToSum
    colsum = 0;
    thisrow = x(i,:);
    for i = 1:m
        colsum = colsum + thisrow(i);
    end
end

end

```

When you run this code, instead of returning the sum of the values in each row and displaying:

```

ans =

    10    10

```

MATLAB displays:

```

ans =

    0    0    0    0    0    0    0    0    0    10

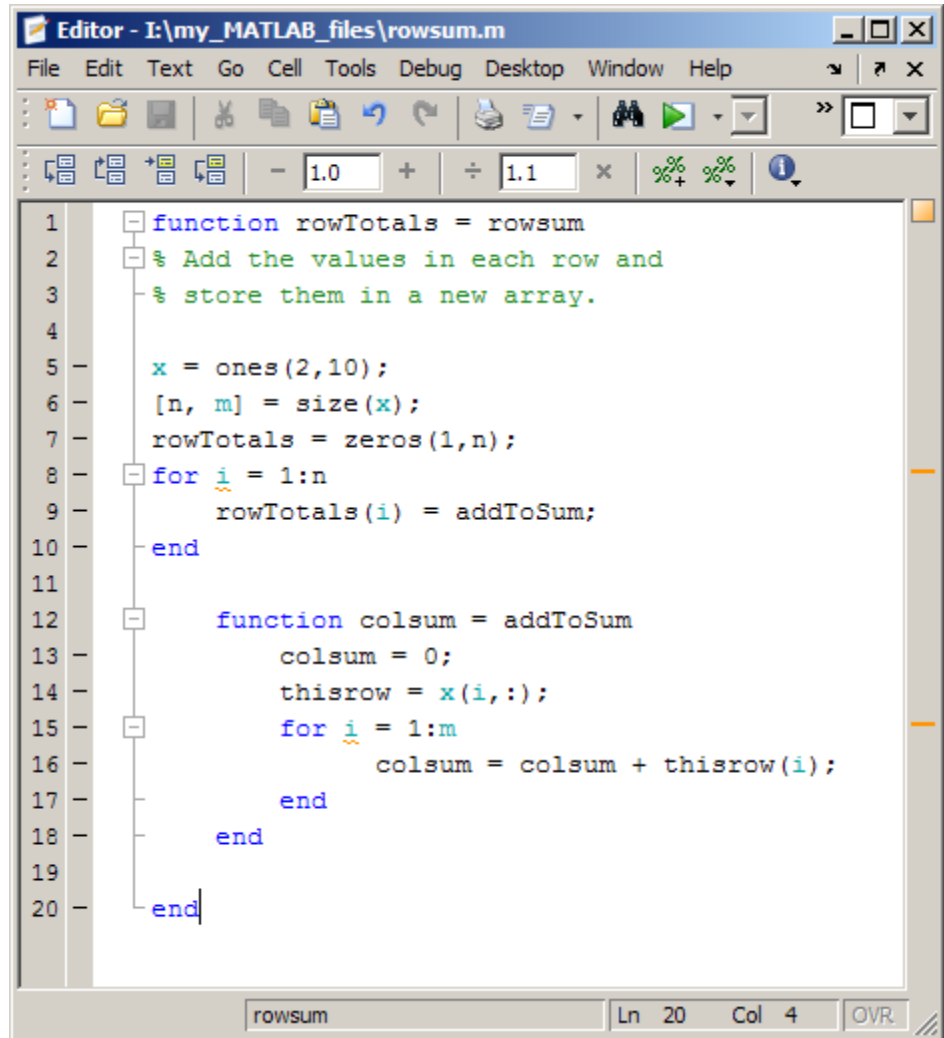
```

Examine the code by following these steps:

- 1 Copy the code into the Editor.

Notice the variable `i` appears in teal blue, which indicates `i` is not a local variable. Both the `rowTotals` function and the `addToSum` functions set and use the variable `i`.

The variable `n`, at line 6 appears in black, indicating that it does not span multiple functions.



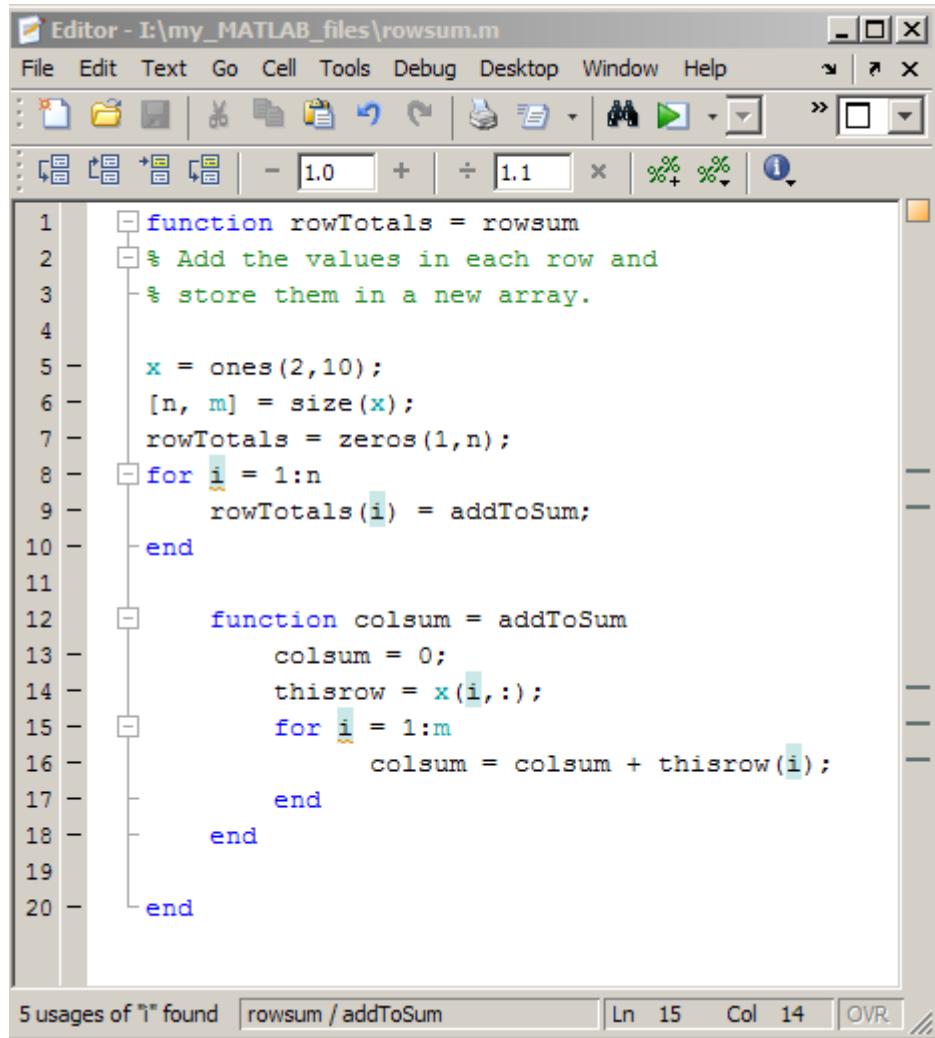
2 Hover the mouse pointer over an instance of variable `i`.

A tooltip appears: The scope of variable `i` spans multiple functions.

3 Click the tooltip link for information about variables whose scope span multiple functions.

4 Click an instance of `i`.

Every reference to `i` highlights in sky blue, the status bar indicates 5 usages of `i` found, and markers appear in the indicator bar on the right side of the Editor. One marker appears for each highlighted item.



```
1 function rowTotals = rowsum
2 % Add the values in each row and
3 % store them in a new array.
4
5 x = ones(2,10);
6 [n, m] = size(x);
7 rowTotals = zeros(1,n);
8 for i = 1:n
9     rowTotals(i) = addToSum;
10 end
11
12 function colsum = addToSum
13     colsum = 0;
14     thisrow = x(i,:);
15     for i = 1:m
16         colsum = colsum + thisrow(i);
17     end
18 end
19
20 end
```

5 usages of 'i' found | rowsum / addToSum | Ln 15 Col 14 | OVR

5 Hover over one of the indicator bar markers.

A tooltip appears and displays the name of the function or variable and the line of code represented by the marker.

- 6 Click a marker to navigate to the line indicated in tooltip for that marker.

This is particularly useful when your file contains more code than you can view at one time in the Editor.

Fix the code by changing the instance of `i` at line 15 to `y`.

You can see similar highlighting effects when you click on a function reference. For instance, click on `addToSum`.

Debugging Process and Features

In this section...

- “Ways to Debug MATLAB Files” on page 9-141
- “Preparing for Debugging” on page 9-141
- “Setting Breakpoints” on page 9-144
- “Running a File with Breakpoints” on page 9-148
- “Stepping Through a File” on page 9-150
- “Examining Values” on page 9-152
- “Correcting Problems and Ending Debugging” on page 9-158
- “Using Conditional Breakpoints” on page 9-166
- “Breakpoints in Anonymous Functions” on page 9-168
- “Breakpoints in Methods That Overload Functions” on page 9-169
- “Error Breakpoints” on page 9-170

Ways to Debug MATLAB Files

You can debug MATLAB files using the Editor, which is a graphical user interface, as well as by using debugging functions from the Command Window. You can use both methods interchangeably. These topics and the example describe both methods.

Preparing for Debugging

Do the following to prepare for debugging:

- 1** Open the file — To use the Editor for debugging, open it with the file to run.
- 2** Save changes — If you are editing the file, save the changes before you begin debugging. If you try to run a file with unsaved changes from within the Editor, the file is automatically saved before it runs. If you run a file with unsaved changes from the Command Window, MATLAB software runs the saved version of the file. Therefore, you do not see the results of your changes.

- 3 Add the files to a folder on the search path or put them in the current folder. Be sure the file you run and any files it calls are in folders that are on the search path. If all required files are in the same folder, you can instead make that folder the current folder.

Debugging Example – The Collatz Problem

The debugging process and features are best described using an example. To prepare to use the example, create two 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 resolves to 1 for all positive integers. The files for this example are useful for studying the Collatz problem. The file `collatz.m` generates the sequence of integers for any given n . The file `collatzplot.m` calculates the number of integers in the sequence for all integers from 1 through m , and plots the results. The plot shows patterns that you can study further.

Following are the results when n is 1, 2, or 3.

n	Sequence	Number of Integers in the Sequence
1	1	1
2	2 1	2
3	3 10 5 16 8 4 2 1	8

Files for the Collatz Problem. Following are the two files you use for the debugging example. To create these files on your system, open two new files. Select and copy the following code from the Help browser and paste it into the files. Save and name the files `collatz.m` and `collatzplot.m`. Save them to your current folder or add the folder where you save them to the search path. One of the files has an embedded error to illustrate the debugging features.

Open the files by issuing the following commands, and then saving each file to a local folder:

```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...  
             'examples','collatz.m'))
```

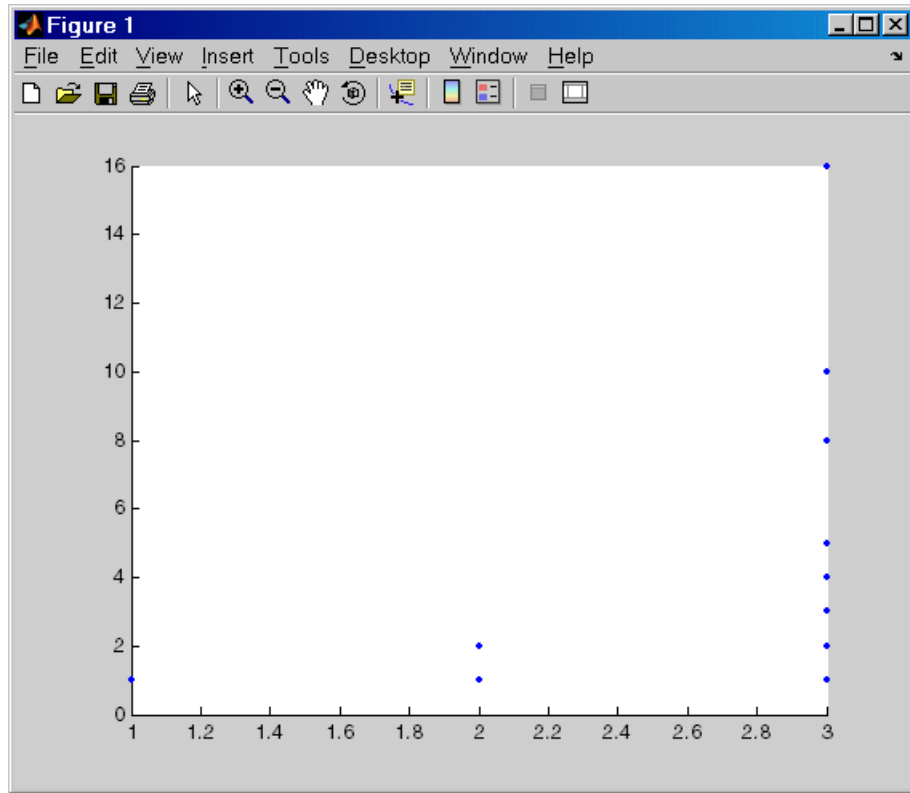
```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...  
             'examples','collatzplot.m'))
```

Trial Run for the Example. Open the file `collatzplot.m`. Make sure that the current folder is the folder in which you saved `collatzplot`.

Try out `collatzplot` to see if it works correctly. Use a simple input value, for example, 3, and compare the results to those shown in the preceding table. Typing

```
collatzplot(3)
```

produces the plot shown in the following figure.



The plot for $n = 1$ appears to be correct—for 1, the Collatz series is 1, and contains one integer. But for $n = 2$ and $n = 3$, it is wrong. There should be only one value plotted for each integer, the number of integers in the sequence, which the preceding table shows to be 2 (for $n = 2$) and 8 (for $n = 3$). Instead, multiple values are plotted. Use MATLAB debugging features to isolate the problem.

Setting Breakpoints

Set breakpoints to pause execution of the MATLAB file so you can examine values where you think the problem can be. You can set breakpoints in the Editor, using functions in the Command Window, or both.

There are three basic types of breakpoints you can set in MATLAB files:

- A standard breakpoint, which stops at a specified line in a file. For details, see “Setting Standard Breakpoints” on page 9-146.
- A conditional breakpoint, which stops at a specified line in a file only under specified conditions. For details, see “Using Conditional Breakpoints” on page 9-166.
- An error breakpoint that stops in any file when it produces the specified type of warning, error, or NaN or infinite value. For details, see “Error Breakpoints” on page 9-170.

You can disable standard and conditional breakpoints so that MATLAB temporarily ignores them, or you can remove them. For details, see “Disabling and Clearing Breakpoints” on page 9-159. Breakpoints do not persist 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 folder or in folders on the search path. When you add or remove a breakpoint in a file that is not in a folder on the search path or in the current folder, a dialog box appears. This dialog box presents options that allow you to add or remove the breakpoint. You can either change the current folder to the folder containing the file, or you can add the folder containing the file to the search path.

Do not set a breakpoint at a `for` statement if you want to examine values at increments in the loop. For example, in

```
for n = 1:10
    m = n+1;
end
```

MATLAB executes the `for` statement only once, which is efficient. Therefore, when you set a breakpoint at the `for` statement and step through the file, you only stop at the `for` statement once. Instead place the breakpoint at the next line, `m=n+1` to stop at each pass through the loop.

You cannot set breakpoints while MATLAB is busy, for example, running a file, unless that file is paused at a breakpoint.


Setting Standard Breakpoints

To set a standard breakpoint using the Editor:

- 1 If you have changed the file, save it.
- 2 Click in the breakpoint alley at an executable line where you want to set the breakpoint.
 - The *breakpoint alley* is the narrow column on the left side of the Editor, to the right of the line number.
 - Executable lines are preceded by a - (dash).

If you attempt to set breakpoints at lines that are not executable, such as comments or blank lines, MATLAB sets it at the next executable line.

Other ways to set a breakpoint are to:

- Position the cursor in an executable line and then click the Set/clear breakpoint button  on the toolbar.
- From the **Debug** menu, select **Set/Clear Breakpoint**.

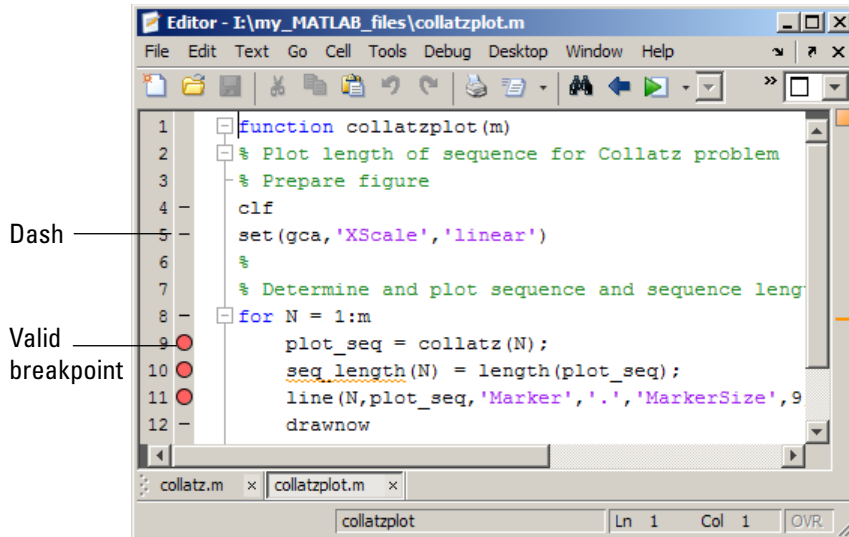
Setting Breakpoints for the Example. It is unclear whether the problem in the example is in `collatzplot` or `collatz`. To start, follow these steps:

- 1 In `collatzplot.m`, click the dash in the breakpoint alley at line 9 to set a breakpoint.

This breakpoint enables you to step into `collatz` to see if the problem is there.

- 2 Set additional breakpoints at lines 10 and 11.

These breakpoints stop the program so you can examine the interim results.

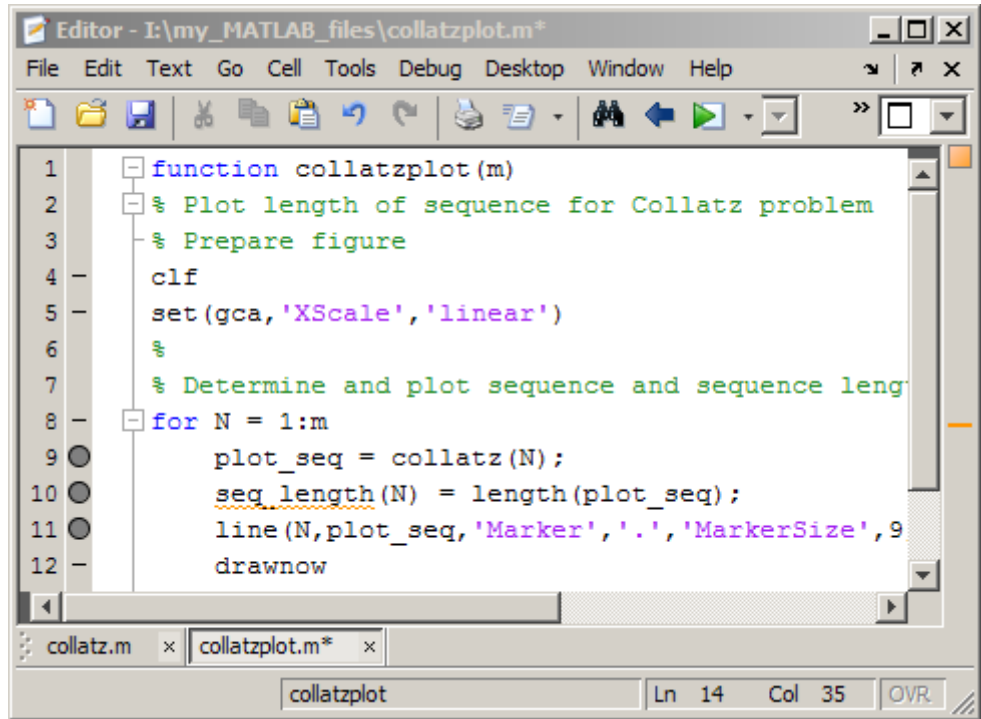


Understanding Valid (Red) and Invalid (Gray) Breakpoints. Red breakpoints indicate valid standard breakpoints.

Breakpoints are gray for either of these reasons:

- There are unsaved changes in the file. Save the file to make breakpoints valid. The gray breakpoints become red, indicating they are now valid. Any gray breakpoints that you entered at invalid breakpoint lines automatically move to the next valid breakpoint line with a 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.

The following image shows three invalid breakpoints at lines 9, 10, and 11. Notice the asterisk next to the file name in the title bar. It indicates the file has unsaved changes.



Function Alternative for Setting Breakpoints

To set a breakpoint using the debugging functions, use `dbstop`. For the example, type:

```
dbstop in collatzplot at 9
dbstop in collatzplot at 10
dbstop in collatzplot at 11
```

Running a File with Breakpoints

After setting breakpoints, run the file from the Command Window or the Editor.

Running the Example

For the example, run `collatzplot` for the simple input value, 3, by typing the following in the Command Window:

```
collatzplot(3)
```

The example, `collatzplot`, requires an input argument and therefore runs only from the Command Window or from a run configuration with a value specified.

Results of Running a File Containing Breakpoints

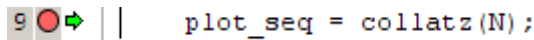
Running the file results in the following:

- The prompt in the Command Window changes to

```
K>>
```

indicating that MATLAB is in debug mode.

- The program pauses at the first breakpoint. This means that line will be executed when you continue. The pause is indicated in the Editor by the green arrow just to the right of the breakpoint. In the example, the pause is at line 9 of `collatzplot`, as shown here:



```
9 | plot_seq = collatz(N);
```

If you use debugging functions from the Command Window, the line at which you are paused is displayed in the Command Window. For the example, it would show:

```
9 plot_seq = collatz(N);
```

- The function displayed in the **Stack** field on the toolbar changes to reflect the current function (sometimes referred to as the caller or calling workspace). The call stack includes subfunctions as well as called functions. If you use debugging functions from the Command Window, use `dbstack` to view the current call stack.
- If the file you are running is not in the current folder or a folder on the search path, you are prompted to either add the folder to the path or change the current folder.

In debug mode, you can set breakpoints, step through programs, examine variables, and run other functions.

MATLAB software could become nonresponsive if it stops at a breakpoint while displaying a modal dialog box or figure that your file creates. In that event, press **Ctrl+C** to go the MATLAB prompt.


Stepping Through a File






While debugging, you can step through a MATLAB file, pausing at points where you want to examine values.

Use any of the following methods:


- The Editor toolbar buttons
If the toolbar is not displaying, right-click the Editor menu bar, and then select **Editor Toolbar**.
- Step options in the Editor **Debug** menu
- The `dbstep` or `dbcont` function

Details about these methods appear in the following table.


Toolbar Button	Debug Menu Item	Description	Function Alternative
	Run <i>file</i> or Run Configuration for <i>file</i>	Commence execution of file and run until completion or until a breakpoint is encountered. The Run Configurations for <i>file</i> menu option provides a submenu. The submenu enables you to select a particular run configuration or to edit the run configurations for the MATLAB file. If you choose Run <i>file</i> , MATLAB uses the default run configuration.	None

Toolbar Button	Debug Menu Item	Description	Function Alternative
None	Go Until Cursor	Continue execution of file until the line where the cursor is positioned. Also available on the context menu.	None
	Step	Execute the current line of the file.	dbstep
	Step In	Execute the current line of the file and, if the line is a call to another function, step into that function.	dbstep in
	Continue	Resume execution of file until completion or until another breakpoint is encountered.	dbcont
	Step Out	After stepping in, run the rest of the called function or subfunction, leave the called function, and pause.	dbstep out
	Exit Debug Mode	Exit debug mode.	dbquit out

Continue Running in the Example

In the example, `collatzplot` is paused at line 9. Because the problem results are correct for $N/n = 1$, continue running until $N/n = 2$. Press the Continue button  three times to move through the breakpoints at lines 9, 10, and 11. Now the program is again paused at the breakpoint at line 9.

Stepping into the Called Function in the Example

Now that `collatzplot` is paused at line 9 during the second iteration, use the Step In button  or type `dbstep in` in the Command Window to step into `collatz` and walk through that file. Stepping into line 9 of `collatzplot` goes to line 9 of `collatz`. If `collatz` is not open in the Editor, it automatically opens if you have selected **Debug > Open Files When Debugging**.

The pause indicator at line 9 of `collatzplot` changes to a hollow arrow ⇨, indicating that MATLAB control is now in a subfunction called from the main program. The call stack shows that the current function is now `collatz`.

In the called function, `collatz` in the example, you can do the same things you can do in the main (calling) function—set breakpoints, run, step through, and examine values.

Examining Values

While the program is paused, you can view the value of any variable currently in the workspace. Examine values when you want to see whether a line of code has produced the expected result or not. If the result is as expected, continue running or step to the next line. If the result is not as you expect, then that line, or a previous line, contains an error. Use the following methods to examine values:

- “Selecting the Workspace” on page 9-152
- “Viewing Values as Data Tips in the Editor” on page 9-153
- “Viewing Values in the Command Window” on page 9-154
- “Viewing Values in the Workspace Browser and Variable Editor” on page 9-155
- “Evaluating a Selection” on page 9-156
- “Examining Values in the Example” on page 9-156
- “Problems Viewing Variable Values from the Parent Workspace” on page 9-157

Many of these methods are used in “Examining Values in the Example” on page 9-156.

Selecting the Workspace

Variables assigned through the Command Window and created using scripts are considered to be in the base workspace. Variables created in a function belong to their own function workspace. To examine a variable, you must first select its workspace. When you run a program, the current workspace is shown in the **Stack** field. To examine values that are part of another

workspace for a currently running function or for the base workspace, first select that workspace from the list in the **Stack** field.

If you use debugging functions from the Command Window:

- To display the call stack, use `dbstack`.
- To change to a different workspace, use `dbup` and `dbdown`.
- To list the variables in the current workspace, use `who` or `whos`.

Workspace in the Example. At line 9 of `collatzplot`, you stepped in, and the current line is 9 in `collatz`. The **Stack** field shows that `collatz` is the current workspace.

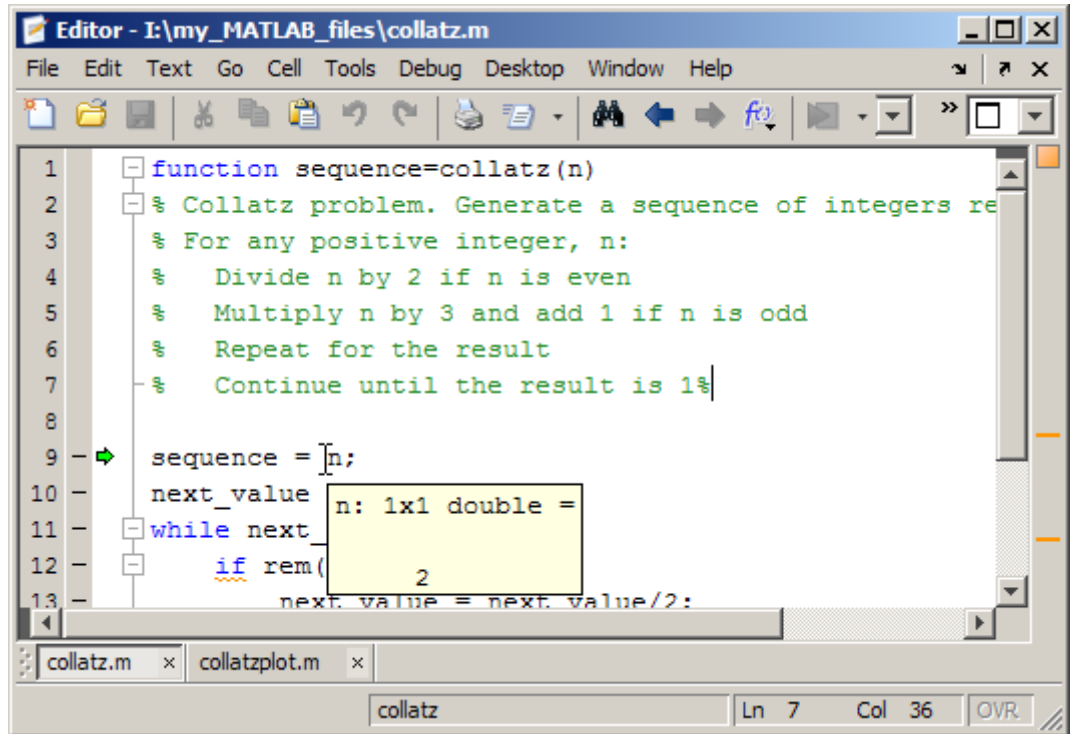
Viewing Values as Data Tips in the Editor

In the Editor, position the mouse pointer to the left of a variable. Its current value appears in a data tip, which is like a tooltip for data. The data tip stays in view until you move the pointer. If you have trouble getting the data tip to appear, click in the line containing the variable and then move the pointer next to the variable.

A related function is `datatipinfo`.

You must enable the **Enable datatips in edit mode** preference, which is disabled by default. For details, see “Setting Display Preferences” on page 9-16.

Data Tips in the Example. Position the mouse pointer over `n` in line 9 of `collatz`. The data tip shows that `n = 2`, as expected.



Viewing Values in the Command Window

You can examine values while in debug mode at the `K>>` prompt. To see the variables currently in the workspace, use `who`. Type a variable name in the Command Window and it displays the variable's current value. For the example, to see the value of `n`, type

```
n
```

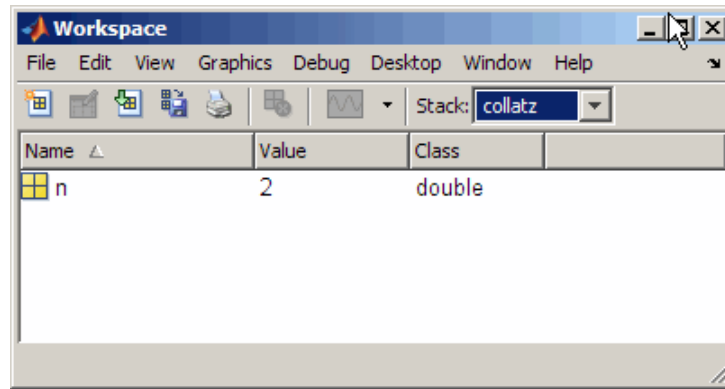
The Command Window displays the expected result

```
n =
    2
```

and displays the debug prompt, `K>>`.

Viewing Values in the Workspace Browser and Variable Editor

You can view the value of variables in the **Value** column of the Workspace browser. The Workspace browser displays all variables in the current workspace. Use **Stack** in the Workspace browser to change to another workspace and view its variables.

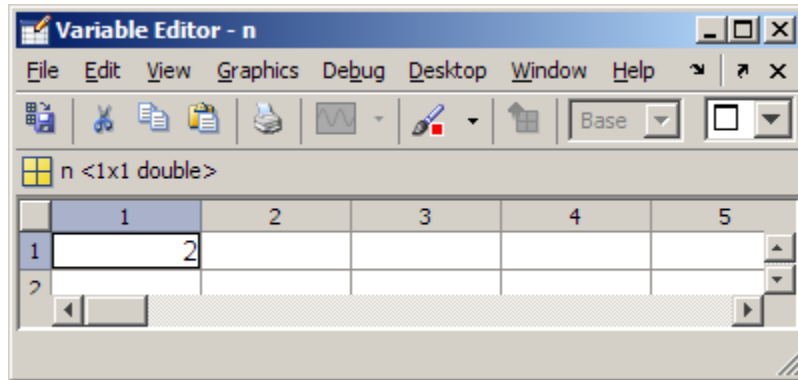


The **Value** column does not show all details for all variables. To see details, double-click a variable in the Workspace browser. The Variable Editor opens, displaying the content for that variable. You can open the Variable Editor directly for a variable using `openvar`.

To see the value of `n` in the Variable Editor for the example, type

```
openvar n
```

and the Variable Editor opens, showing that `n = 2` as expected.



Evaluating a Selection

Select a variable or equation in a MATLAB file in the Editor. Right-click and select **Evaluate Selection** from the context menu (for a single-button mouse, press **Ctrl+click**). The Command Window displays the value of the variable or equation. You cannot evaluate a selection while MATLAB is busy, for example, running a file.

Examining Values in the Example

Step from line 9 through line 13 in `collatz`. Step again, and the pause indicator jumps to line 17, just after the `if` loop, as expected. Step again, to line 18, check the value of `sequence` in line 17 and see that the array is

```
2 1
```

as expected for $n = 2$. Step again, which moves the pause indicator from line 18 to line 11. At line 11, step again. Because `next_value` is now 1, the `while` loop ends. The pause indicator is at line 11 and appears as a green down arrow \blacktriangledown . This indicates that processing in the called function is complete and program control will return to the calling program. Step again from line 11 in `collatz` and execution is now paused at line 9 in `collatzplot`.

Note that instead of stepping through `collatz`, the called function, as was just done in this example, you can step out from a called function back to the calling function, which automatically runs the rest of the called function and returns to the next line in the calling function. To step out, use the Step Out button or type `dbstep out` in the Command Window.

In `collatzplot`, step again to advance to line 10, and then to line 11. The variable `seq_length` in line 10 is a vector with the elements:

```
1  2
```

which is correct.

Finally, step again to advance to line 12. Examining the values in line 11, `N = 2` as expected, but the second variable, `plot_seq`, has two values, where only one value is expected. While the value for `plot_seq` is as expected:

```
2  1
```

it is the incorrect variable for plotting. Instead, `seq_length(N)` should be plotted.

Problems Viewing Variable Values from the Parent Workspace

Sometimes, if you set a breakpoint in a function, and then attempt to view the value of a variable in the parent workspace using the `dbup` command, the value of the variable is currently under construction. Therefore, the value is not available. This is true whether you view the value by specifying the `dbup` command in the Command Window or by using the **Stack** field on the Editor toolbar.

In such cases, MATLAB returns the following message, where *x* is the variable for which you are trying to examine the value:

```
K>> x
??? Reference to a called function result under construction x.
```

For example, suppose you have code such as the following:

```
x = collatz(x);
```

MATLAB detects that the evaluation of `collatz(x)` replaces the input variable, *x*. To optimize memory use, MATLAB overwrites the memory that *x* currently occupies to hold a new value for *x*. When you request the value of *x*, and it is under construction, its value is not available, and MATLAB displays the error message.

Correcting Problems and Ending Debugging

The following are some of the ways to correct problems and end the debugging session:

- “Changing Values and Checking Results” on page 9-158
- “Ending Debugging” on page 9-158
- “Disabling and Clearing Breakpoints” on page 9-159
- “Saving Breakpoints” on page 9-161
- “Correcting Problems in a MATLAB File” on page 9-161
- “Completing the Example” on page 9-162
- “Running Sections in MATLAB Files That Have Unsaved Changes” on page 9-165

“Completing the Example” on page 9-162 uses many of these features.

Changing Values and Checking Results


While debugging, you can change the value of a variable in the current workspace to see if the new value produces expected results. While the program is paused, assign a new value to the variable in the Command Window, Workspace browser, or Variable Editor. Then continue running or stepping through the program. If the new value does not produce the expected results, the program has a different problem.

Ending Debugging

After identifying a problem, end the debugging session. You must end a debugging session if you want to change and save a file to correct a problem, or if you want to run other functions in MATLAB.

Note Quit debug mode before editing a file. If you edit a file while in debug mode, you can get unexpected results when you run the file. If you do edit a file while in debug mode, breakpoints turn gray, indicating that results might not be reliable. See “Understanding Valid (Red) and Invalid (Gray) Breakpoints” on page 9-147 for details.

If you attempt to save an edited file while in debug mode, a dialog box opens allowing you to exit debug mode and save the file.

To end debugging, click the Exit debug mode button , or select **Exit Debug Mode** from the **Debug** menu.

You can instead use the function `dbquit` or the **Shift+F5** keyboard shortcut to end debugging.

After quitting debugging, pause indicators in the Editor display no longer appear, and the normal prompt `>>` appears in the Command Window instead of the debugging prompt, `K>>`. You can no longer access the call stack.

Disabling and Clearing Breakpoints


Disable a breakpoint to ignore it temporarily. Clear a breakpoint to remove it.

Disabling and Enabling Breakpoints. You can disable selected breakpoints so the program temporarily ignores them and runs uninterrupted, for example, after you think you identified and corrected a problem. This is especially useful for conditional breakpoints—see “Using Conditional Breakpoints” on page 9-166.

To disable a breakpoint, do one of the following:

- Right-click the breakpoint icon and select **Disable Breakpoint** from the context menu.
- Click anywhere in a line and select **Enable/Disable Breakpoint** from the **Debug** menu.

To disable a conditional breakpoint, use either of the methods in the preceding list or click the conditional breakpoint icon. An X appears through the breakpoint icon as shown here.

```
9  plot_seq = collatz(N);
```

When you run `dbstatus`, the resulting message for a disabled breakpoint is

```
Breakpoint on line 9 has conditional expression 'false'.
```

After disabling a breakpoint, you can reenable it to make it active again or you can clear it.

To reenable a breakpoint, do either of the following:

- Right-click the breakpoint icon and select **Enable Breakpoint** from the context menu.
- Click anywhere in a line and select **Enable/Disable Breakpoint** from the **Debug** menu.

The X no longer appears on the breakpoint icon and program execution will pause at that line.


Clearing (Removing) Breakpoints. All breakpoints remain in a file until you clear (remove) them or until they are cleared automatically. Clear a breakpoint after determining that a line of code is not causing a problem.

To clear a breakpoint in the Editor:

- Click anywhere in a line that has a breakpoint and select **Set/Clear Breakpoint** from the **Debug** menu.
- Click a standard breakpoint icon, or a disabled conditional breakpoint icon.
- Use the `dbclear in file at lineno` command in the Command Window. For the example, clear the breakpoint at line 9 in `collatzplot` by typing:

```
dbclear in collatzplot at 9
```

To clear all breakpoints in all files:

- Select **Debug > Clear Breakpoints in All Files** on the toolbar.
- Click the Clear breakpoints in all files button .
- Use `dbclear all` in the Command Window.

For the example, clear all of the breakpoints in `collatzplot` by typing:

```
dbclear all in collatzplot
```

Breakpoints clear automatically when you:

- End the MATLAB session.
- Clear the file using `clear name` or `clear all`.

Note When `clear name` or `clear all` is in a statement in a file that you are debugging, it clears the breakpoints.

Saving Breakpoints

You can use the `s=dbstatus` syntax and then save `s` to save the current breakpoints to a MAT-file. At a later time, you can load `s` and restore the breakpoints using `dbstop(s)`. For more information, including an example, see the `dbstatus` reference page.

Correcting Problems in a MATLAB File

To correct a problem in a MATLAB file:

- 1 Quit debugging.

Do not modify a file while MATLAB is in debug mode. If you do, breakpoints turn gray, indicating that results might not be reliable. See “Understanding Valid (Red) and Invalid (Gray) Breakpoints” on page 9-147 for details.

- 2 Modify the file.
- 3 Save the file.

- 4 Set, disable, or clear breakpoints, as appropriate.
- 5 Run the file again to be sure that it produces the expected results.

Completing the Example

To correct the problem in the example:

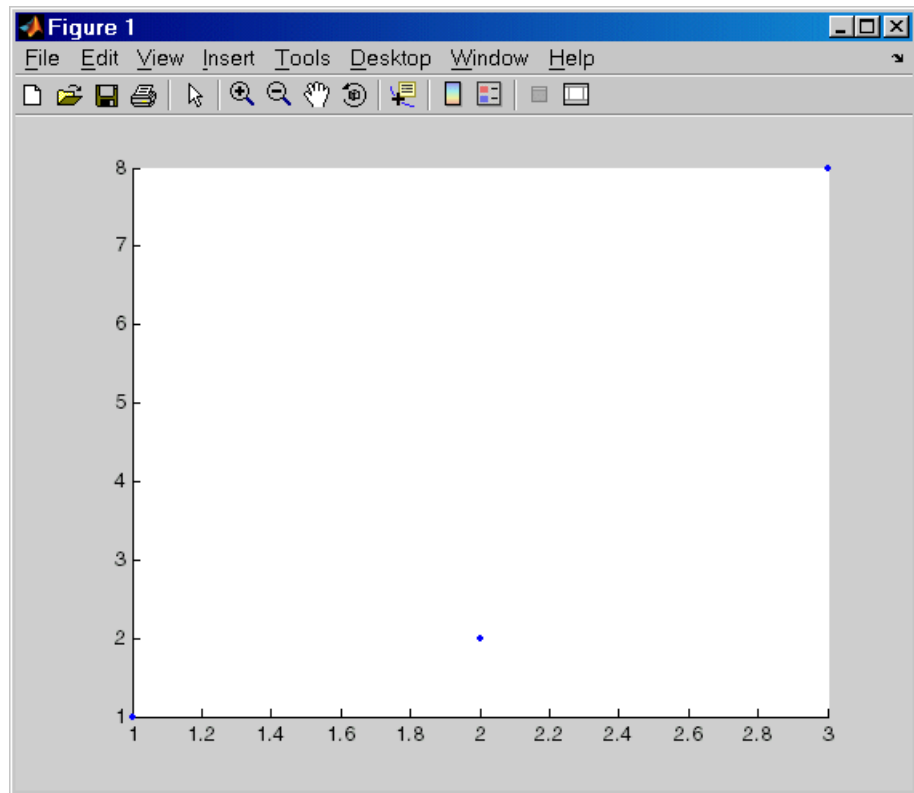
- 1 End the debugging session. One way to do this is to select **Exit Debug Mode** from the **Debug** menu.
- 2 In `collatzplot.m` line 11, change the string `plot_seq` to `seq_length(N)` and save the file.
- 3 Clear the breakpoints in `collatzplot.m`. One way to do this is by typing

```
dbclear all in collatzplot
```


in the Command Window.
- 4 Run `collatzplot` for `m = 3` by typing

```
collatzplot(3)
```


in the Command Window.
- 5 Verify the result. The figure shows that the length of the Collatz series is 1 when $n = 1$, 2 when $n = 2$, and 8 when $n = 3$, as expected.



- 6 Test the function for a slightly larger value of m , such as 6, to be sure that the results are still accurate. To make it easier to verify `collatzplot` for $m = 6$ as well as the results for `collatz`, add this line at the end of `collatz.m`

```
sequence
```

which displays the series in the Command Window. The results for when $n = 6$ are

```
sequence =
```

```
6      3      10      5      16      8      4      2      1
```

Then run `collatzplot` for $m = 6$ by typing

```
collatzplot(6)
```

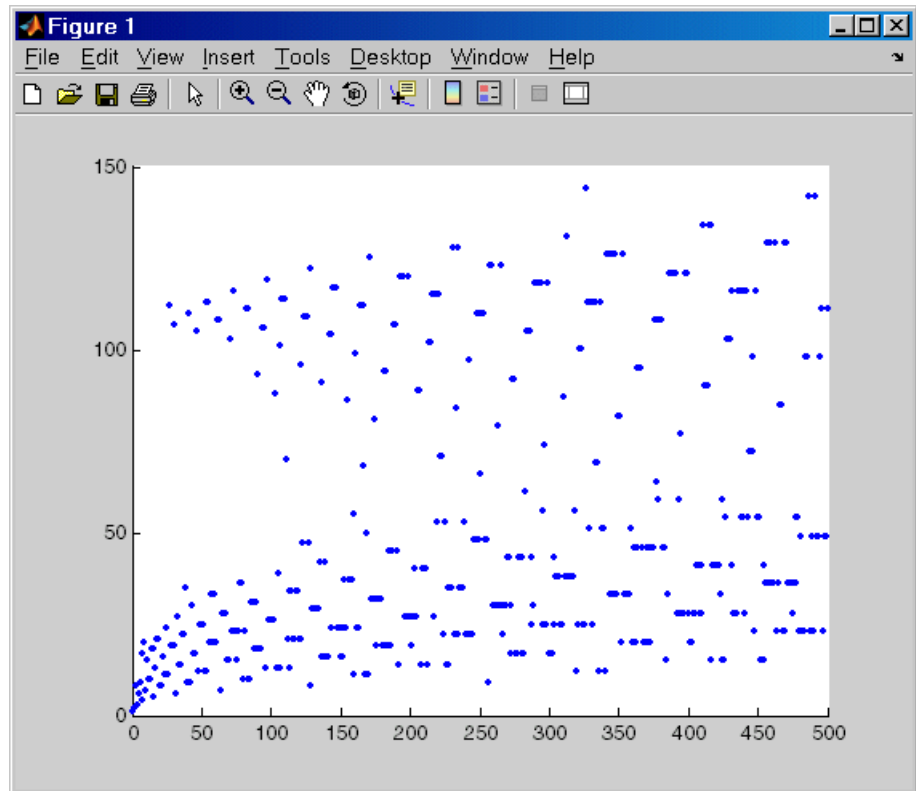
- 7** To make debugging easier, you ran `collatzplot` for a small value of `m`. Now that you know it works correctly, run `collatzplot` for a larger value to produce more interesting results. Before doing so, you consider disabling output for the line you just added in step 6, line 19 of `collatz.m`, by adding a semicolon to the end of the line so it appears as

```
sequence;
```

Then run

```
collatzplot(500)
```

The following figure shows the lengths of the Collatz series for $n = 1$ through $n = 500$.



Running Sections in MATLAB Files That Have Unsaved Changes

It is a good practice to modify a MATLAB file after you quit debugging, and then save the modification and run the file. Otherwise, you might get unexpected results. However, there are situations where you want to experiment during debugging. Perhaps you want to modify a part of the file that has not yet run, and then run the remainder of the file without saving the change. Follow these steps:

- 1 While stopped at a breakpoint, modify a part of the file that has not yet run.

Breakpoints turn gray, indicating they are invalid.

- 2 Select all of the code after the breakpoint, right-click, and then select **Evaluate Selection** from the context menu.

You can also use cell mode to do this.

Using 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 stop only if `N` is greater than or equal to 2. This section covers the following topics:

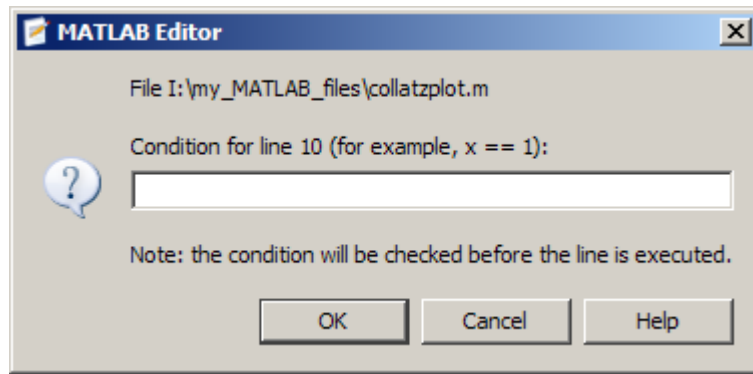
- “Setting Conditional Breakpoints” on page 9-166
- “Modifying, Disabling, and Clearing Conditional Breakpoints” on page 9-167
- “Function Alternatives for Manipulating Conditional Breakpoints” on page 9-168

Setting Conditional Breakpoints

To set a conditional breakpoint:

- 1 Click in the line where you want to set the conditional breakpoint.
- 2 From the **Debug** menu, select **Set/Modify Conditional Breakpoint**. If a standard breakpoint exists at that line, use this same method to make it conditional.

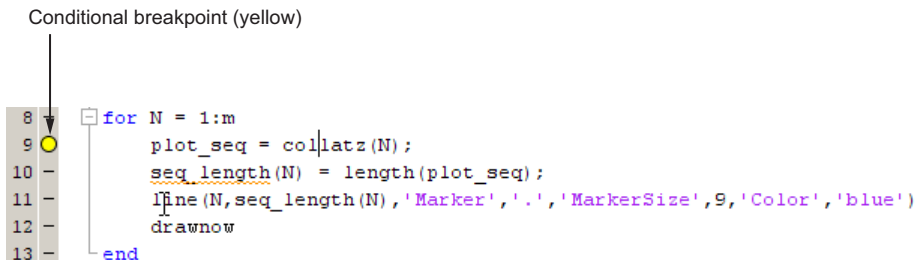
The MATLAB Editor conditional breakpoint dialog box opens as shown in this example.



- 3 Type a condition in the dialog box, where a condition is any valid MATLAB expression that returns a logical scalar value. Click **OK**. As noted in the dialog box, the condition is evaluated before running the line. For the example, at line 9 in `collatzplot`, enter the following as the condition:

`N>=2`

A yellow breakpoint icon (indicating the breakpoint is conditional) appears in the breakpoint alley at that line.



When you run the file, MATLAB software enters debug mode and pauses at the line only when the condition is met. In the `collatzplot` example, MATLAB runs through the `for` loop once and pauses on the second iteration at line 9 when `N` is 2. If you continue executing, MATLAB pauses again at line 9 on the third iteration when `N` is 3.

Modifying, Disabling, and Clearing Conditional Breakpoints

The following table describes how to adjust conditional breakpoints.

To	Do This
Modify a condition for a breakpoint in the current line.	Right-click the conditional breakpoint icon, and then from the context menu, select Set/Modify Condition .
Disable a conditional breakpoint.	Click the associated conditional breakpoint icon.
Clear a conditional breakpoint.	Double-click the associated conditional breakpoint icon.

Function Alternatives for Manipulating Conditional Breakpoints


The following table lists the functions available for adjusting conditional breakpoints from the Command Window.

To	Use This Function
Set a conditional breakpoint.	<code>dbstop</code>
Clear a conditional breakpoint.	<code>dbclear</code>
View a list of currently set breakpoints, including the conditional expression for each conditional breakpoint.	<code>dbstatus</code>

Breakpoints in Anonymous Functions

You can set multiple breakpoints in a line of MATLAB code that contains anonymous functions. You can do both of the following:

- Set a breakpoint for the line itself (MATLAB software stops at the start of the line).
- Set a breakpoint for each anonymous function in that line.

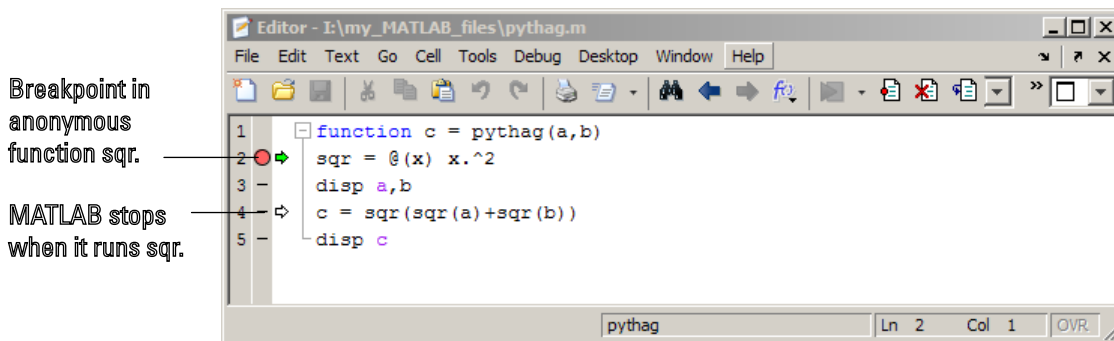
When you add a breakpoint to a line containing an anonymous function, the Editor asks where in the line you want to add the breakpoint. If there is more than one breakpoint in a line, the breakpoint icon is blue , regardless of the status of any of the breakpoints on the line.

To display information in a tooltip about all breakpoints on a line, position the pointer on the blue icon.

To perform a breakpoint action for a line that can contain multiple breakpoints, such as **Clear Breakpoint**, right-click the breakpoint alley at that line, and then select the action.

When you set a breakpoint in an anonymous function, MATLAB stops when the anonymous function is called.

The following illustration shows the Editor when you set a breakpoint in the anonymous function `sqr` in line 2, and then run the file. MATLAB stops when it runs `sqr` in line 4. After you continue execution, MATLAB stops again when it runs `sqr` the second time in line 4. The **Stack** display shows the anonymous function.



Breakpoints in Methods That Overload Functions

MATLAB functions often call other MATLAB functions and methods to perform their operations. If you set a breakpoint in a class method, and then run a MATLAB function that results in calling that method, execution stops at the breakpoint. This behavior can be confusing if you are unaware that the MATLAB function calls the method containing the breakpoint.

For instance, suppose you do the following:

- 1 Define a class named `MyClass` that overloads the MATLAB `size` function.

- 2 Create an instance of `MyClass`.
- 3 Insert breakpoints within the `MyClass` `size` method.
- 4 Call `whos`.

When you call the `whos` function, it calls the `size` function to obtain size information about the variables in the workspace. Under the preceding circumstances, because `MyClass` overloads the `size` function, `whos` calls the `MyClass` `size` method instead of the default `size` function to determine the size of the `MyClass` object. Execution stops at the breakpoint you set in the `size` method. You can enable the MATLAB function to execute to completion by either stepping or continuing through the method. To prevent this behavior from recurring, remove the breakpoints.

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 by using the error breakpoint occurs. MATLAB then enters debug mode and opens the file containing the error, with the pause indicator at the line containing the error. Files open only when you select **Debug > Open 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 and Clearing Error Breakpoints” on page 9-170
- “Error Breakpoint Types and Options” on page 9-171
- “Examples of Setting Warning and Error Breakpoints” on page 9-172
- “Function Alternative for Manipulating Error Breakpoints” on page 9-174

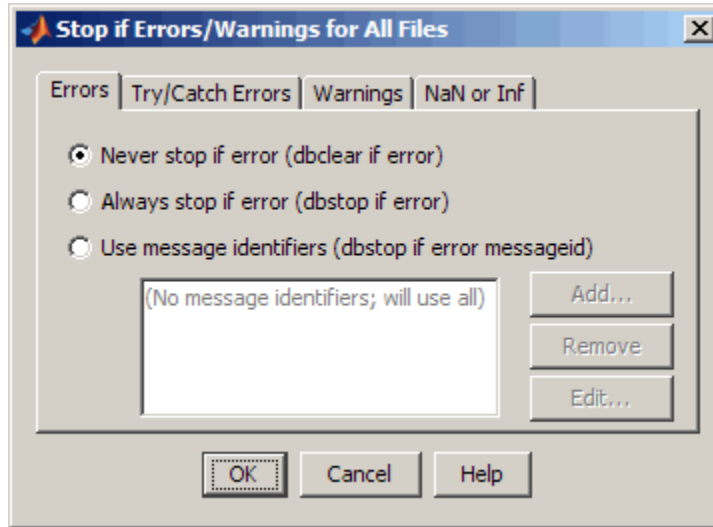
Setting and Clearing Error Breakpoints

To set error breakpoints:

- 1 Select **Debug > Stop if Errors/Warnings**.

- 2 In the Stop if Errors/Warnings for All Files dialog box, specify error breakpoints on all appropriate tabs, and then click **OK**.

To clear error breakpoints, select the **Never stop if ...** option for all appropriate tabs, and then click **OK**.



Error Breakpoint Types and Options

As the tabs in the Stop if Errors/Warnings for All Files dialog box suggest, there are four basic types of error breakpoints you can set:

- **Errors**

When an error occurs, execution stops, unless the error is in a `try...catch` block. MATLAB enters debug mode and opens the file to the line in the `try` portion of the block that produced the error. You cannot resume execution.

- **Try/Catch Errors**

When an error occurs in a `try...catch` block, execution pauses. MATLAB enters debug mode and opens the file to the line that produced the error. You can resume execution or use debugging features.

- **Warnings**

When a warning occurs, MATLAB pauses, enters debug mode, and opens the file, paused at the line that produced the warning. You can resume execution or use debugging features.

- **NaN or Inf**

When an operator, function call, or scalar assignment produces a NaN (not-a-number) or Inf (infinite) value, MATLAB pauses, enters debug mode, and opens the file. MATLAB pauses immediately after the line that encountered the value. You can resume execution or use debugging features.

Select options for these error breakpoints:

- Click **Never stop if error...** on a tab to clear that type of breakpoint.
- Click **Always stop if error...** on a tab to set that type of breakpoint.
- Select **Use message identifiers...** on a tab to limit each type of error breakpoint (except NaN or Inf). Execution stops only for the error you specify by the corresponding message identifier.

This option is not available for the **NaN or Inf** type of error breakpoint. You can add multiple message identifiers, and edit or remove them.

Examples of Setting Warning and Error Breakpoints

Pausing Executing for Warnings. To pause execution when MATLAB produces a warning:

- 1 Click the **Warnings** tab.
- 2 Click **Always stop if warning**, and then click **OK**.

Now, when you run a file and MATLAB produces a warning, execution pauses and MATLAB enters debug mode. The file opens in the Editor at the line that produced the warning.

Setting Breakpoints for a Specific Error. To stop execution for a specific error add a message identifier:

- 1** Click the **Errors, Try/Catch Errors, or Warnings** tab.
- 2** Click **Use Message Identifiers**.
- 3** Click **Add**.
- 4** In the resulting Add Message Identifier dialog box, type the message identifier of the error for which you want to stop. The identifier is of the form `component:message` (for example, `MATLAB:nargchk:notEnoughInputs`). Then click **OK**.

The message identifier you specified appears in the Stop if Errors/Warnings for All Files dialog box.

- 5** Click **OK**.

Obtaining Error Message Identifiers. To obtain an error message identifier generated by a MATLAB function, run the function to produce the error, and then call `MException.last`. For example:

```
surf
MException.last
```

The Command Window displays the `MException` object, including the error message identifier in the `identifier` field. For this example, it displays:

```
ans =

MException

Properties:
  identifier: 'MATLAB:nargchk:notEnoughInputs'
  message: 'Not enough input arguments.'
  cause: {}
  stack: [1x1 struct]

Methods
```

Obtaining Warning Message Identifiers. To obtain a warning message identifier generated by a MATLAB function, run the function to produce the warning. Then, run:

```
[m,id] = lastwarn
```

MATLAB returns the last warning identifier to `id`. An example of a warning message identifier is `MATLAB:concatenation:integerInteraction`.

Function Alternative for Manipulating Error Breakpoints

The function equivalent for each option in the Stop if Errors/Warnings for All Files dialog box, appears to the right of each option. For example, the function equivalent for **Always stop if error** is `dbstop if error`. Use these functions in the Command Window as listed in the following table.

To	Use This Function
Set error breakpoints.	<code>dbstop</code>
Clear error breakpoints.	<code>dbc clear</code>
View a list of currently set breakpoints, including the condition and message identifier for each error breakpoint.	<code>dbstatus</code>

Evaluating Subsections of Files Using Code Cells

In this section...

“What Are Code Cells?” on page 9-175

“Scenarios for Evaluating Sections of Code” on page 9-176

“Process for Evaluating Sections of Files” on page 9-177

“Defining Code Cells” on page 9-178

“Understanding Nested Code Cells” on page 9-185

“Navigating Among Code Cells in a File” on page 9-193

“Evaluating Code Cells” on page 9-194

What Are Code Cells?

MATLAB files often have a natural structure consisting of multiple sections. Especially for larger files, you typically focus efforts on a single section at a time, working with the code in just that section. Similarly, when conveying information about your files to others, often you describe the sections of the code. To facilitate these processes, use *code cells*, where a *code cell* refers to a section of code. A code cell contains the contiguous lines of code that you want to evaluate as a whole in a MATLAB script. A code cell has boundaries to define its start and end. Because code cell features operate on code cells, it is important to understand how you define boundaries explicitly, how MATLAB defines boundaries implicitly, and how implicitly and explicitly defined code cell boundaries interact to create code cells, as described in “Defining Code Cells” on page 9-178

Specifically, MATLAB software uses code cells for:

- Evaluating sections of code in the Editor — This makes the experimental phase of your work with MATLAB scripts easier. This process is sometimes referred to as rapid code iteration. “Scenarios for Evaluating Sections of Code” on page 9-176 provides some situations where this process is useful. Additional sections provide an overview of the process and details for defining, evaluating, and modifying values in code cells.

- Publishing MATLAB files — This enables you to include code and results in a presentation format such as HTML. Publishing using code cells also requires you to define cells. You can use the cell navigation and evaluation you specify for running sections of code or define and use code cells explicitly for publishing. See Chapter 11, “Publishing MATLAB Code” for complete details.

Scenarios for Evaluating Sections of Code

When working with a MATLAB file, you often experiment with your code—modifying it, testing it, and updating it—until you have a file that does what you want. For example:

- Suppose you have code that plots data. You might break the code into two code cells: the code in the first cell creates the basic results, while the code in the second cell labels the plot. The two code cells enable you to experiment with the plot of the data first, and then when that is final, change the plot properties to affect the style of presentation. This scenario is presented in “Example of Evaluating Code Cells” on page 9-198.
- Suppose you have an two images that you want to add, and then display the results. As part of this algorithm, you want to adjust the brightness of the second image before adding it to the first image. You can read in those images, tweak the second image’s brightness, read it in again, adjust its brightness, and so on using the cell mode toolbar buttons until you see the brightness you want. There is no need to save the file between adjustments. If you have an active Internet connection, you can watch the Rapid Code Iteration Using Cells video demo that illustrates this example and more.

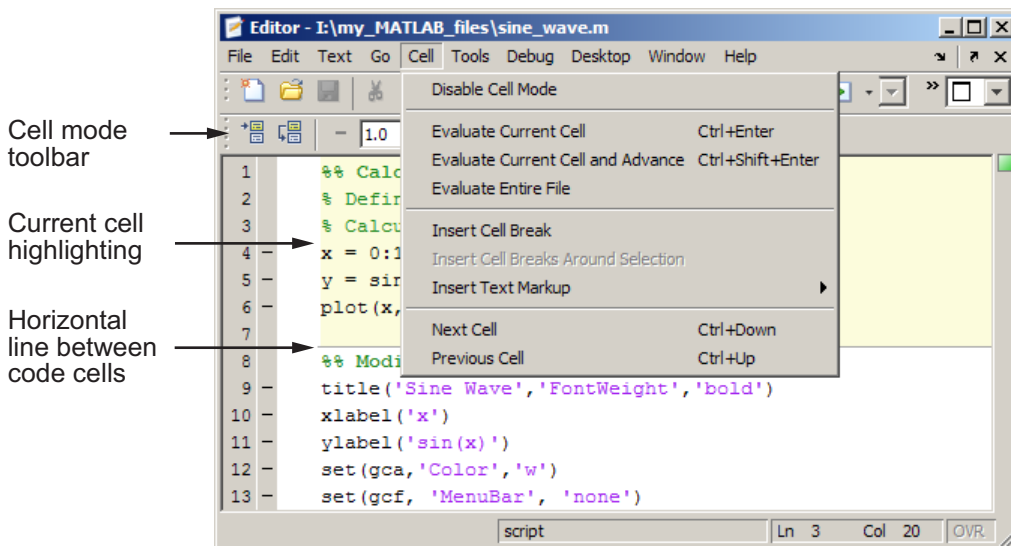
Use the MATLAB code cell features with MATLAB scripts to facilitate this process. You also can use code cell features with MATLAB function files, but there are some restrictions—see “Using Code Cells in MATLAB Function Files” on page 9-196.

Note Cell mode is supported for use with MATLAB code files (.m files) only. It is not for use with plain text files. When used with plain text files, results are unpredictable.

Process for Evaluating Sections of Files

This is the overall process of using code cells to evaluate sections of code:

- 1 In the MATLAB Editor, select **Cell > Enable Cell Mode**. Items in the **Cell** menu become selectable. The cell toolbar appears, unless you had previously hidden it. With cell mode enabled, hide or show the toolbar by right-clicking in the Editor menu bar or toolbars and selecting **Cell Toolbar** from the context menu.
- 2 Define the boundaries of the cells in a MATLAB script using cell features. Code cells are denoted by a specialized comment syntax, `%%`. For details, see “Defining Code Cells” on page 9-178.
- 3 After you define the code cells, use cell features. Cell features enable you 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 affect the result. For details, see “Evaluating Code Cells” on page 9-194.



Defining Code Cells

You define code cell boundaries explicitly by inserting a line that begins with a cell break (also referred to as a cell divider), which is two percent sign characters (%%). White space can precede these two characters, and text can follow them, if there is white space between the %% characters and the text. For details, see “Defining Code Cell Boundaries Explicitly” on page 9-179.

MATLAB defines implicit cell boundaries in a code block only when you specify one or more explicit cell breaks within that code block. MATLAB defines implicit cell breaks as follows:

- MATLAB defines implicit cell breaks at the top and bottom of the file, to create an implicit cell that contains the entire file. However, the Editor does not highlight the resulting cell, which encloses the entire file, unless you add one or more explicit cell breaks to the file.
- If you define an explicit cell break in a function, MATLAB defines implicit cell breaks at the function declaration and at the function end statement.

The resulting cells are nested within the full file cell. If you do not end the function with an explicit end statement, MATLAB behaves as though the end of the function occurs immediately before the start of the next function.

- If you define an explicit cell break within a language construct (such as an if or while statement), MATLAB defines implicit cell breaks at the lines containing the start and end of the language construct.

The resulting cells are nested within the full file cell, and the function in which the code block occurs, if any.

If an implicit cell break and an explicit cell break occur on the same line, they collapse into one explicit cell break. For more information on nested cells, see “Understanding Nested Code Cells” on page 9-185.

This section includes the following topics:

- “Defining Code Cell Boundaries Explicitly” on page 9-179
- “Creating Titles for Code Cells” on page 9-180
- “Highlighting Code Cells” on page 9-180
- “Example of Defining Code Cells” on page 9-181

- “Fixing Code Cell Highlighting Problems” on page 9-182
- “Removing Code Cells” on page 9-184
- “Summary of Cell Mode and Code Cell Requirements” on page 9-184

Defining Code Cell Boundaries Explicitly

To define code cell boundaries explicitly, insert cell breaks:

1 Ensure that cell mode is enabled. (See “Scenarios for Evaluating Sections of Code” on page 9-176.)

2 Optionally, to help you distinguish cells from each other, do one or both of the following:

- Include a faint gray horizontal line (rule) above each cell to help you distinguish the cells from each other.

Select **File > Preferences > Colors > Programming Tools**, and then in **Cell display options**, select **Show lines between cells**.


The horizontal lines do not appear in the file when you print it.

- Set a color to indicate the current cell.

Select **File > Preferences > Colors > Programming Tools**. Then, in **Cell display options**, select **Highlight cells**, and then select the color that you want. By default, MATLAB highlights the current cell in yellow.

The current cell is the cell where you have placed the cursor. Like the lines between cells, highlighting helps you distinguish the cells from each other.

3 Do one of the following to insert the cell breaks:

- Position the cursor just before the line at which you want to start the cell and select **Cell > Insert Cell Break**.
- Click the Insert cell break button .
- Enter two percent signs (%%) at the start of the line where you want to begin the new cell.
- Select the lines of code you want in a cell, and then select **Cell > Insert Cell Breaks Around Selection**.

Note Program control statements, such as `if ... end`, must be contained within a single cell. You cannot insert a cell break between the `if` and the end statements.

You can define a cell at the start of a new empty file, enter code for the cell, define the start of the next cell, enter its code, and so on. Redefine cells by defining new cells, removing existing cell boundaries, and moving lines of code.

Creating Titles for Code Cells

The Editor emphasizes the special meaning of the start of a cell by making any text following the percent signs appear bold. The text on the `%` line is called the *cell title*. Including text in cell titles is optional, however, it improves the readability of the file and is used for cell publishing features.

To create a cell title, after the `%` characters that specify a cell break, type a space, followed by a description of the cell.

Highlighting Code Cells

When the cursor is positioned in any line within a cell, the Editor highlights the entire cell that contains that line with a yellow background, by default. This identifies it as the *current cell*. The current cell is used, for example, when you select the **Evaluate Current Cell** option from the **Cell** menu.

Turning Off Code Cell Highlighting.

- 1 Select **File > Preferences > Colors > Programming Tools**.
- 2 Under **Cell display options**, clear **Highlight cells**.

Setting a Color for Code Cell Highlighting.

- 1 Select **File > Preferences > Colors > Programming Tools**.
- 2 Under **Cell display options**, select **Highlight cells**.
- 3 Click the down arrow next to the color block beside **Highlight cells**, and then select the color with which you want to highlight cells.

Example of Defining Code Cells

This example defines two cells for a file called `sine_wave.m`, shown in the figure that follows. To open this code in your Editor, run the following command and then save the file to a local folder:

```
edit(fullfile(matlabroot, 'help', 'techdoc', ...  
'matlab_env', 'examples', 'sine_wave.m'))
```

The steps that follow insert a cell break into the code to create two cells. The code in the first cell creates the basic results, while the second labels the plot. The two cells enable 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.

1 Select **Cell > Enable Cell Mode**.

When cell mode is enabled, the **Cell** menu displays **Disable Cell Mode**.

2 Select **File > Preferences > Colors > Programming Tools**, and then select **Highlight cells** and **Show lines between cells**.

3 Position the cursor at the start of the first line. Select **Cell > Insert Cell Break**.

The Editor inserts a cell break `%%` as the first line and moves the rest of the file down one line. All lines appear highlighted in yellow, indicating that the entire file is a single cell, assuming that you have that display preference for cells selected.

4 After the cell break, type a space, and then enter a cell title.

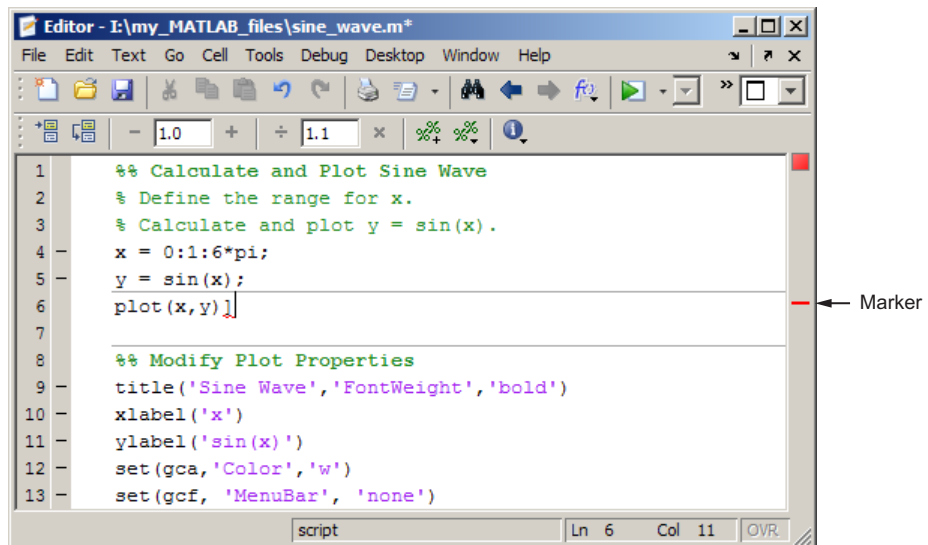
```
%% Calculate and Plot Sine Wave
```

5 Position the cursor at the start of line 7, `title...`, and then select **Cell > Insert Cell Break**.

The Editor inserts a line containing only a cell break (`%%`) at line 7 and moves the remaining lines down one line. A horizontal line that helps you distinguish the two cells appears above the cell break line. Lines 7 through 12 appear highlighted in yellow, indicating they comprise the current cell.

For example, suppose your code currently appears as specified in the “Example of Defining Code Cells” on page 9-181. Make sure that automatic code analysis is enabled by selecting **File > Preferences > Code Analyzer**, and then selecting **Enable integrated warning and error messages**.

If you accidentally insert a syntax error (a closing bracket at the end of line 6), then the error is evident by a message marker. The cell highlighting remains as is.



```
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
8 %% Modify Plot Properties
9 title('Sine Wave','FontWeight','bold')
10 xlabel('x')
11 ylabel('sin(x)')
12 set(gca,'Color','w')
13 set(gcf, 'MenuBar', 'none')
```

However, if you attempt to introduce a new cell at line 7, an extra cell divider line appears between lines 5 and 6.

```

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  %%
8  %% Modify Plot Properties
9  title('Sine Wave','FontWeight','bold')
10 xlabel('x')
11 ylabel('sin(x)')
12 set(gca,'Color','w')
13 set(gcf,'MenuBar','none')

```

To fix the problem, correct the syntax error. The extraneous cell divider disappears.

Removing Code Cells

To remove a code cell, do one of the following:

- Delete one of the percent signs (%) from the line that starts the cell.
This changes the line from a cell break to a standard comment.
- Delete the entire line that contains the %% characters.

In both cases, because you remove the cell break, MATLAB merges the two cells that were previously separated by the cell break.

Summary of Cell Mode and Code Cell Requirements

The following list summarizes facts to keep in mind when using cell mode and defining cells:

- Cell mode is supported for use with MATLAB code files (.m files) only. It is not for use with plain text files.

- MATLAB does not execute the code in lines beginning with cell break characters, %%.
- MATLAB considers the entire file to be a single cell; therefore, the first line in a file does not have to begin with %%.
- For program control statements, such as `if ... end`, a cell must contain both the opening and closing statements, that is, it must contain both the `if` and the `end` statements.
- You can set preferences for cell display options by selecting **Select File > Preferences > Colors > Programming Tools**, and then making choices for **Cell Display options**.
- If code analysis finds errors in a file, cell dividers and highlighting might not appear as you expect. For details, see “Fixing Code Cell Highlighting Problems” on page 9-182.

For more information, see “Preventing and Identifying Coding Problems” on page 9-107.

Understanding Nested Code Cells

You can insert cells within nested code, which results in nested cells. The following sections illustrate how inserting explicit cell breaks interacts with the implicit cell breaks that MATLAB inserts within a file:

- “File Without Explicit Code Cell Breaks” on page 9-185
- “How Nesting Code Cell Breaks Result in Cells” on page 9-186
- “Example File with Nested Code Cell Breaks” on page 9-187
- “Associating Code Cell Breaks with Subfunctions” on page 9-191

File Without Explicit Code Cell Breaks

The following code when viewed in the Editor displays no cells or highlighting. It is a single, implicit code cell, defined by MATLAB.

```
function fourier
    t = 0:.1:pi*4;
    y = sin(t);
    updatePlot(1,t,y);
```

```
        for k = 3:2:9
            y = y + sin(k*t)/k;
            display(sprintf('When k = %.1f',k));
        end
    end

    function updatePlot(k,t,x)
        cla
        plot(t,x)

    end
```

To follow this example, save the code to a local folder with the file name `fourier.m`.

How Nesting Code Cell Breaks Result in Cells

Suppose you insert two cell breaks into `fourier.m` as follows:

- 1** One within the `fourier` function, at line 5.
- 2** One within the `for` loop, at line 8

This results in the following cells, which are illustrated in “Example File with Nested Code Cell Breaks” on page 9-187:

- One cell at the outermost level, from the top to the bottom of the file.
- Two cells at the second level, within the `fourier` function:
 - One from the implicit break at line 2 to the explicit break at line 5.
 - One from the explicit break at line 5 to the implicit break before line 11 (end of the function).
- One cell at the third level, within the `for` loop from the explicit line break at line 7 to the implicit line break before line 10.

Example File with Nested Code Cell Breaks

The following images illustrate how inserting explicit cell breaks, as described in “How Nesting Code Cell Breaks Result in Cells” on page 9-186, affect the appearance of the file:

- **First level of nesting** — When you place the cursor outside a function, at the outermost level, the entire file appears highlighted, showing that it comprises a cell at this level of nesting.

Cursor at outermost level

```

1  function fourier
2  -
3  t = 0:.1:pi*4;
4  y = sin(t);
5  updatePlot(1,t,y);
6  %%
7  for k = 3:2:9
8  -
9  %%
10 y = y + sin(k*t)/k;
11 display(sprintf('When k = %.1f',k));
12 end
13 end
14 function updatePlot(k,t,x)
15 -
16 cla
17 plot(t,x)
18 end

```

MATLAB only defines implicit cell breaks in a code block if you specify an explicit cell break within that code block. Therefore, because function `updatePlot` in this example has no explicit (and therefore, no implicit) cell breaks defined for it, when you place the cursor within that function, MATLAB considers the cursor to be within the cell that encloses the whole file.

```

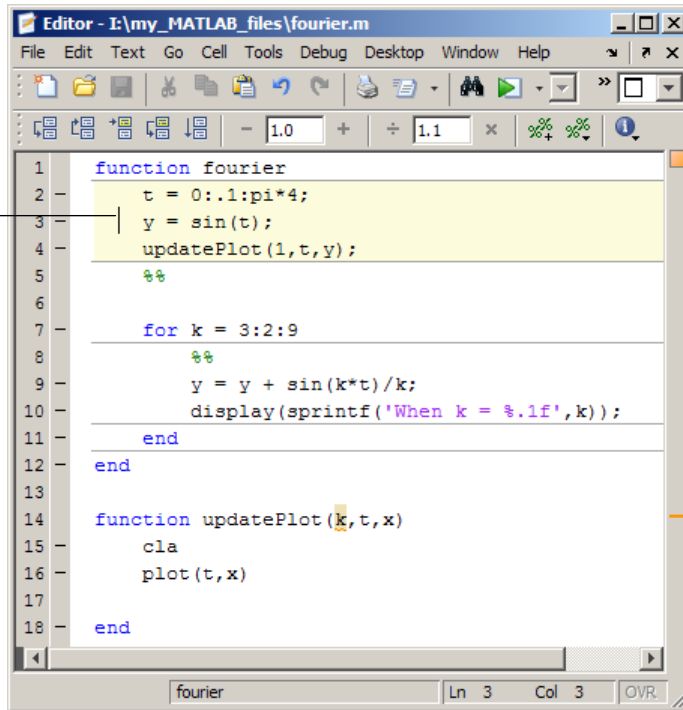
1  function fourier
2      t = 0:.1:pi*4;
3      y = sin(t);
4      updatePlot(1,t,y);
5      %%
6
7      for k = 3:2:9
8          %%
9          y = y + sin(k*t)/k;
10         display(sprintf('When k = %.1f',k));
11     end
12 end
13
14 function updatePlot(k,t,x)
15     cla
16     plot(t,x)
17
18 end

```

Cursor in function with no explicit, and therefore no implicit cells defined

- **Second level of nesting** — When you place the cursor within the function (but outside the for loop), either the first or second cell at this level of nesting appears highlighted, depending on where the cursor is located.

First cell
within function



The image shows a MATLAB Editor window titled "Editor - I:\my_MATLAB_files\fourier.m". The window contains a MATLAB script with the following code:

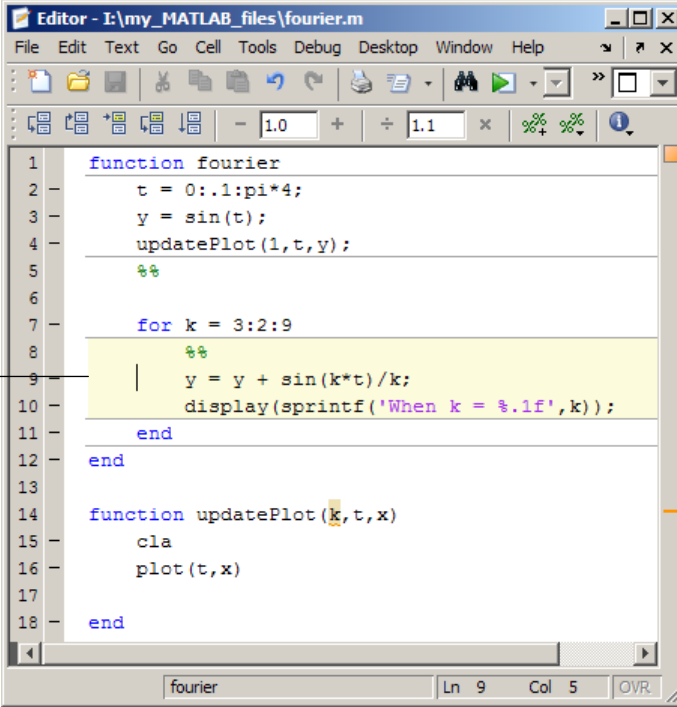
```
1 function fourier
2     t = 0:.1:pi*4;
3     y = sin(t);
4     updatePlot(1,t,y);
5     %%
6
7     for k = 3:2:9
8         %%
9         y = y + sin(k*t)/k;
10        display(sprintf('When k = %.1f',k));
11    end
12 end
13
14 function updatePlot(k,t,x)
15     cla
16     plot(t,x)
17
18 end
```

The code is divided into several sections by horizontal lines, representing code cells. The first cell, containing lines 2 through 4, is highlighted in yellow. A text label "First cell within function" with a pointer indicates this cell. The status bar at the bottom shows "fourier", "Ln 3", "Col 3", and "OVR".

Second cell
within function

```
1 function fourier
2     t = 0:.1:pi*4;
3     y = sin(t);
4     updatePlot(1,t,y);
5     %%
6
7     for k = 3:2:9
8         %%
9         y = y + sin(k*t)/k;
10        display(sprintf('When k = %.1f',k));
11    end
12 end
13
14 function updatePlot(k,t,x)
15     cla
16     plot(t,x)
17
18 end
```

- **Third level of nesting** — When you place the cursor within the for loop, the cell within this loop is highlighted.

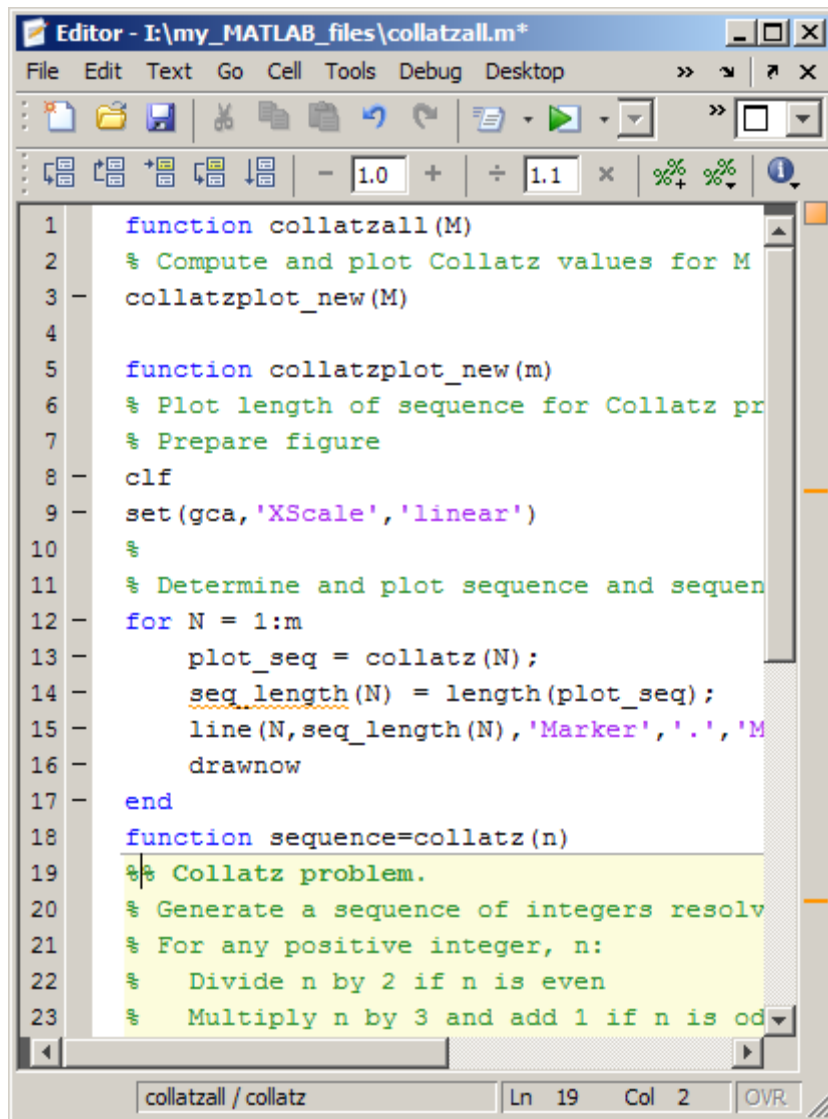


```
1 function fourier
2     t = 0:.1:pi*4;
3     y = sin(t);
4     updatePlot(1,t,y);
5     %%
6
7     for k = 3:2:9
8         %%
9         | y = y + sin(k*t)/k;
10        display(sprintf('When k = %.1f',k));
11    end
12 end
13
14 function updatePlot(k,t,x)
15     cla
16     plot(t,x)
17
18 end
```

Cell within for loop

Associating Code Cell Breaks with Subfunctions

If you want a cell break to be associated with a subfunction, place the cell break within the subfunction, rather than above the subfunction declaration. Otherwise, the cell break creates a single cell within the code block that precedes the subfunction. The following two images demonstrate this using `collatzall.m`.



```
1 function collatzall(M)
2 % Compute and plot Collatz values for M
3 collatzplot_new(M)
4
5 function collatzplot_new(m)
6 % Plot length of sequence for Collatz pr
7 % Prepare figure
8 clf
9 set(gca, 'XScale', 'linear')
10 %
11 % Determine and plot sequence and sequen
12 for N = 1:m
13     plot_seq = collatz(N);
14     seq_length(N) = length(plot_seq);
15     line(N, seq_length(N), 'Marker', '.', 'M
16     drawnow
17 end
18 function sequence=collatz(n)
19 %% Collatz problem.
20 % Generate a sequence of integers resolv
21 % For any positive integer, n:
22 %   Divide n by 2 if n is even
23 %   Multiply n by 3 and add 1 if n is od
```

```




1  function collatzall(M)
2  % Compute and plot Collatz values for M
3  collatzplot_new(M)
4
5  function collatzplot_new(m)
6  % Plot length of sequence for Collatz pr
7  % Prepare figure
8  clf
9  set(gca,'XScale','linear')
10 %
11 % Determine and plot sequence and sequen
12 for N = 1:m
13     plot_seq = collatz(N);
14     seq_length(N) = length(plot_seq);
15     line(N,seq_length(N),'Marker','.', 'M
16     drawnow
17 end
18 %% Collatz Problem
19 function sequence=collatz(n)
20 % Collatz problem.
21 % Generate a sequence of integers resolv
22 % For any positive integer, n:
23 %   Divide n by 2 if n is even

```

Navigating Among Code Cells in a File

You can navigate among cells in a file without evaluating the code within those cells, as described in the table that follows. This can be useful when

you want to jump quickly from cell to cell within a file. You might do this, for example, to find the point at which you want to begin cell evaluation. By default, the cell mode navigation buttons are not on the Editor Cell Mode toolbar. For information on how to add them, see “Setting Toolbars Preferences for Desktop Tools” on page 2-156 .

Operation	Instructions
Move to the next cell.	Select Cell > Next Cell or click the Next Cell button  .
Move to the previous cell.	Select Cell > Previous Cell or click the Previous Cell button  .
Move to a specific cell.	Do either of the following: <ul style="list-style-type: none"> Use the Editor Cell Mode toolbar: <ol style="list-style-type: none"> Click the Show cell titles button . Select the cell title to which you want to move. Use the Go menu: <ol style="list-style-type: none"> Select Go > Go To. The Go To dialog box opens. Select Function or cell title. Select the cell title to which you want to move. Click OK.

Evaluating Code Cells

As you develop a MATLAB file, you can use the Editor cell features to evaluate the file cell-by-cell. This method helps you to experiment with, debug, and fine-tune your code. You can navigate from cell to cell, and evaluate each cell individually. See the following topics for details:

- “Evaluating Code Cells in a File” on page 9-195
- “Processing Considerations When Evaluating Code Cells” on page 9-196




- “Modifying Values in a Code Cell” on page 9-197
- “Example of Evaluating Code Cells” on page 9-198

Evaluating Code Cells in a File

The cell evaluation features run the cell code currently shown in the Editor, even if the file contains unsaved changes. The file does not have to be on the search path. To evaluate a cell, it must contain all the values it requires, or the values must exist in the MATLAB workspace.

To run the code in a cell, use the **Cell** menu evaluation items or equivalent buttons in the cell mode toolbar. When you evaluate a cell, the results display in the Command Window, figure window, or elsewhere, depending on the code evaluated.

The following table provides instructions on evaluating code cells.

Operation	Instructions
Run the code in the current cell.	Select Cell > Evaluate Current Cell or click the Evaluate cell button  .
Run the code in the current cell, and then move to the next cell.	Select Cell > Evaluate Current Cell and Advance or click the Evaluate cell and advance button  .
Run all the code in the file.	<p>Select Cell > Evaluate Entire File or click the Evaluate entire file button . By default, the Evaluate entire file button is not on the Editor Cell Mode toolbar. See “Setting Toolbars Preferences for Desktop Tools” on page 2-156 for information on how to add it.</p> <hr/> <p>Note A beep indicates there is an error. See the Command Window for the error message.</p> <hr/>

Processing Considerations When Evaluating Code Cells

This section describes processing considerations to take into account when you evaluate code cells in MATLAB files.

Setting Breakpoints. While you can set breakpoints and debug a file containing cells, when you evaluate a file from the **Cell** menu or cell toolbar, breakpoints are ignored. To run the file and stop at breakpoints, use **Run/Continue** in the **Debug** menu. This means you cannot debug while running a single cell.

Using Code Cells in MATLAB Function Files. You can define and evaluate cells in MATLAB function files as long as the variables referenced in the code cell are in your workspace. This can be useful during debugging. If execution is stopped at a breakpoint, you can define cells and execute them without saving the file. If you are not debugging, add the necessary variables to the base workspace, and then execute the cells.

Using Function Names as Variable Names in Code Cells. If you use a MATLAB function name as a variable name within a cell, you might receive an unexpected error when you evaluate the cell. The precedence rules that MATLAB typically follows do not apply when it evaluates a cell. Typically, MATLAB evaluates variables before functions. However, when you evaluate cells, MATLAB parses all the cell code and loads it into memory before evaluating it. Therefore, functions might be evaluated before variables under some circumstances, as illustrated by the following example.

Suppose you create a MAT-file, `mydata.mat`, using the following commands:

```
clear all
info=5;
save mydata.mat
clear all
```

When you enter the following commands in the Command Window, `b` evaluates to `5`, as expected:

```
load mydata
b=info
```

However, when you evaluate the same commands in a code cell, `b` evaluates to the MATLAB `info` function, thus the Command Window displays the following error:

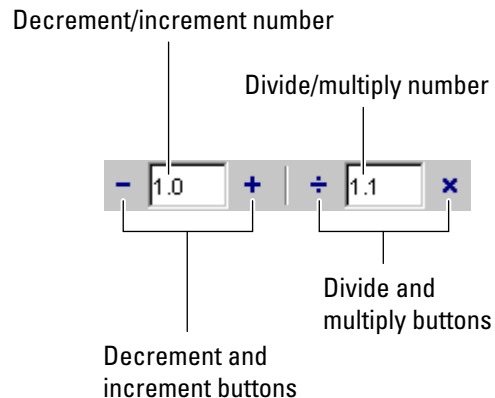
```
??? Error using ==> info
Too many output arguments.
```

For this reason, consider avoiding using function names as variable names within code cells.

Modifying Values in a Code Cell

You can use code cell features to modify numbers in a cell, which also automatically reevaluates the cell. This helps you experiment with and fine-tunes your code.

To modify a number in a cell, select the number (or place the cursor near it) and use the value modification tool in the cell toolbar. Using this tool, you can specify a number and press the appropriate math operator to add (increment), subtract (decrement), multiply, or divide the number. The cell then automatically reevaluates.



You can use the numeric keypad operator keys (`-`, `+`, `/`, and `*`) instead of the operator buttons on the toolbar.

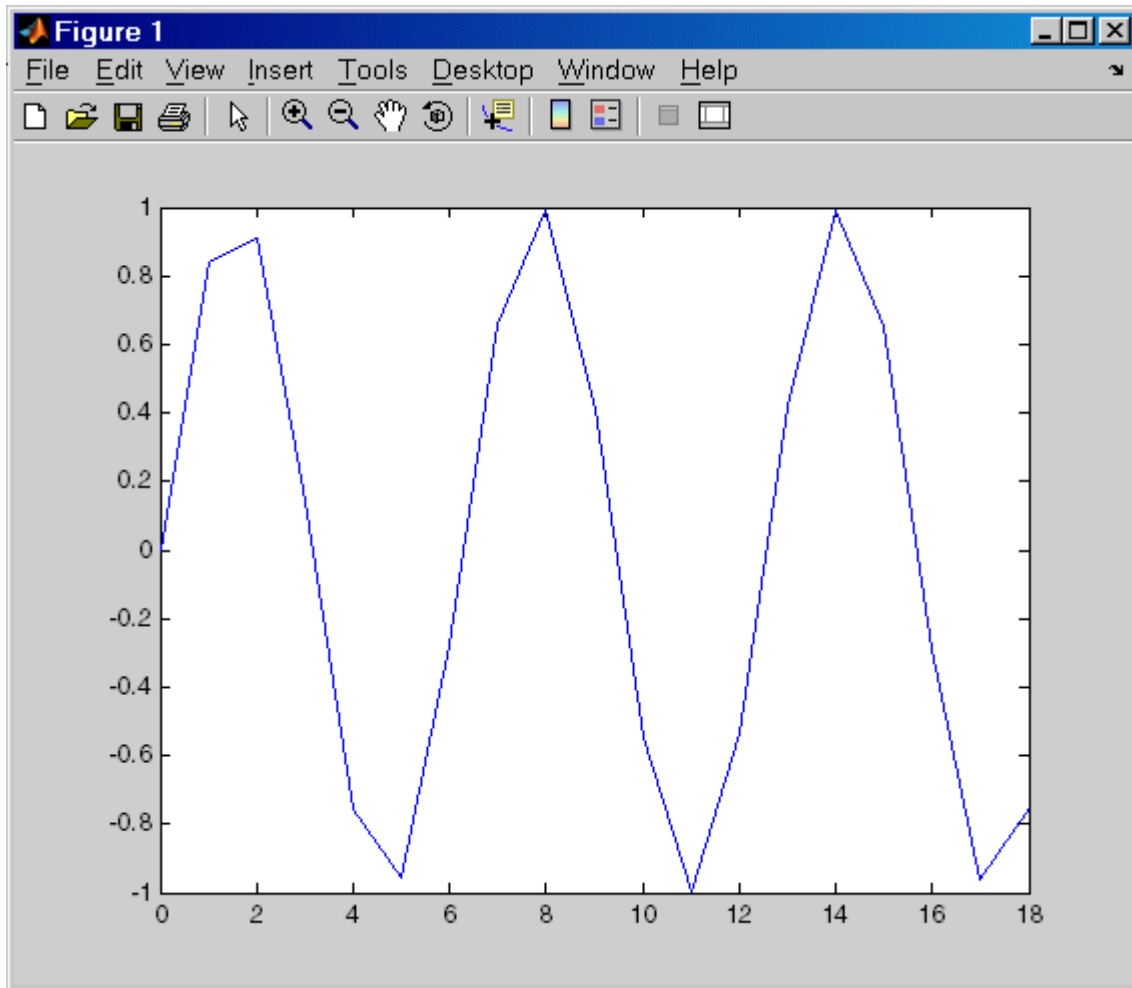
Note MATLAB software does not automatically save changes you make to values using the cell toolbar. To save changes, select **File > Save**.

Example of Evaluating Code Cells

In this example, modify the values for `x` in `sine_wave.m`:

- 1 Run the first cell in `sine_wav.m`. Click somewhere in the first cell, that is, between lines 1 and 6. Select **Cell > Evaluate Current Cell**. The following figure appears.

Plot generated by running `sine_wave.m`.



- 2 Assume that you want to produce a smoother curve. Use more values for x in `0:1:6*pi`. Position the cursor in line 4, next to the 1. In the cell toolbar, change the 1.1 default multiply/divide by value to 2. Click the Divide button \div .

Line 4 becomes

```
4 - x = 0:0.5:6*pi;
```

and the length of x doubles. The plot automatically updates. The curve still has some rough edges.

- 3 To add more values for x , click the Divide button three more times. Line 4 becomes

```
4 - x = 0:0.0625:6*pi;
```

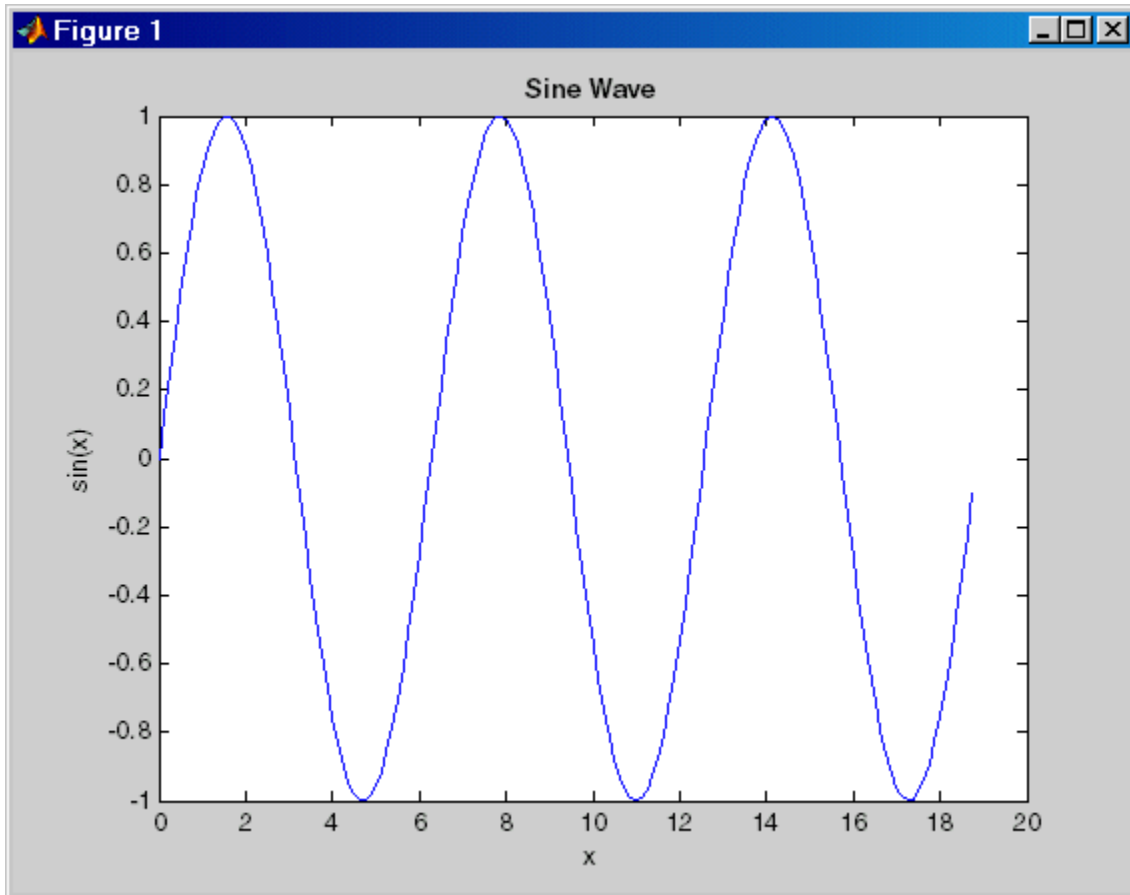
The curve is smooth, but because there are more values, processing time is slower. It would be better to find a smaller x that still produces a smooth curve.

- 4 In the cell toolbar, click the Multiply button once. The increment for x as shown in line 4 changes from 0.0625 to 0.125.

The resulting curve is still smooth.

- 5 Save these changes. Select **File > Save**.
- 6 Now you can apply the plot properties, defined in the second cell, that is, lines 7 through 12. You do not need to evaluate the entire file to apply the plot properties. Instead, position the cursor in the second cell and select **Cell > Evaluate Current Cell** to evaluate the current cell.

MATLAB updates the figure.



Debugging Functions

- `dbstop`—Set breakpoints
- `dbclear`—Clear breakpoints
- `dbcont`—Resume execution
- `dbdown`—Reverse workspace shift performed by `dbup`, while in debug mode
- `dbmex`—Enable MEX-file debugging (on UNIX platforms)
- `dbstack`—Function call stack
- `dbstatus`—List breakpoints
- `dbstep`—Execute one or more lines from current breakpoint
- `dbtype`—List text file with line numbers
- `dbup`—Shift current workspace to workspace of caller, while in debug mode
- `dbquit`—Quit debug mode

Tuning and Managing MATLAB Code Files

This set of tools provides useful information about the MATLAB code files in a folder that can help you refine the files and improve performance. The tools can help you polish these files before providing them to others to use.

- “Using MATLAB Reports” on page 10-2
- “Using the Code Analyzer Report” on page 10-22
- “Profiling for Improving Performance” on page 10-27

Using MATLAB Reports

In this section...

“Refining and Improving Files Using Reports” on page 10-2

“Identifying Files with Reminder Annotations” on page 10-4

“Generating a Summary View of the Help Components in Functions and Scripts” on page 10-8

“Displaying and Updating a Report on the Contents of a Folder” on page 10-11

“Displaying Dependencies Among MATLAB Code Files” on page 10-15

“Identifying How Much of a File Ran When Profiled” on page 10-20

See also “Using the Code Analyzer Report” on page 10-22 the Comparison Tool.


Refining and Improving Files Using Reports

Reports help you refine MATLAB code files within a given folder and improve their performance. They are also useful for checking the quality of files before you distribute them for use by others, to share on MATLAB Central, or for a toolbox. (A toolbox is a collection of files for use with MATLAB and related products.)

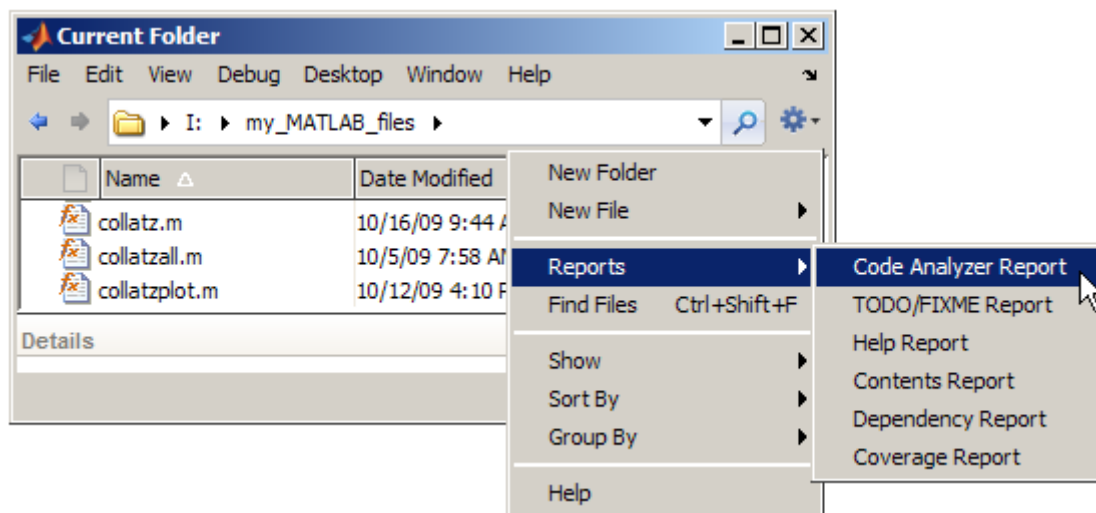
See also “Using the Code Analyzer Report” on page 10-22.

Accessing Reports

To access reports from the MATLAB Current Folder browser:

- 1 Select **Desktop > Current Folder** or click the Current Folder browser.
- 2 Navigate to the folder containing the files for which you want to produce reports.
- 3 On the Current Folder browser toolbar, click the **Actions** button , and then select the type of report you want to run.

The report runs for all the MATLAB code files in the current folder.



The report you select appears as an HTML document in the MATLAB Web Browser.

Note You cannot run reports when the path is a UNC (Universal Naming Convention) path, that is, starts with \\ . Instead, use an actual hard drive on your system, or a mapped network drive.

Using Reports

All MATLAB reports contain various links that enable you to access additional information, as described in the table that follows:

To	Do this
Open a file in the Editor to view it or modify it.	Click a file name in the report.
Open a file at the line listed in a report.	Click the line number.
Update a report after modifying report options.	Click Rerun This Report .

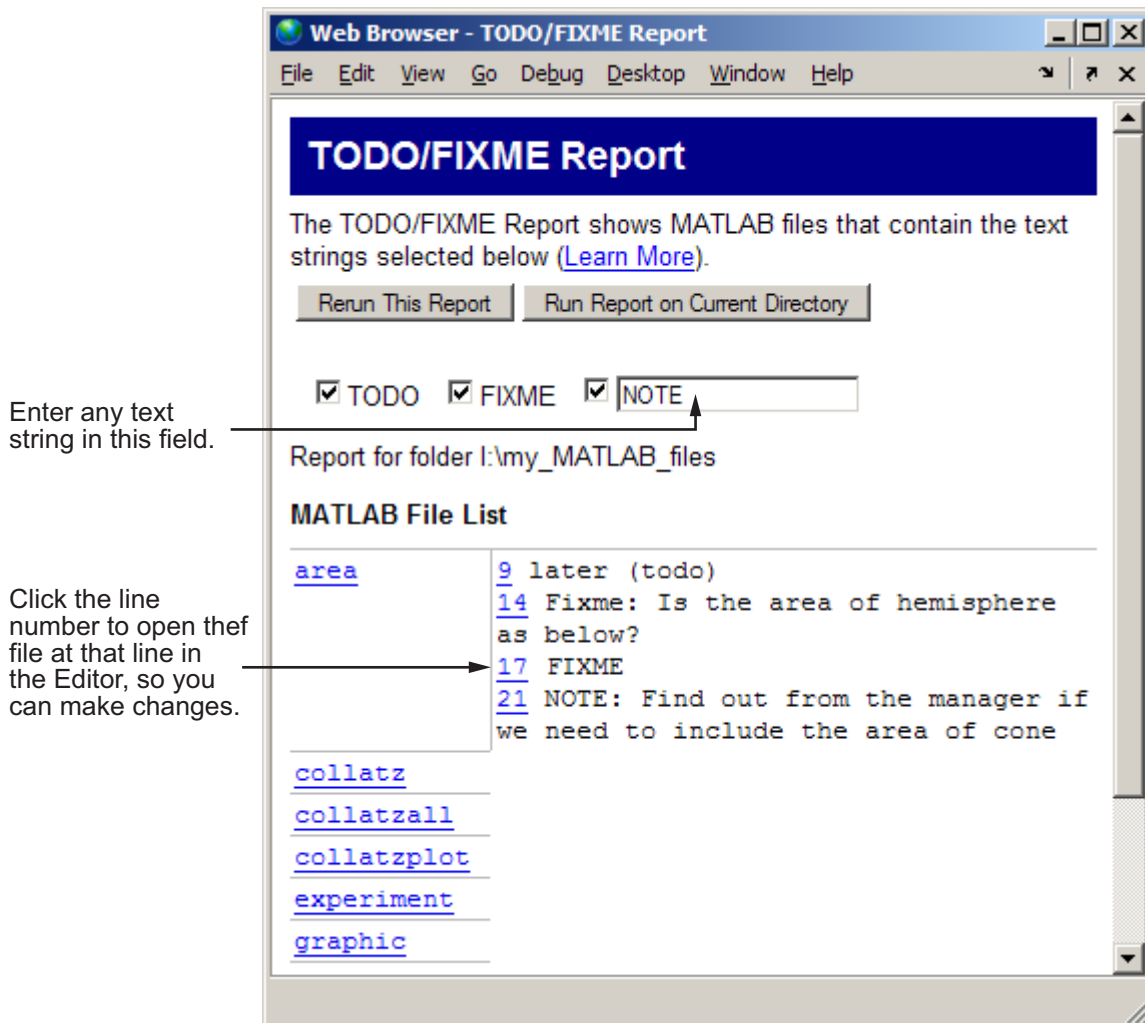
To	Do this
Update a report after changing any files in the folder.	Click Rerun This Report .
Generate the same type of report for a different folder.	<ol style="list-style-type: none">1 Keep the current report open.2 Change the MATLAB current folder.3 Click Run Report on Current Folder.

Note Clicking **Rerun This Report** reruns the report for the folder shown in the report, not for the MATLAB current folder.

Identifying Files with Reminder Annotations

The TODO/FIXME Report identifies all the MATLAB code files within a given folder that you have annotated. You annotate a file by adding comments with the text TODO, FIXME, or a string of your choosing. Annotating a file makes it easier to find areas of your code that you intend to improve, complete, or update later. The TODO/FIXME Report presents a list of files containing the annotations in a Web browser.

This sample TODO/FIXME Report shows files containing the strings TODO, FIXME, and NOTE. The search is case insensitive.



Working with TODO/FIXME Reports

- 1 Select **Desktop > Current Folder** and navigate to the folder containing the files for which you want to produce a TODO/FIXME report.

- 2 On the Current Folder browser toolbar, click the **Actions** button , and then select **Reports > TODO/FIXME Report**.

The TODO/FIXME Report opens in the MATLAB Web Browser.

- 3 In the TODO/FIXME Report window, select one or more of the following to specify the lines that you want the report to include:
 - TODO
 - FIXME
 - The text field check box

You can then enter any text string in this field, including a regular expression. For example, you can enter NOTE, tbd, or re.*check.

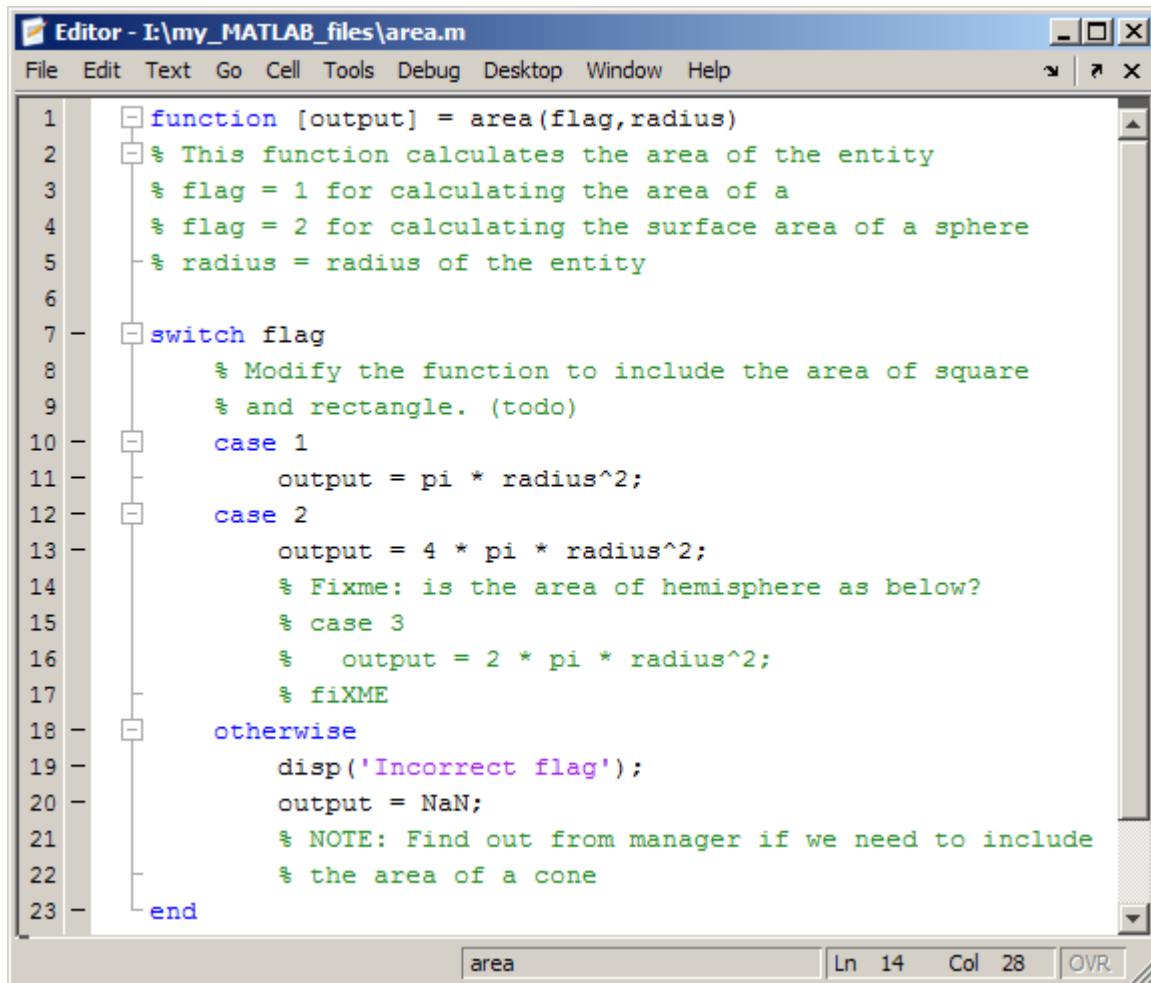
- 4 Run the report on the files in the current folder, by clicking **Rerun This Report**.

The window refreshes and lists all lines in the MATLAB files within the specified folder that contain the strings you selected in step 1. Matches are not case sensitive.

If you want to run the report on a folder other than the one currently specified in the report window, change the current folder. Then, click **Run Report on Current Folder**.

To open a file in the Editor at a specific line, click the line number in the report. Then you can change the file, as needed.

Suppose you have a file, `area.m`, in the current folder. The code for `area.m` appears in the image that follows.



```
Editor - I:\my_MATLAB_files\area.m
File Edit Text Go Cell Tools Debug Desktop Window Help
1 function [output] = area(flag,radius)
2 % This function calculates the area of the entity
3 % flag = 1 for calculating the area of a
4 % flag = 2 for calculating the surface area of a sphere
5 % radius = radius of the entity
6
7 switch flag
8     % Modify the function to include the area of square
9     % and rectangle. (todo)
10 case 1
11     output = pi * radius^2;
12 case 2
13     output = 4 * pi * radius^2;
14     % Fixme: is the area of hemisphere as below?
15     % case 3
16     % output = 2 * pi * radius^2;
17     % fixME
18 otherwise
19     disp('Incorrect flag');
20     output = NaN;
21     % NOTE: Find out from manager if we need to include
22     % the area of a cone
23 end
area Ln 14 Col 28 OVR
```

When you run the TODO/FIXME report on the folder containing `area.m`, with the TODO and FIXME strings selected and the string NOTE specified and selected, the report lists:

9 and rectangle. (todo)

14 Fixme: Is the area of hemisphere as below?

17 fixME

21 NOTE: Find out from the manager if we need to include

<u>area</u>	<u>9</u> and rectangle. (todo)
	<u>14</u> Fixme: Is the area of hemisphere as below?
	<u>17</u> fixme
	<u>21</u> NOTE: Find out from the manager if we need to include

Notice the report includes the following:


- Line 9 as a match for the TODO string. The report includes lines that have a selected string regardless of its placement within a comment.
- Lines 14 and 17 as a match for the FIXME string. The report matches selected strings in the file regardless of their casing.
- Line 21 as a match for the NOTE string. The report includes lines that have a string specified in the text field, assuming that you select the text field.

Generating a Summary View of the Help Components in Functions and Scripts

A Help Report presents a summary view of the help component of your MATLAB files. Use this information to assist you in identifying files of interest or files that lack a help component. It is a good practice to provide help for your files not only to assist you in recalling their purpose, but to assist others who use the files.

In MATLAB, the *help component* is all contiguous nonexecutable lines (comment lines and blank lines), starting with the second line of a function file or the first line of a script file. For more information about creating help for your files, see the reference page for the help function.

Working with Help Reports

- 1 Select **Desktop > Current Folder** and navigate to the folder containing the MATLAB files for which you want to produce a Help Report.
- 2 On the Current Folder browser toolbar, click the **Actions** button , and then select **Reports > Help Report**.

The Help report opens in the MATLAB Web Browser.

3 Select one or more options, described in the following list, to have the Help Report display the specified help information:

- **Show class methods** to have the Help Report display help information for class methods created using the `classdef` keyword as well as functions.
- **Description** to have the Help Report display the first line of help in the file. If the first comment line is empty, or if there is not a comment before the executable code, then **No description line**, highlighted in pink, appears instead.
- **Examples** have the Help Report display the line number where the examples section of the help begins. The Help Report looks for a line in the help that begins with the string `example` or `Example` and displays any subsequent nonblank comment lines. Select this option to easily locate and go to examples in your files.

It is a good practice to include examples in the help for your MATLAB code files. If you do not have examples in the help for all program files, use this option to identify those without examples. If the report does not find examples in the MATLAB code file help, **No example**, highlighted in pink, appears.

- **Show all help** have the Help Report display complete MATLAB code file help, which is all contiguous nonexecutable lines (comment lines and blank lines), starting with the second line of a function file, or the first line of a script file. The help displayed also includes overloaded functions and methods, which are not actually part of the help comments, but are automatically generated when `help` runs. Description lines are reported twice if you also have also selected **Description**.

If the comment lines before the executable code are empty, or if there are no comments before the executable code, **No help**, highlighted in pink, appears instead.

- **See Also** have the Help Report display the line number for the `see also` line in the help. The `see also` line in help lists related functions. When the MATLAB Command Window displays the help for a MATLAB code 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 files.

The report looks for a line in the help that begins with the string `See also`. If the report does not find a `see also` line in the help, **No see-also line**, highlighted in pink, appears. This helps you identify those files without a `see also` line, should you want to include one in each MATLAB code file.

The report also indicates when a file noted in the `See also` line is not in a folder on the search path. You might want to move that file to a folder 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 folder to the path or make its folder become the current folder.

- **Copyright** have the Help Report display the line number for the copyright line in the file. The report looks for a comment line in the 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).

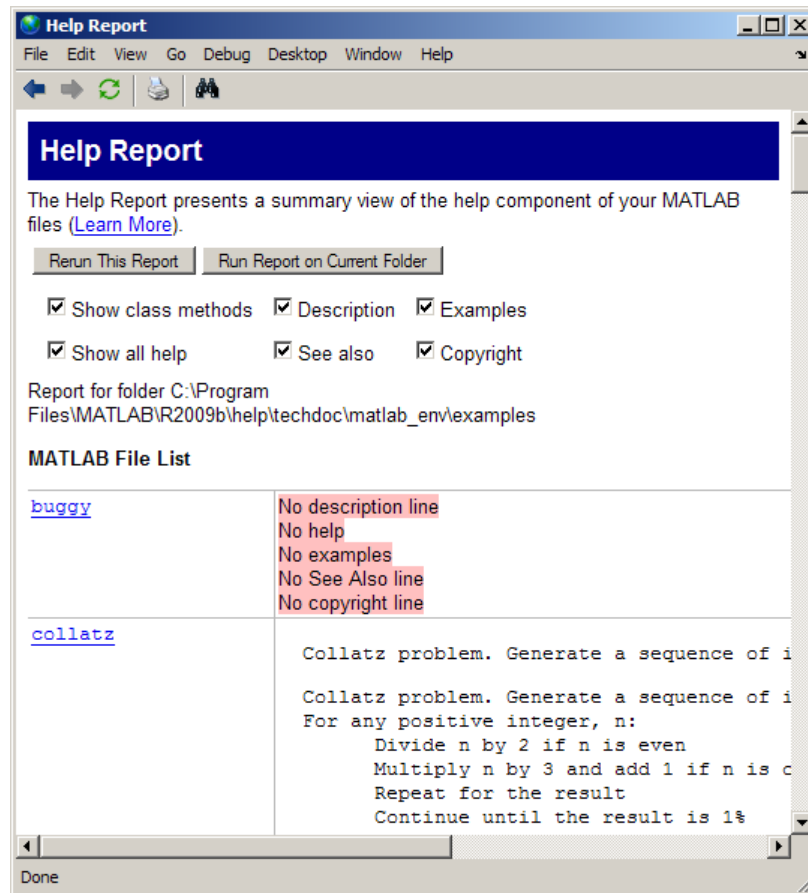
It is a good practice to include a copyright line in the help that notes the year you created the file and the current year. For example, for a file you created in 2001, include this line

```
% Copyright 2001-2008
```

If the report:

- Does not find a copyright line in the help, **No copyright line**, highlighted in pink, appears.
- Finds that the end of the date range is not the current year, **Copyright year is not current**, highlighted in pink, appears.

4 Click **Rerun This Report**. Your report resembles the following image.




Displaying and Updating a Report on the Contents of a Folder

The Contents Report displays information about the integrity of the Contents.m file for a given folder. A Contents.m file includes the file name and a brief description of each MATLAB code file in the folder. The Contents Report helps you to maintain the Contents.m file. It displays discrepancies between the Contents.m file and the MATLAB code files in the folder.

When you type `help` followed by the folder name, such as `help mydemos`, The MATLAB Command window displays the information contained within the

mydemos/Contents.m file. For more information, see “Providing Help for Your Program” in the MATLAB Programming documentation.

Working with Contents Reports

- 1 Select **Desktop > Current Folder** and navigate to the folder containing the files for which you want to produce a Contents report.
- 2 On the Current Folder browser toolbar, click the **Actions** button , and then select **Reports > Contents Report**.

The Contents Report opens in the MATLAB Web browser. If there is no Contents.m file for the folder, 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. You can edit the Contents.m file in the Editor to include the names of files you plan to create, or to remove entries for files that you do not want to expose when displaying help for the folder, such as code intended for internal use only.

- 3 Update the Contents.m file to reflect changes you make to files in the folder. For example, when you remove a file from a folder, remove its entry from the Contents.m file.

Choose from the following options to updating the contents:

- **edit Contents.m**— Opens the Contents.m file in the Editor.
- **fix spacing**—Automatically align the file names and descriptions in the Contents.m file.
- **fix all** makes all of the suggested changes at once.
- To make changes on a case-by-case basis, read each question in the Contents Report, and then click **yes** if you want to make the suggested change.

Contents File Report

The Contents Report displays information about the integrity of the Contents.m file for the folder ([Learn More](#)).

[Rerun This Report](#) [Run Report on Current Folder](#)

[[edit Contents.m](#) | [fix spacing](#) | [fix all](#)]

Report for folder I:\MATLAB

MATLAB

Files

[function file](#) - Summary of this function
[publish latex](#) - LaTeX Equations:

Description lines do not match for file [publish latex](#).
 Use this description from the **file**? (default) [[yes](#)]

`publish_latex - $$ A = B - C $$`

Or put this description from the **Contents** into the file? [[yes](#)]

`publish_latex - LaTeX Equations:`

`startup test -`
`stuff - Summary of this function`

`Untitled1 -`

File Untitled1 does not appear in this folder.
 Remove it from Contents.m? [[yes](#)]

File [a1](#) is in the folder but not Contents.m

Messages in the Contents File Report

No Contents File. This message appears if there is no `Contents.m` file in the folder. Click **yes** to automatically create a `Contents.m` file, which contains the file names and descriptions for all MATLAB code files in the folder.

```
No Contents.m file. Make one? [ yes ]
```

File Not Found. This message appears when a file included in `Contents.m` is not in the folder. These messages are highlighted in pink. For example, a message such as

```
File helloworld does not appear in this folder.  
Remove it from Contents.m? [ yes ]
```

means the `Contents.m` file includes an entry for `helloworld`, but that file is not in the folder. This might be because:

- You removed the file `helloworld`.
- You manually added `helloworld` to `Contents.m` because you planned to create the file, but have not as yet.
- You renamed `helloworld`.

Description Lines Do Not Match. This message appears when the description line in the file's help does not match the description provided for the file in `Contents.m`. These messages are highlighted in pink. Click **yes** to replace the description in the `Contents.m` file with the description from the file. Or select the option to replace the description line in the help using the description for that file in `Contents.m`.

```
Description lines do not match for file logo5.  
Use this description from the file? (default) [ yes ]  
  logo5      - This is the basic logo image for MATLAB 7  
Or put this description from the Contents into the file? [ yes ]  
  logo5 - This is the basic logo image for MATLAB
```

Files Not In Contents.m. This message appears when a file in the folder is not in `Contents.m`. These messages are highlighted in gray. Click **yes** to add the file name and its description line from the file help to the `Contents.m` file.

```
collatzall is in the folder but not Contents.m
collatzall - Plot length of sequence for Collatz problem
Add the line shown above? [ yes ]
```

Creating a New Contents.m File to Reflect All Files in the Current Folder

If you always want the Contents.m file to reflect all files in the current folder, you can automatically generate a new Contents.m file rather than changing the file based on the Contents Report, as follows:

- 1 Delete the existing Contents.m file.
- 2 Run the Contents Report.
- 3 Click **yes** when prompted for MATLAB to automatically create a Contents Report.

Displaying Dependencies Among MATLAB Code Files

The Dependency Report shows dependencies among MATLAB code files in a folder. Use this report to determine:

- Which files in the folder are required by other files in the folder
- If any files in the current folder will fail if you delete a file
- If any called files are missing from the current folder

The report does not list:

- Files in the `toolbox/matlab` folder because every MATLAB user has those files.

Therefore, if you use a function file that shadows a built-in function file, MATLAB excludes both files from the list.

- Files called from anonymous functions.
- The superclass for a class file.
- Files called from `eval`, `evalc`, `run`, `load`, function handles, and callbacks.

MATLAB does not resolve these files until run time, and therefore the Dependency Report cannot discover them.

- Some method files.


The Dependency Report finds class constructors that you call in a MATLAB file. However, any methods you execute on the resulting object are unknown to the report. These methods can exist in the `classdef` file, as separate method files, or files belonging to superclass or superclasses of a method file.

To provide meaningful results, the Dependency Report requires the following:

- The search path when you run the report is the same as when you run the files in the folder. (That is, the current folder is at the top of the search path.)
- The files in the folder for which you are running the report do not change the search path or otherwise manipulate it.
- The files in the folder do not load variables, or otherwise create name clashes that result in different program elements with the same name.

Note Do not use the Dependency Report to determine which MATLAB code files someone else needs to run a particular file. Instead use the `depfun` function.

Creating Dependency Reports

- 1** Select **Desktop > Current Folder** and navigate to the folder containing the files for which you want to produce a Dependency Report.
- 2** On the Current Folder browser toolbar, click the **Actions** button , and then select **Reports > Dependency Report**.

The Dependency Report opens in the MATLAB Web Browser.

- 3** If you want, select one or more options within the report, as follows:

- To see a list of all MATLAB code files (children) called by each file in the folder (parent), select **Show child functions**.

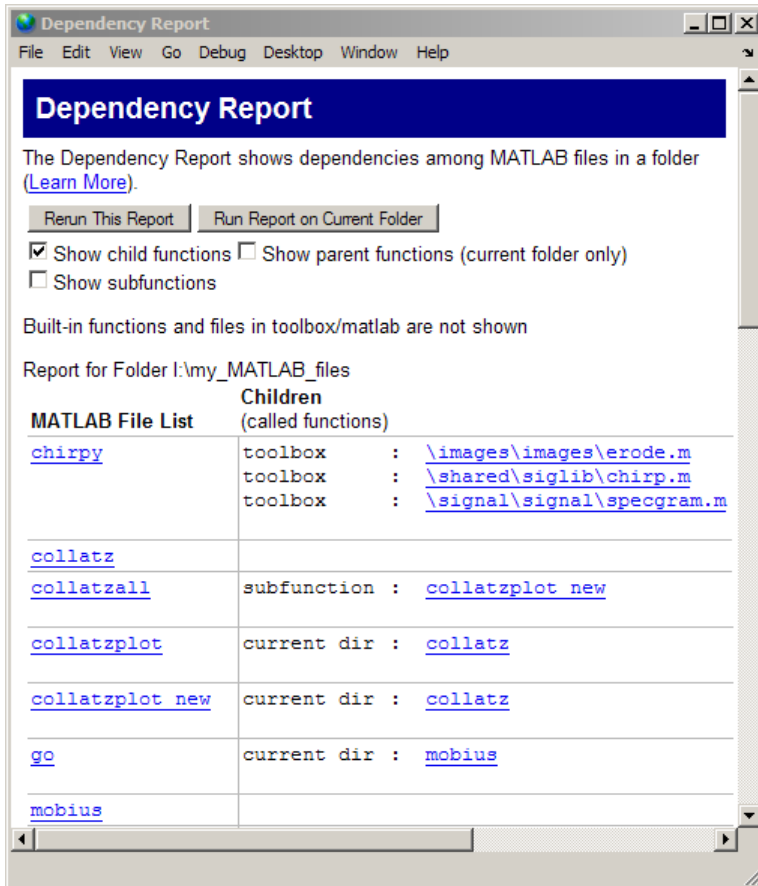
The report indicates where each child function resides, for example, in a specified toolbox. If the report specifies that the location of a child function is unknown, it can be because:

- The child function is not on the search path.
 - The child function is not in the current folder.
 - The file was moved or deleted.
- To list the files that call each MATLAB code file, select **Show parent functions**.
- The report limits the parent (calling) functions to functions in the current folder.
- To include subfunctions in the report, select **Show subfunctions**. The report lists subfunctions directly after the main function and highlights them in gray.

4 Click Run Report on Current Folder.

Reading and Working with Dependency Reports

The following image shows a Dependency Report. It indicates that `chirpy.m` calls two files in Signal Processing Toolbox and one in Image Processing Toolbox. It also shows that `go.m` calls `mobius.m`, which is in the current folder.



The Dependency Report includes the following:

- MATLAB File List

The list of files in the folder on which you ran the Dependency Report. Click a link in this column to open the file in the Editor.

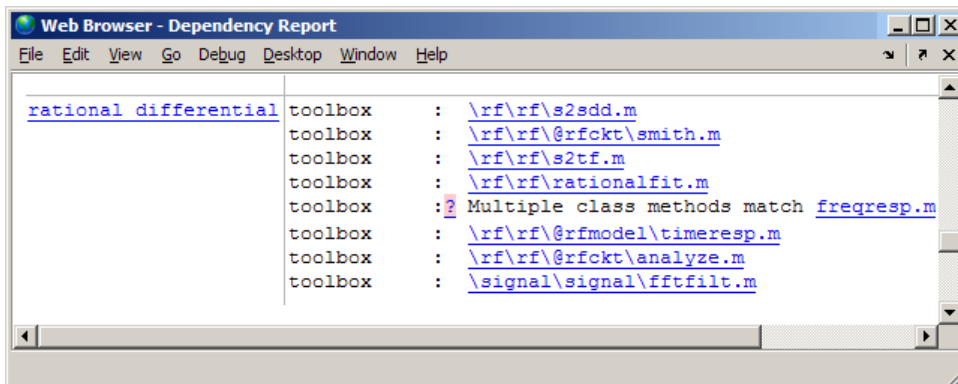
- Children

The function or functions called by the MATLAB file.

Click a link in this column to open the MATLAB file listed in the same row, and go to the first reference to the called function. For instance, suppose your Dependency Report appears as shown in the previous image. Clicking `\images\images\erode.m` opens `chirpy.m` and places the cursor at the first line that references `erode`. In other words, it does not open `erode.m`.

- Multiple class methods

Because the report is a static analysis, it cannot determine run-time data types and, therefore, cannot identify the particular class methods required by a file. If multiple class methods match a referenced method, the Dependency Report inserts a question mark link next to the file name. The question mark appears in the following image.



Click the question mark link to list the class methods with the specified name that MATLAB might use. MATLAB lists *almost all* the method files on the search path that match the specified method file (in this case, `freqresp.m`). Do not be concerned if the list includes methods of classes and MATLAB built-in functions that are unfamiliar to you.

It is not necessary for you to determine which file will be used. MATLAB determines which method to use using the object that the program calls at run time. The report provides this list to indicate possible toolboxes and method files used.

The following image shows the contents of the right side of the Web Browser after you click the question mark link.

```

toolbox : \rf\rf\s2sdd.m
toolbox : \rf\rf\@rfckt\smith.m
toolbox : \rf\rf\s2tf.m
toolbox : \rf\rf\rationalfit.m
toolbox : ? Multiple class methods match freqresp.m

Unable to determine which of the following files will run: (Learn More)
\control\ctrlobolete\freqresp.m
W:\bat\Akernel\perfect\matlab\toolbox\control\control\@lti\freqresp.m
W:\bat\Akernel\perfect\matlab\toolbox\ident\ident\@idproc\freqresp.m
W:\bat\Akernel\perfect\matlab\toolbox\ident\ident\@idpoly\freqresp.m
W:\bat\Akernel\perfect\matlab\toolbox\ident\ident\@idmodel\freqresp.m
W:\bat\Akernel\perfect\matlab\toolbox\ident\ident\@idfrd\freqresp.m
W:\bat\Akernel\perfect\matlab\toolbox\rf\rf\@rfmodel\freqresp.m

```


Identifying How Much of a File Ran When Profiled

Run the Coverage Report after you run the Profiler to identify how much of a file ran when it was profiled. For example, when you have an if statement in your code, that block of code might not run during profiling, depending on conditions.

You can view coverage details in the Profiler detail report, or by following these steps:

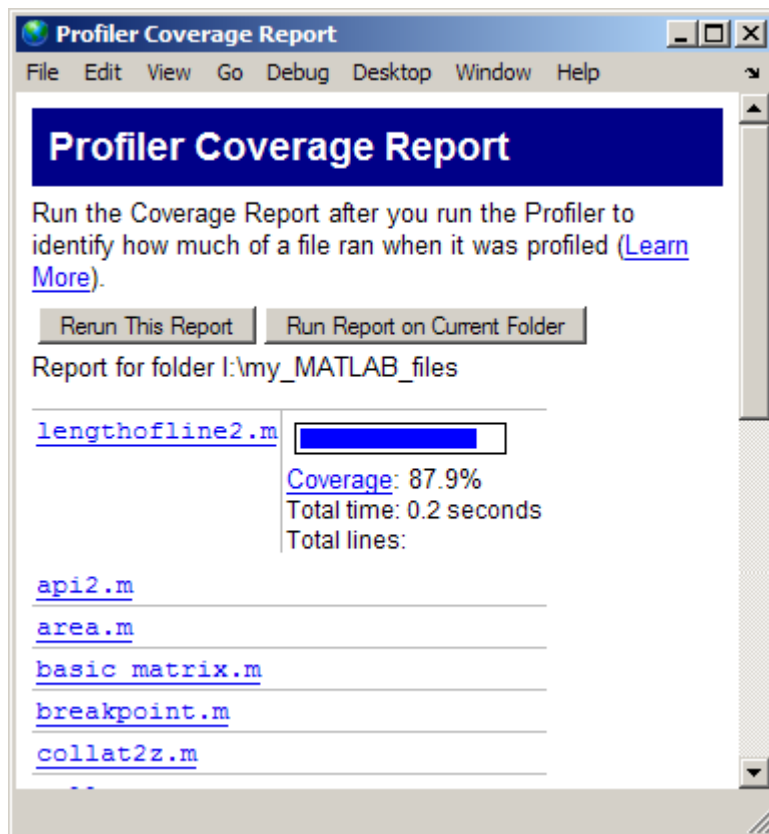
- 1 On the MATLAB desktop, select **Desktop > Profiler**.
- 2 Profile a MATLAB code file in the Profiler.

For detailed instructions, see “Profiling for Improving Performance” on page 10-27.

- 3 Select **Desktop > Current Folder** and navigate to the folder containing the file for which you ran the Profiler.
- 4 On the Current Folder browser toolbar, click the **Actions** button , and then select **Reports > Coverage Report**.

The Profiler Coverage Report appears, providing a summary of coverage for the file you profiled. In the image that follows, the profiled file is `lengthofline2.m`.

- 5 Click the **Coverage** link to see the Profile Detail Report for the file.



Using the Code Analyzer Report

In this section...

“Running the Code Analyzer Report” on page 10-22

“Changing Code Based on Messages” on page 10-24

“Other Ways to Access Messages” on page 10-25


Running the Code Analyzer Report

The Code Analyzer Report displays potential errors and problems, as well as opportunities for improvement in your code through M-Lint messages. For example, a common message indicates that a variable `foo` might be unused.

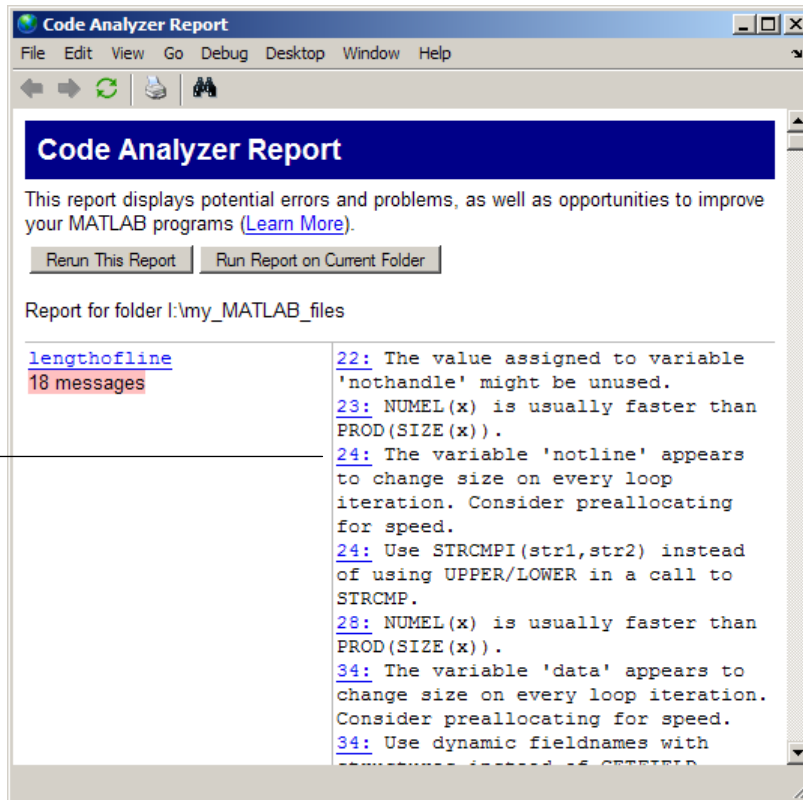
To run the Code Analyzer Report:

- 1 In the Current Folder browser, navigate to the folder that contains the files you want to check. To use the example shown in this documentation, `lengthoffline.m`, you can change the current folder by running

```
cd(fullfile(matlabroot,'help','techdoc','matlab_env','examples'))
```

- 2 If you plan to modify the example, save the file to a folder for which you have write access. Then, make that folder the current MATLAB folder. This example, saves the file in `I:\my_MATLAB_files`.
- 3 In the Current Folder browser toolbar, click the **Actions** button , and then select **Reports > Code Analyzer Report**.

The report displays in the MATLAB Web Browser, showing those files identified as having potential problems or opportunities for improvement.



Line number and message

- 4 For each message in the report, review the suggestion and your code. Click the line number to open the file in the Editor at that line, and change the file based on the message. Use the following general advice:
 - If you are unsure what a message means or what to change in the code, click the link in the message if one appears. For details, see “Preventing and Identifying Coding Problems” on page 9-107.
 - If the message does not contain a link, and you are unsure what a message means or what to do, search for related topics in the Help browser. For examples of messages and what to do about them, including specific changes to make for the example, `lengthoffline.m`, see “Changing Code Based on Messages” on page 10-24.

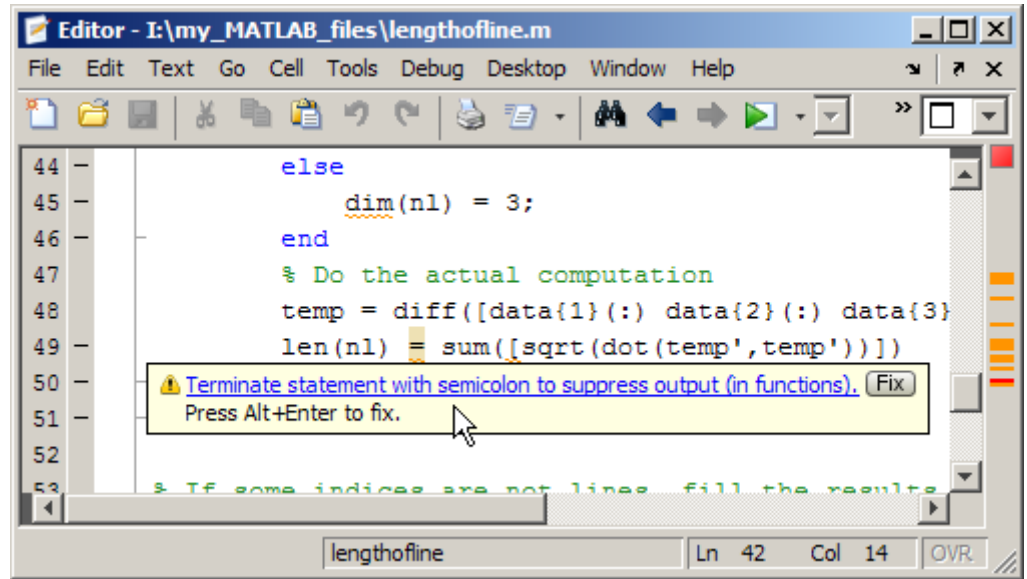
- The messages do not provide perfect information about every situation and in some cases, you might not want to change anything based on the message. For details, see “Understanding the Limitations of Code Analysis” on page 9-132.
 - If there are certain messages or types of messages you do not want to see, you can suppress them. For details, see “Suppressing Message Indicators and Messages” on page 9-117.
- 5** After modifying it, save the file. Consider saving the file to a different name if you made significant changes that might introduce errors. Then you can refer to the original file, if needed, to resolve problems with the updated file. Use **Tools > Compare Against** in the Editor to help you identify the changes you made to the file. For more information, see “Comparing Text Files” on page 7-52.
 - 6** Run and debug the file or files again to be sure that you have not introduced any inadvertent errors.
 - 7** If the report is displaying, click **Rerun This Report** to update the report based on the changes you made to the file. Ensure that the messages are gone, based on the changes you made to the files.

Changing Code Based on Messages

For information on how to correct the potential problems presented in M-Lint messages, use the following resources:

- Open the file in the Editor and click the link for an extended message in the tooltip, as shown in the image following this list. Not all messages have extended messages.
- Look for relevant topics in the Programming Fundamentals and “Programming Tips” documentation.
- Use the Help browser **Search** and **Index** panes to find documentation about terms presented in the messages.

The following image shows a tooltip with a link to an extended M-Lint message. The orange *line* under the = sign indicates a tooltip, including an extended message, displays if you hover over the = sign. The orange *highlighting* indicates that an automatic fix is available.



Other techniques to help you identify problems in and improve your code are in these topics:

- “Enable syntax highlighting” on page 9-22 in the Command Window and the Editor
- “Examining Errors” on page 3-7 generated when you run the file
- “Finding Errors, Debugging, and Correcting MATLAB Files” on page 9-104
- “Profiling for Improving Performance” on page 10-27 for improving performance

Other Ways to Access Messages

You can get M-Lint messages using any of the following methods. Each provides the same messages, but in a different format:

- Access the Code Analyzer Report for a file from the Editor **Tools** menu or from the Profiler detail report.
- Run the `mlint` function, which analyzes the specified file and displays messages in the Command Window.

- Run the `mlintrpt` function, which runs `mlint` and displays the messages in the Web Browser.
- Use automatic code analysis while you work on a file in the Editor. See “Preventing and Identifying Coding Problems” on page 9-107.

Profiling for Improving Performance

In this section...

“What Is Profiling?” on page 10-27
“Profiling Process and Guidelines” on page 10-28
“Using the Profiler” on page 10-30
“Profile Summary Report” on page 10-36
“Profile Detail Report” on page 10-38
“The profile Function” on page 10-46

What Is Profiling?

Profiling is a way to measure where a program spends time. To assist you in profiling, MATLAB software provides a graphical user interface, called the Profiler, which is based on the results returned by the `profile` function. Once you identify which functions are consuming the most time, you can determine why you are calling them. Then, look for ways to minimize their use and thus improve performance. It is often helpful to decide whether the number of times the code calls a particular function is reasonable. Because programs often have several layers, your code might 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 profiling spends most of its time on calls to a few built-in functions, you have probably optimized the code as much as you can.

Note When using the Parallel Computing Toolbox™ software, you can use the parallel profiler to profile parallel jobs. See “Using the Parallel Profiler” for details.

Profiling Process and Guidelines

Here is a general process you can follow to use the Profiler to improve performance in your code. This section includes the following topics:

- Using Profiling as a Debugging Tool
- “Using Profiling to Understand an Unfamiliar File” on page 10-29

Tip 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 are called most frequently. See “Profile Summary Report” on page 10-36 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 10-38 for more information.

Consider keeping a copy of your first detail report as a basis for comparison. After you change the function file, run the Profiler again and compare the reports.

- 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 called only 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 identical 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 code.

For example, if a particular section of a file did not run, you can look at the detail reports to see what lines did run. The detail report 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 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**. Using **Ctrl+C** can be useful when a file is taking much more time to run than expected.

Using Profiling to Understand an Unfamiliar File

For a lengthy MATLAB code file that you did not create, or with which you are unfamiliar, use the Profiler to see how the file actually works. Use the Profiler detail reports to see the lines called.

If there is an existing GUI tool (or file) like one that you want to create, start profiling, use the tool, then stop profiling. Look through the Profiler detail reports to see what functions and lines ran. This helps you determine the lines of code in the file that are most like the code you want to create.

Using the Profiler


Use the Profiler to help you determine where you can modify your code to make performance improvements. The Profiler is a tool that shows you where a file is spending its time. This section covers:

- “Opening the Profiler” on page 10-30
- “Running the Profiler” on page 10-30
- “Profiling a Graphical User Interface” on page 10-35
- “Profiling Statements from the Command Window” on page 10-35
- “Changing Fonts for the Profiler” on page 10-35

For information about the reports generated by the Profiler, see “Profile Summary Report” on page 10-36 and “Profile Detail Report” on page 10-38.

Opening the Profiler

You can use any of the following methods to open the Profiler:

- Select **Desktop > Profiler** from the MATLAB desktop.
- Click the Profiler button  in the MATLAB desktop toolbar.
- With a file open in the MATLAB Editor, select **Tools > Open Profiler**.
- Select one or more statements in the Command History window, right-click to view the context menu, and then select **Profile Code**.
- Type `profile viewer` in the Command Window:

Running the Profiler

To profile a MATLAB code file or a line of code:

- 1 If your system uses Intel® multi-core chips, consider restricting the active number of CPUs to one.

See one of the following for details:

- “Intel Multi-Core Processors — Setting for Most Accurate Profiling on Windows Systems” on page 10-32

- “Intel Multi-Core Processors — Setting for Most Accurate Profiling on Linux Systems” on page 10-33

2 In the Command Window, type `profile viewer`.

3 Do one of the following in the Profiler:

- For a statement you have not profiled in the current MATLAB session:

In the **Run this code** field, type the statement you want to run.

For example, you can run the Lotka-Volterra demo, which is provided with MATLAB demos (`lotkademo`):

```
[t,y] = ode23('lotka',[0 2],[20;20])
```

- For a statement you previously profiled in the current MATLAB session:

1 Select the statement from the list box—MATLAB automatically starts profiling the code.

2 Skip to step 5.

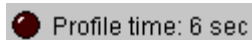
4 Click **Start Profiling**.

While the Profiler is running, the **Profile time** indicator is green and the number of seconds it reports increases. The **Profile time** indicator appears at the top right of the Profiler window.



A screenshot of a Profiler window showing a green circular indicator and the text "Profile time: 1 sec".

When the Profiler finishes, the **Profile time** indicator becomes dark red and shows the length of time the Profiler ran. The statements you profiled display as having been executed in the Command Window.



A screenshot of a Profiler window showing a dark red circular indicator and the text "Profile time: 6 sec".

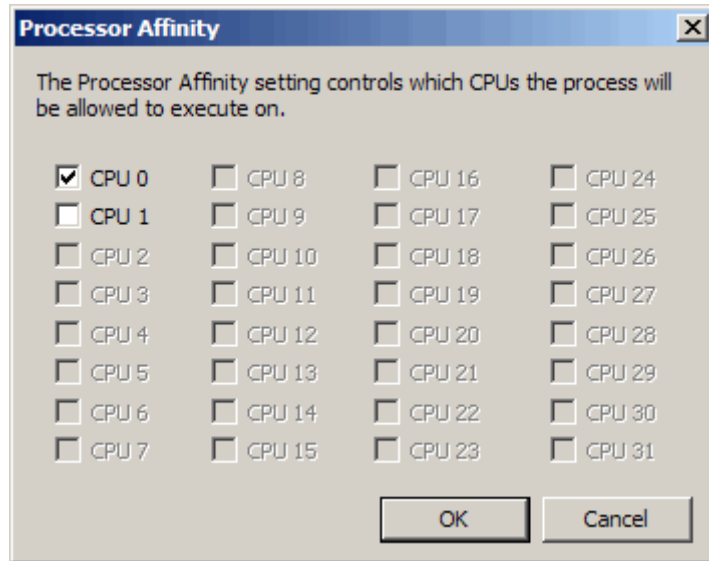
This time is not the actual time that your statements took to run. It is the wall clock (or `tic/toc`) time elapsed from when you clicked **Start Profiling** until profiling stops. If the time reported is very different from what you expected (for example, hundreds of seconds for a simple statement), you might have had profiling on longer than you realize. This time also does not match the time reported in Profiler Summary report statistics, which is based on `cpu` time by default, not wall clock time. To view profile statistics based on wall clock time, use the `profile` function with the `-timer real` option as shown in “Using the profile Function to Change the Time Type Used by the Profiler” on page 10-50.

- 5** 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 10-36.
- 6** Reset the number of active CPUs to the original setting if you restricted the number in step 1.

Intel Multi-Core Processors – Setting for Most Accurate Profiling on Windows Systems. If your system uses Intel multi-core chips, and you plan to profile using CPU time, set the number of active CPUs to one before you start profiling. This results in the most accurate and efficient profiling.

- 1** Open Windows Task Manager.
- 2** On the **Processes** tab, right-click `MATLAB.exe` and then click **Set Affinity**.
The Processor Affinity dialog box opens.
- 3** In the Processor Affinity dialog box, note the current settings, and then clear all the CPUs except one.

Your Processor Affinity dialog box should appear like the following image.



4 Click **OK**.

5 Reset the state of the Profiler so that it recognizes the processor affinity changes you made. The easiest way to do so is to change the Profiler timer setting to `real` and then back to `cpu`, by issuing the following in the Command Window:

```
profile -timer real
profile -timer cpu
```

Remember to set the number of CPUs back to their original settings when you finish profiling. Rerun the preceding steps, and restore the original selections in the Processor Affinity dialog box in step 3.

Intel Multi-Core Processors – Setting for Most Accurate Profiling on Linux Systems. If your system uses Intel multi-core chips, and you plan to profile using CPU time, set the number of active CPUs to one before you start profiling. This results in the most accurate and efficient profiling.

For example, to set the processor affinity to one you can use the Linux `taskset` command, as follows:

- 1 Get the process ID (PID) of the currently running MATLAB instance:

```
ps -C MATLAB
PID  TTY  TIME      CMD
8745 pts/1 00:00:50  MATLAB
```

The PID in this example is 8745.

- 2 Call the Linux `taskset` command to get the current number of active CPUs for the MATLAB process:

```
taskset -pc 8745
pid 8745's current affinity list: 0-3
```

The `-p` option specifies that `taskset` operate on an existing PID, instead of creating a new task. The `-c` option lists the processor numbers.

- 3 Call the Linux `taskset` command again — this time to set the processor affinity for the MATLAB process to one CPU (that is, CPU #0):

```
taskset -pc 0 8745
pid 8745's current affinity list: 0-3
pid 8745's  new affinity list: 0
```

For more information on the syntax of `taskset`, execute `man taskset` from a Linux terminal.

Reset the state of the Profiler so that it recognizes the processor affinity changes you made. The easiest way to do this is to change the Profiler timer setting to `real` and then back to `cpu`, by issuing the following in the Command Window:

```
profile -timer real
profile -timer cpu
```

Remember to set the number of CPUs back to its original setting when you finish profiling. Rerun the preceding steps, and then restore the original number of CPUs returned in step 2.

Profiling a Graphical User Interface

You can run the Profiler for a graphical user interface, such as the Filter Design and Analysis tool included with Signal Processing Toolbox. You can also run the Profiler for an interface you created, such as one built using GUIDE.

To profile a graphical user interface:

- 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 **Stop Profiling**, step 1, until after you start the graphical interface.)
- 3** Use the graphical interface. When you finish, 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:

- 1** In the Profiler, clear the **Run this code** field and click **Start Profiling**.
- 2** In the Command Window, enter and run the statements you want to profile.
- 3** After running all the statements, click **Stop Profiling** in the Profiler.

The Profile Summary report appears in the Profiler.

Changing Fonts for the Profiler

To change the fonts used in the Profiler:

- 1** Select **File > Preferences > Fonts** to open the Font Preferences dialog box.
- 2** Select the code or text font that you want to use in the Profiler. The Profiler is an HTML Proportional Text tool. For more information, click the **Help** button in the dialog box.

3 Click **Apply** or **OK**. The Profiler font reflects the changes.

Profile Summary Report

The Profile Summary report presents statistics about the overall execution of the function and provides summary statistics for each function called. The report formats these values in four columns.

- **Function Name** — A list of all the functions and subfunctions called by the profiled function. When first displayed, the functions are listed in order by the amount of time they took to process. To sort the functions alphabetically, click the **Function Name** link at the top of the column.
- **Calls** — The number of times the function was called while profiling was on. To sort the report by the number of times functions were called, click the **Calls** link at the top of the column.
- **Total Time** — The total time spent in a function, including all child functions called, in seconds. The time for a function includes time spent on child functions. To sort the functions by the amount of time they consumed, click the **Total Time** link at the top of the column. By default, the summary report displays profiling information sorted by **Total Time**. Be aware that the Profiler itself uses some time, which is included in the results. Also note that total time can be zero for files whose running time was inconsequential.
- **Self Time** — The total time spent in a function, *not* including time for any child functions called, in seconds. If MATLAB can determine the amount of time spent for profiling overhead, MATLAB excludes it from the self time also. (MATLAB excludes profiling overhead from the total time and the time for individual lines in the Profile Detail Report as well.)

The bottom of the Profiler page contains a message like one of the following, depending on whether MATLAB can determine the profiling overhead:

- Self time is the time spent in a function excluding:
 - The time spent in its child functions
 - Most of the overhead resulting from the process of profiling

In the present run, self time excludes 0.240 secs of profiling overhead. The amount of remaining overhead reflected in self time cannot be determined, and therefore is not excluded.


- Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes some overhead resulting from the process of profiling.

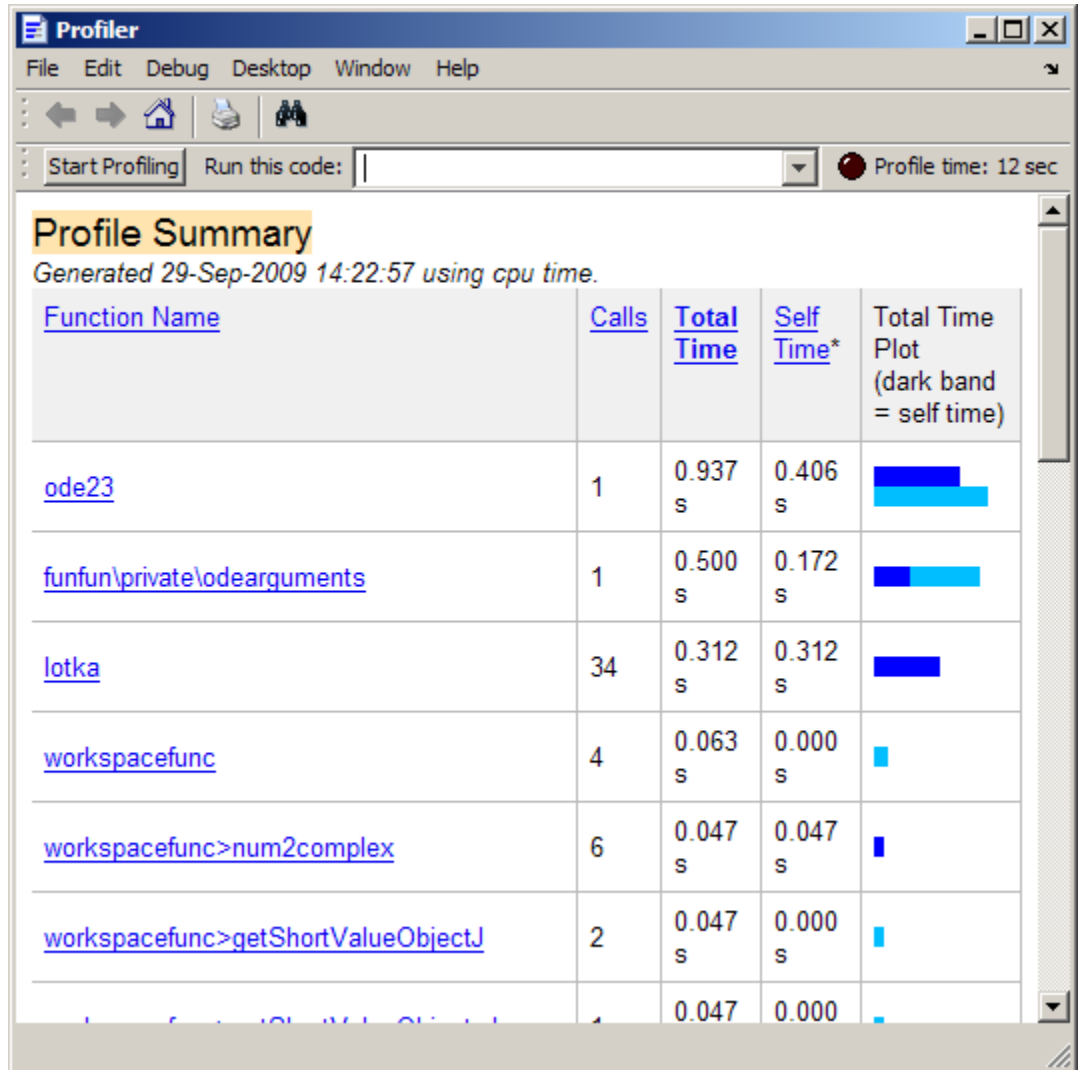
To sort the functions by this time value, click the **Self Time** link at the top of the column.

- **Total Time Plot** — Graphic display showing self time compared to total time.

The following is an image of the summary report for the Lotka-Volterra model used in “Example: Using the profile Function” on page 10-47.


In the summary report, you can:

- Print it, by clicking the Print button .
- Get more detailed information about a particular function by clicking its name in the **Function Name** column. See “Profile Detail Report” on page 10-38 for more information.
- Sort by a given column by clicking the name of the column. For example, click [Function Name](#) to sort by the names of the functions included in the summary report.



Profile Detail Report

The Profile Detail report shows profiling results for a selected function that was called during profiling. A Profile Detail report has seven sections. The topics that follow describe each section. By default, the Profile Detail report

includes all seven sections, although, depending on the function, not every section contains data. To return to the Profile Summary report from the Profile Detail report, click the Home button  in the toolbar.

The following topics provide details about opening and using a Profile Detail Report:

- “Opening the Profile Detail Report” on page 10-39
- “Controlling the Contents of the Detail Report Display” on page 10-39
- “Profile Detail Report Header” on page 10-40
- “Parent Functions” on page 10-41
- “Busy Lines” on page 10-41
- “Child Functions” on page 10-42
- “Code Analyzer Results” on page 10-43
- “File Coverage” on page 10-44
- “Function Listing” on page 10-45

Opening the Profile Detail Report

To open the Profile Detail Report:

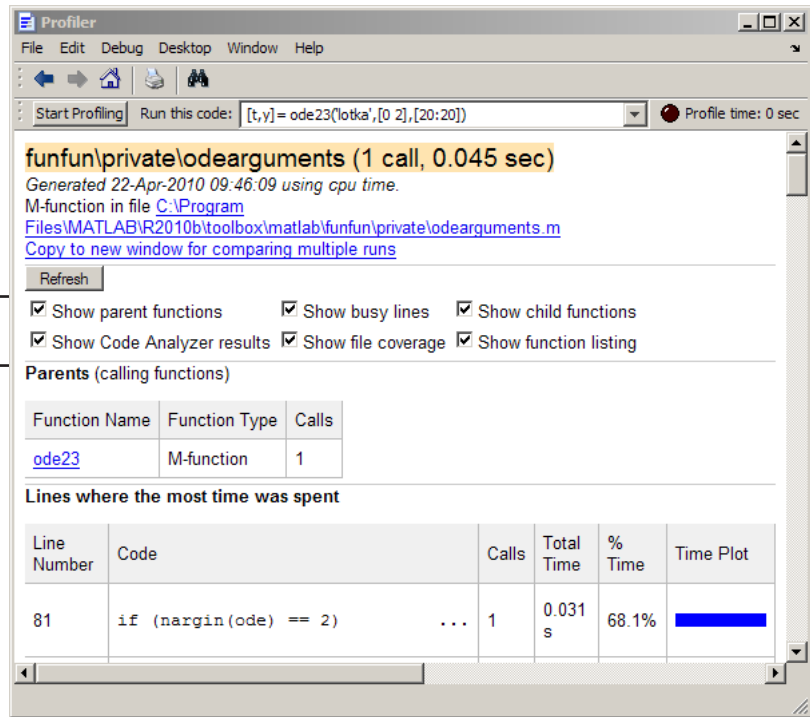
- 1 Create a Profile Summary report, as described in “Using the Profiler” on page 10-30.
- 2 Click a function name listed in the Profile Summary report.

Controlling the Contents of the Detail Report Display

To specify which sections the Profile Detail Report includes:

- 1 Select report options from the set of check boxes at the top of the report.
- 2 Click the **Refresh** button.

Report Options



Profile Detail Report Header

The detail report header includes:

- The name of the function profiled
- The number of times the profiled function was called in the parent function
- The amount of time the profiled function used
- A link that opens the function in your default text editor
- A link that copies the report to a separate window

Creating a copy of the report is helpful when you change the file, run the Profiler for the updated file, and compare the Profile Detail reports for the two runs. Do not change files provided with products from MathWorks, that is, files in the *matlabroot* / *toolbox* folders.

Parent Functions

To include the **Parents** section in the Detail Report, select the **Show parent functions** check box. This section of the report provides information about the parent functions, with links to their detail reports. Click the name of a parent function to open a Detail Report for that parent function.

Show parent functions Show busy lines Show child functions
 Show Code Analyzer results Show file coverage Show function listing

Parents (calling functions)

Function Name	Function Type	Calls
ode23	M-function	1







Busy Lines

To include information about the lines of code that used the most amount of processing time in the detail report, select the **Show busy lines** check box.

Refresh




Show parent functions Show busy lines Show child functions
 Show Code Analyzer results Show file coverage Show function listing

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
110	<code>f0 = feval(ode,t0,y0,args{:});...</code>	1	0.482 s	58.5%	
81	<code>if (nargin(ode) == 2) ...</code>	1	0.110 s	13.3%	
148	<code>rtol = odeget(options,'RelTol'...</code>	1	0.063 s	7.6%	
184	<code>htry = abs(odeget(options,'Ini...</code>	1	0.032 s	3.9%	
105	<code>if any(tdir*diff(tspan) <=...</code>	1	0.032 s	3.9%	
All other lines			0.107 s	12.9%	
Totals			0.825 s	100%	

Child Functions

To include the **Children** section of the detail report, select the **Show child functions** check box. This section of the report lists all the functions called by the profiled function. If the called function is a MATLAB code file, you can view the source code for the function by clicking its name.

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
lotka	M-function	1	0.466 s	56.5%	
odeget	M-function	5	0.032 s	3.9%	
double.superiorfloat	M-function	1	0 s	0%	
Self time (built-ins, overhead, etc.)			0.327 s	39.6%	
Totals			0.825 s	100%	

Code Analyzer Results

To include the **Code Analyzer results** section in the detail report display, select the **Show Code Analyzer results** check box. This section of the report provides information about problems and potential improvements for the function. For more information, see “Using the Code Analyzer Report” on page 10-22.

Show parent functions
 Show busy lines
 Show child functions
 Show Code Analyzer results
 Show file coverage
 Show function listing

M-Lint results

Line number	Message
52	EXIST with two input arguments is generally faster and clearer than with one input argument.
80	EXIST with two input arguments is generally faster and clearer than with one input argument.
90	The value assigned to variable 'ME' might be unused.

File Coverage

To include the **Coverage results** section in the detail report display, select the **Show file coverage** check box. This section of the report provides statistical information about the number of lines in the code that executed during the profile run.

Refresh

Show parent functions Show busy lines Show child functions
 Show Code Analyzer results Show file coverage Show function listing

Coverage results
[\[Show coverage for parent directory \]](#)

Total lines in function	190
Non-code lines (comments, blank lines)	57
Code lines (lines that can run)	133
Code lines that did run	53
Code lines that did not run	80
Coverage (did run/can run)	39.85 %

Function Listing

To include the **Function listing** section in the detail report display, select the **Show function listing** check box. If the file is a MATLAB code file, the Profile Detail report includes three columns:

- The first column lists the execution time for each line.
- The second column lists the number of times the line was called
- The third column specifies the source code for the function.

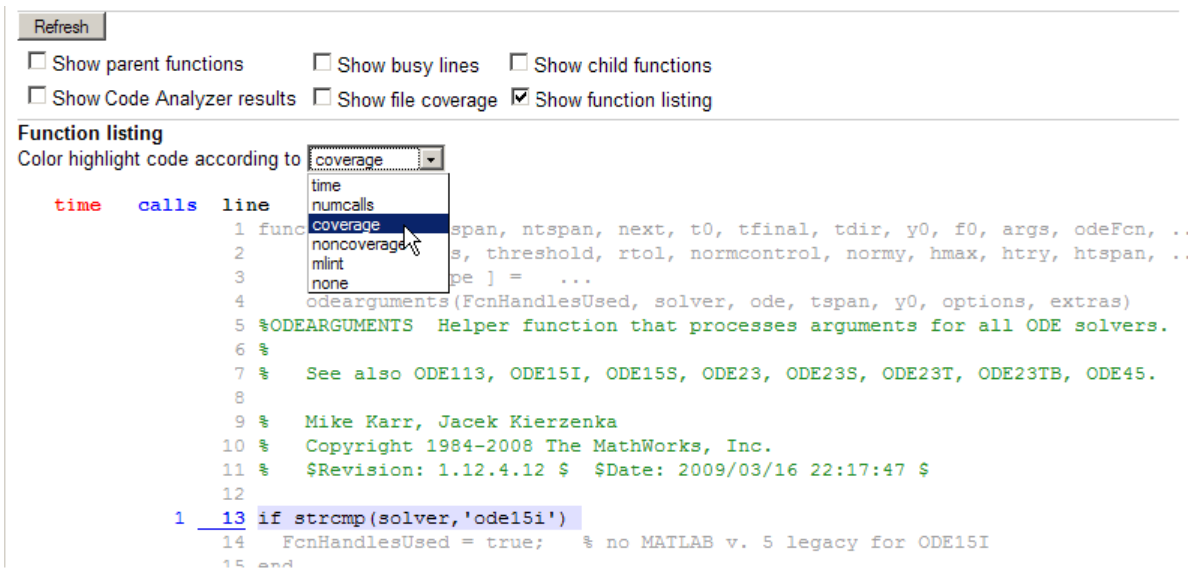
In the function listing, the color of the text indicates the following:

- Green — Comment lines
- Black — Lines of code that executed
- Gray — Lines of code that did not execute

By default, the Profile Detail report highlights lines of code with the longest execution time. The darker the highlighting, the longer the line of code took to execute.

To change the lines of code highlighted based on other criteria, use the drop-down menu in this section of the detail report. The color of the highlighting changes, depending on the drop-down option you choose. You can choose to highlight lines of code called the most, lines of code that were (or were not) executed, or lines called out by M-Lint. Or, you can turn off highlighting by selecting none.

The following image shows that lines highlight in blue when you select coverage from the drop-down menu.



The profile Function

The Profiler is based on the results returned by the `profile` function. The `profile` function provides some features that are not available in the GUI. For example, use the `profile` function to specify that statistics display the time it takes for statements to run as clock time, instead of CPU time.

This section includes the following topics with respect to the `profile` function:

- “Example: Using the profile Function” on page 10-47
- “Accessing profile Function Results” on page 10-48

- “Saving profile Function Reports” on page 10-50
- “Using the profile Function to Change the Time Type Used by the Profiler” on page 10-50

Example: Using the profile Function

This example demonstrates how to run `profile`:

- 1** To start `profile`, type the following in the Command Window:

```
profile on
```

- 2** Execute a MATLAB code 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 files you run. It will add these new statistics to those statistics generated in the previous steps.

- 5** Stop `profile` when you finish gathering statistics.

```
profile off
```

- 6** To view the profile data, call `profile` specifying the `'info'` argument. The `profile` function returns data in a structure.

```
p = profile('info')
```

```
p =
```

```
FunctionTable: [10x1 struct]
FunctionHistory: [2x0 double]
ClockPrecision: 3.3333e-010
ClockSpeed: 3.0000e+009
Name: 'MATLAB'
```

The `FunctionTable` indicates that statistics were gathered for 10 functions.

- 7 To save the profile report, use the `profsave` function. This function stores the profile information in separate HTML files, for each function listed in `FunctionTable` of `p`.

```
profsave(p)
```

By default, `profsave` puts these HTML files in a subfolder of the current folder named `profile_results`, and displays the summary report in your system browser. You can specify another folder name as an optional second argument to `profsave`.

Accessing profile Function Results

The `profile` function returns results in a structure. This example illustrates how you can access these results:

- 1 To start profile, specifying the `history` option, type the following in the Command Window:

```
profile on -history
```

The `history` option specifies that the report include information about the sequence of functions as they are entered and exited during profiling.

- 2 Execute a MATLAB code file. This example runs the Lotka-Volterra predator-prey population model. For more information about this model, type `lotkademo`, which runs a demonstration.

```
[t,y] = ode23('lotka',[0 2],[20;20]);
```

- 3 Stop profiling.

```
profile off
```


- 4** Get the structure containing profile results.

```
stats = profile('info')
stats =
    FunctionTable: [43x1 struct]
    FunctionHistory: [2x754 double]
    ClockPrecision: 3.3333e-010
    ClockSpeed: 3.0000e+009
    Name: 'MATLAB'
```

- 5** The FunctionTable field is an array of structures, where each structure represents a MATLAB function (M-function), MATLAB subfunction, MEX-function, or, because the builtin option is specified, a MATLAB built-in function.

```
stats.FunctionTable

ans =

41x1 struct array with fields:
    CompleteName
    FunctionName
    FileName
    Type
    NumCalls
    TotalTime
    TotalRecursiveTime
    Children
    Parents
    ExecutedLines
    IsRecursive
    PartialData
```

- 6** View the second structure in FunctionTable.

```
stats.FunctionTable(2)

ans =
    CompleteName: [1x79 char]
```

```

        FunctionName: 'ode23'
            FileName: [1x73 char]
                Type: 'M-function'
            NumCalls: 1
            TotalTime: 0.3902
    TotalRecursiveTime: 0
            Children: [20x1 struct]
            Parents: [0x1 struct]
    ExecutedLines: [139x3 double]
            IsRecursive: 0
            PartialData: 0
    
```

- 7** To view the history data generated by `profile`, view the `FunctionHistory`, for example, `stats.FunctionHistory`. The history data is a 2-by-n array. The first row contains Boolean values, where 0 (zero) means entrance into a function and 1 means exit from a function. The second row identifies the function being entered or exited by its index in the `FunctionTable` field. To see how to create a formatted display of history data, see the example on the `profile` reference page.

Saving profile Function Reports

To save the profile report, use the `profsave` function.

This function stores the profile information in separate HTML files, for each function listed in the `FunctionTable` field of the structure, `stats`.

```
profsave(stats)
```

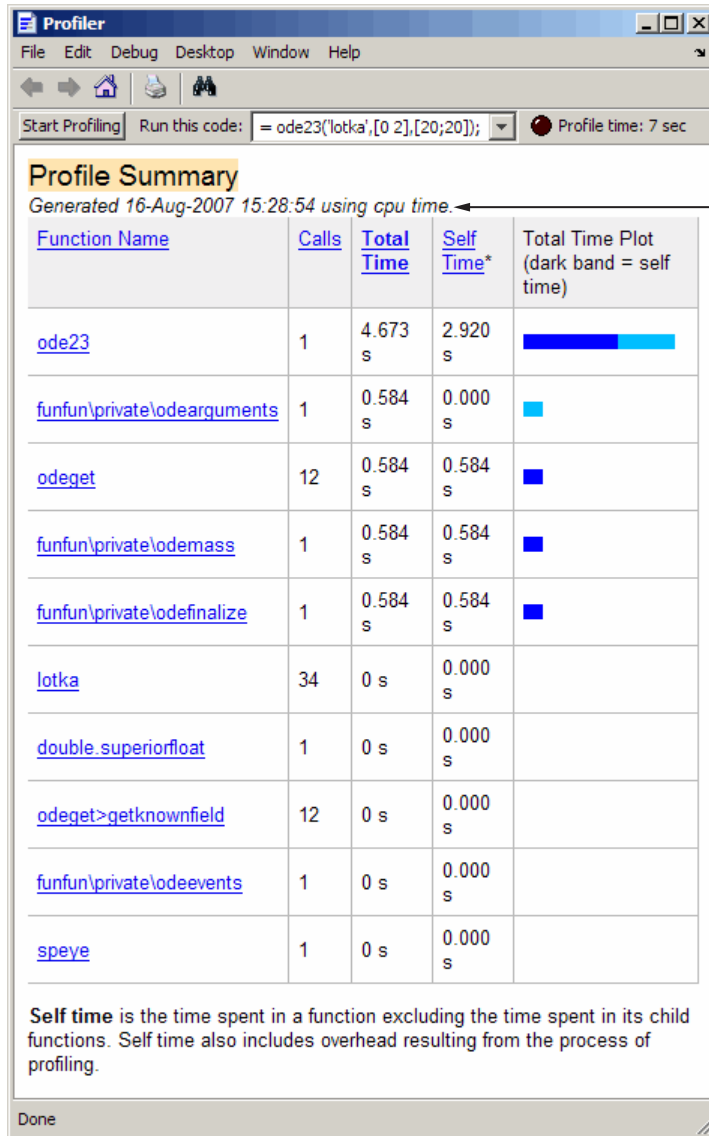
By default, `profsave` puts these HTML files in a subfolder of the current folder named `profile_results`. You can specify another folder name as an optional second argument to `profsave`.

```
profsave(stats, 'mydir')
```

Using the profile Function to Change the Time Type Used by the Profiler

By default, MATLAB generates the Profiler Summary report using CPU time, as opposed to real (wall clock) time. This example illustrates how you can direct MATLAB to use real time instead.

The following image shows the Profiler Summary report as it appears by default, using CPU time.



Generated using cpu time

Specify that the Profiler use real time instead, by using the `profile` function with the `-timer real` option, as shown in this example:

- 1** If the Profiler is currently open, close the Profiler, and if prompted, stop profiling.
- 2** Set the timer to real time by typing the following in the Command Window:

```
profile on -timer real
```

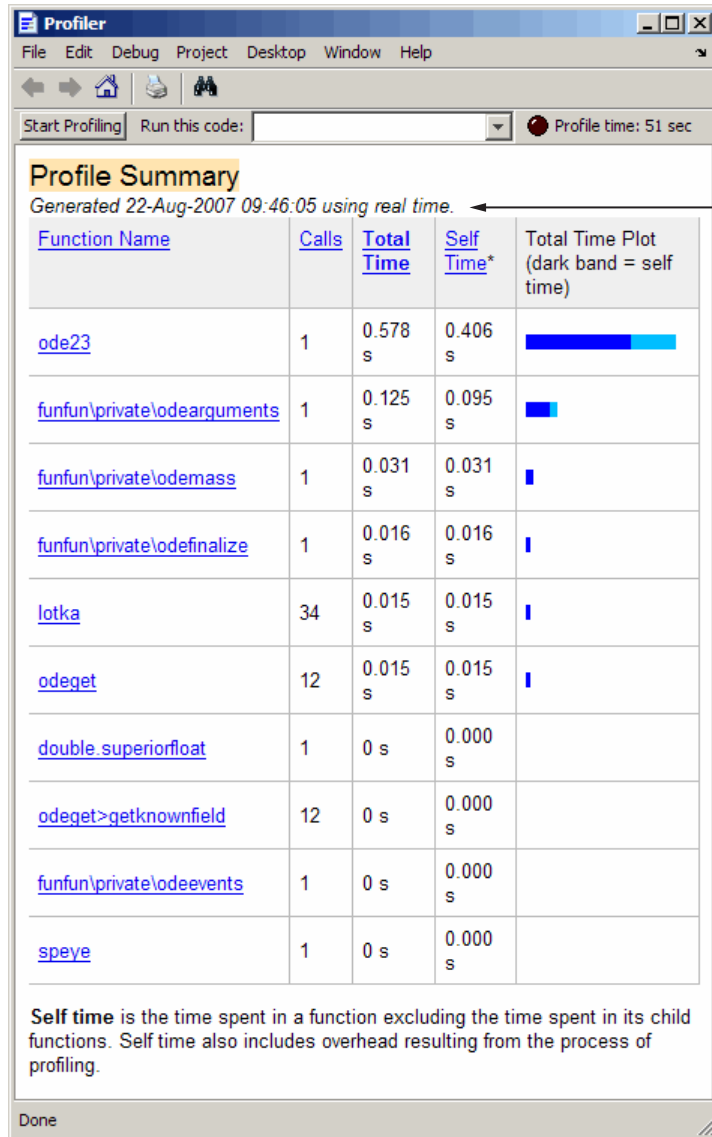
- 3** Run the file that you want to profile. This example runs the Lotka-Volterra predator-prey population model.

```
[t,y] = ode23('lotka',[0 2],[20;20]);
```

- 4** Open the Profiler by typing the following in the Command Window:

```
profile viewer
```

The Profiler opens and indicates that it is using real time, as shown in the following image.



Generated using real time

- 5 To change the timer back to using CPU time:
 - a Close the Profiler, and if prompted, stop profiling.

b Type the following in the Command Window:

```
profile on -timer cpu
```

c Type the following in the Command Window to reopen the Profiler:

```
profile viewer
```

Publishing MATLAB Code

MATLAB software lets you mark up MATLAB code and publish it to various output file formats.

- “Overview of Publishing MATLAB Code” on page 11-2
- “Marking Up MATLAB Comments for Publishing” on page 11-17
- “Marking Up MATLAB Code for Publishing” on page 11-60
- “Specifying Output Preferences for Publishing” on page 11-64
- “Summary of Options for Presenting Your Code to Others” on page 11-106

Overview of Publishing MATLAB Code

In this section...

“What Is Meant by Publishing MATLAB Code?” on page 11-2

“Using Code Cells” on page 11-2

“Process for Publishing MATLAB Code” on page 11-3

“Example of Published MATLAB Code” on page 11-4

“Adding the Markup for the Example” on page 11-10

What Is Meant by Publishing MATLAB Code?

MATLAB software enables you to publish your MATLAB code quickly, so you can describe and share your code with others, even if they do not have MATLAB software. You can include the following within the file you want to publish:

- Commentary on the code, including bulleted and numbered lists, bold and monospace font, preformatted text, LaTeX equations, and so on
- MATLAB code
- Results of running the code, including output to the Command Window and figures created or modified by the code

You can publish in various formats, including HTML, XML, and LaTeX. If Microsoft Word or Microsoft® PowerPoint® applications are on your Microsoft Windows system, you can publish to their formats as well.

If you have an active Internet connection, you can watch the Publishing M Code from the Editor video demo for an overview of the major publishing features using cells with text markup.

Using Code Cells

After you write and debug MATLAB code, if you insert code cells and commentary using text markup features, you can publish the code as a document. A code cell is a section of MATLAB code (see “What Are Code

Cells?” on page 9-175). For the purposes of publishing, a code cell can be either of the following, or both:

- A section of the code that you want to present as a titled subsection within the output
- A portion of code for which you want the results of code evaluation to display as it occurs (for example, each iteration of a `for` loop)

Any code cell features that you use for evaluating and improving your code, as described in “Evaluating Subsections of Files Using Code Cells” on page 9-175, you also can use for publishing purposes. However, to display comments in the output document, those comments must appear at the start of a code cell, before any executable code. This can require you to change code cells that you inserted for evaluating subsections of code. If you do so, be aware that this changes the cells for evaluation purposes, as well.

“Example of Published MATLAB Code” on page 11-4 shows how the code cells and formatted comments appear you publish MATLAB code.

Although you typically include the text markup after you write and debug the code, you can also include text markup as you write the code. Or, you can do both.

Note Cell mode is supported for use with files containing MATLAB code only. It is not intended for use with other text files.

Process for Publishing MATLAB Code

The overall process to publish a MATLAB code file using code cell features in the Editor is as follows:

- 1 Open your file in the Editor.
- 2 Select **Cell > Insert Text Markup** as described in “Marking Up MATLAB Comments for Publishing” on page 11-17.

This enables you to specify how MATLAB comments appear in the output. For example, you can specify that comments appear as bold or monospaced text.

- 3** To publish the file, do one of the following, as described in “Specifying Output Preferences for Publishing” on page 11-64:
 - To publish the file with default publishing properties, select **File > Publish *file name***. When you use this method, MATLAB publishes the file to HTML in an `/html` subfolder of the folder that contains the file you are publishing. However, if you previously specified custom property values, as described in the next list item, publishing uses the last configuration you specified. The output file formats and folder can be different from the default.
 - To specify custom publishing properties, select **File > Publish Configuration for *file name* > Edit Publish Configurations for *file name***, adjust properties, and then click **Publish**. You can, for example, choose to include or exclude the executable code from the output.

Example of Published MATLAB Code

This section provides an example to demonstrate how MATLAB code appears when published. It shows how the file appears before and after text markup is added to code cells to achieve the desired results. This section contains the following topics:

- “Sample File Before Adding Markup” on page 11-4
- “Published Sample File Before Adding Markup” on page 11-5
- “Published Sample File After Adding Markup” on page 11-7

For detailed information on inserting text markup, see “Marking Up MATLAB Comments for Publishing” on page 11-17.

Sample File Before Adding Markup

```
function fourier_demo
    t = 0:.1:pi*4;
    y = sin(t);
```

```
updatePlot(t,y);

% In each iteration of the for loop add an odd
% harmonic to y. As "k" increases, the output
% approximates a square wave with increasing accuracy.

for k = 3:2:9
    % Perform the following mathematical operation
    % at each iteration:
    y = y + sin(k*t)/k;

    display(sprintf('When k = %.1f',k));
    display('Then the plot is:');
    updatePlot (t,y)
end

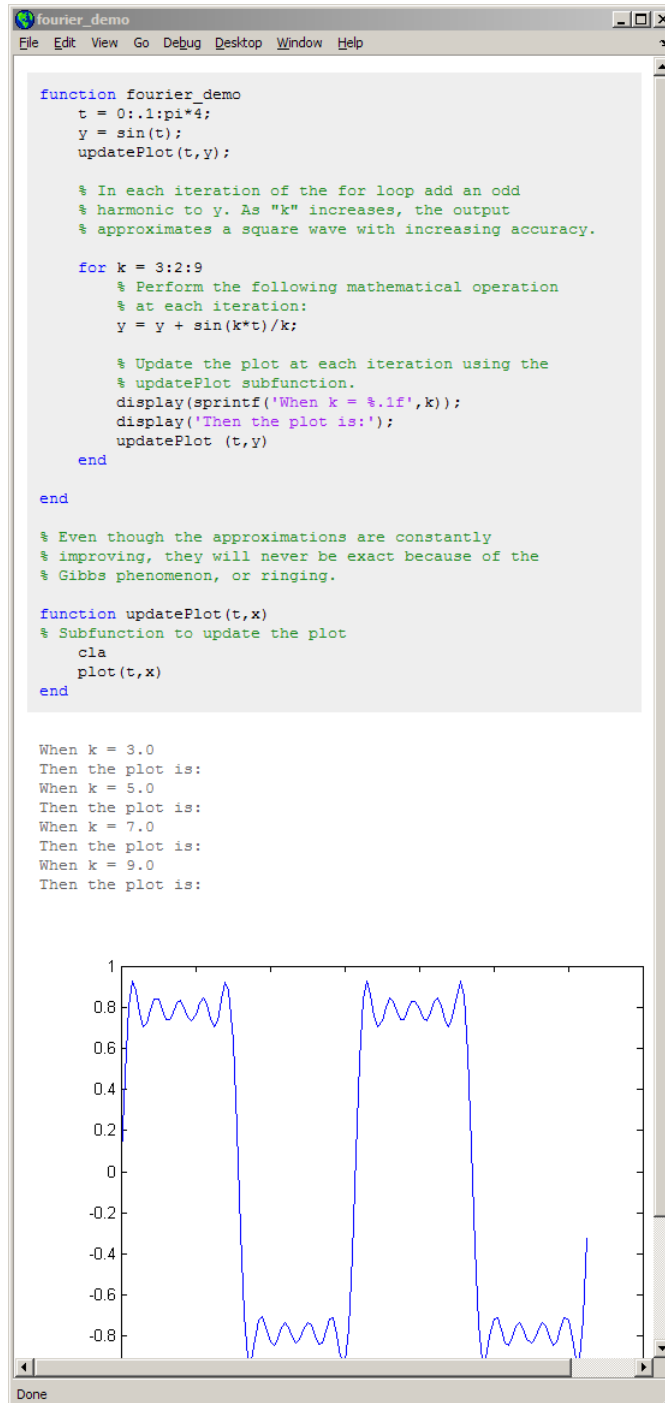
end

% Even though the approximations are constantly
% improving, they will never be exact because of the
% Gibbs phenomenon, or ringing.

function updatePlot(t,x)
% Subfunction to update the plot
    cla
    plot(t,x)
end
```

Published Sample File Before Adding Markup

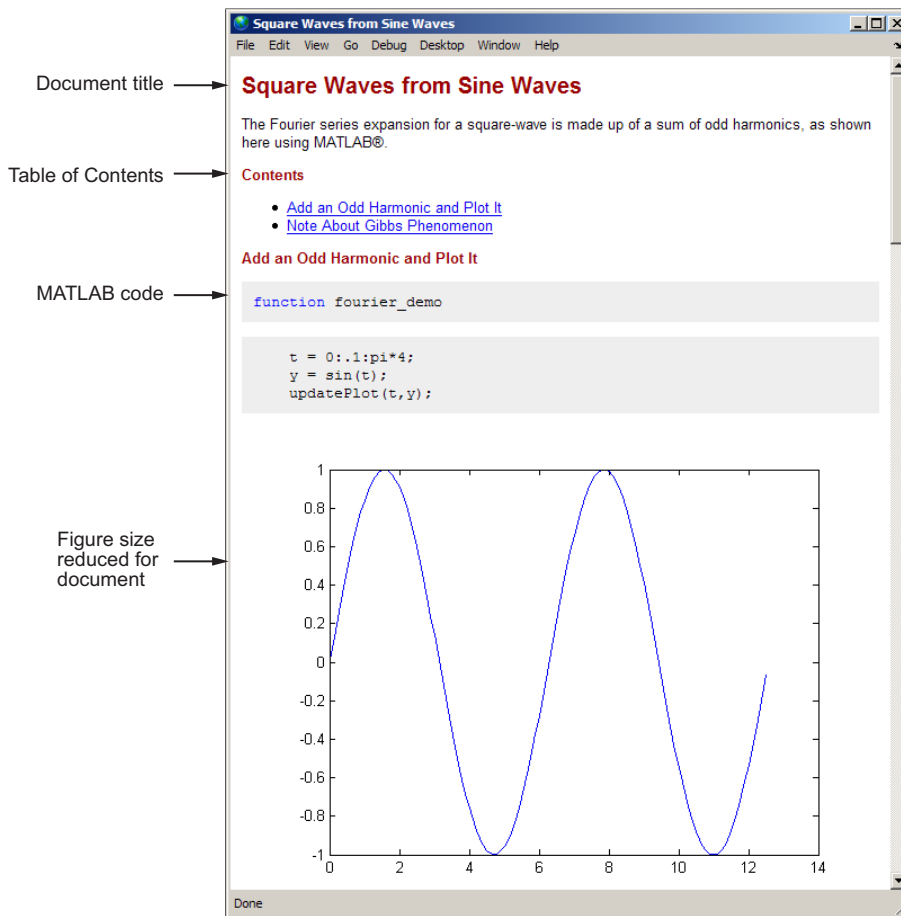
Before you add text markup and cell breaks, publishing `fourier_demo.m` includes the last plot generated by the for loop, but otherwise, has little effect. For example, if you select **File > Publish fourier_demo.m**, the output, as shown in the following figure, are of limited use.



Published Sample File After Adding Markup

If you add comments for clarity, apply text markup, and insert code cell breaks, as described in “Adding the Markup for the Example” on page 11-10, publishing the code transforms the output as shown in the following three figures:

- The first figure shows the top of the output.
- The second figure shows the middle of the output.
- The third figure shows the bottom of the output.



The letter k is italic →

Equation in TeX format →

Includes each iteration of the for loop →

```

Square Waves from Sine Waves
File Edit View Go Debug Desktop Window Help
In each iteration of the for loop add an odd harmonic to y. As k increases, the output
approximates a square wave with increasing accuracy.

for k = 3:2:9

Perform the following mathematical operation at each iteration:


$$y = y + \frac{\sin(k * t)}{k}$$

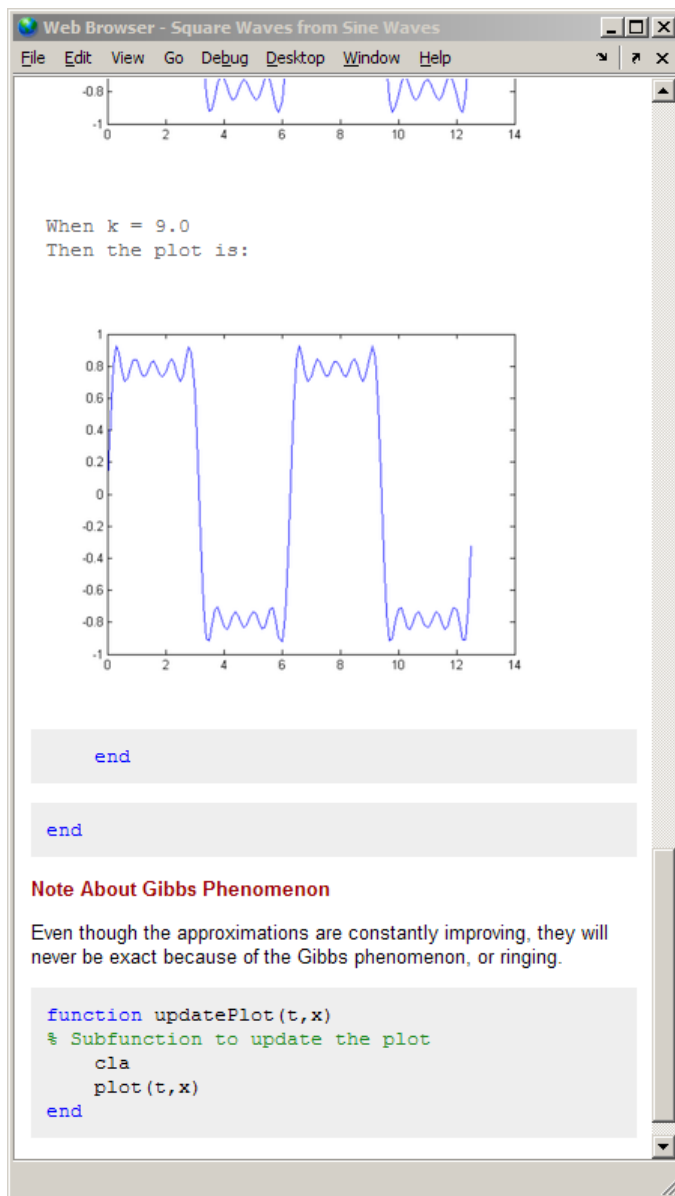

y = y + sin(k*t)/k;

display(sprintf('When k = %.1f',k));
display('Then the plot is:');
updatePlot (t,y)

When k = 3.0
Then the plot is:

1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0 2 4 6 8 10 12 14

When k = 5.0
Then the plot is:
    
```



Adding the Markup for the Example

The following steps apply text markup to the `fourier_demo.m` file. When published to HTML, the output appears as shown in “Published Sample File After Adding Markup” on page 11-7.

For detailed information about each **Cell** menu option, see “Marking Up MATLAB Comments for Publishing” on page 11-17.

- 1 Open `fourier_demo.m` by running the following command:

```
edit(fullfile(matlabroot,'help','techdoc',...  
'matlab_env','examples','fourier_demo.m'))
```

To work with `fourier_demo.m` on your system, save the file to a folder for which you have write permission. In the example, the file is saved to `I:\my_matlab_files\my_mfiles\fourier_demo.m`.

- 2 Enable cell mode by selecting **Cell > Enable Cell Mode**.
- 3 Add an overall title and introduction:
 - a Select **Cell > Insert Text Markup > Document Title and Introduction**. MATLAB adds the following at the top of the file:

```
%% DOCUMENT TITLE  
% INTRODUCTORY TEXT
```

The double percent signs (%%) indicate the start of a new code cell. A single percent sign indicates the beginning of a comment line.

- b Replace **DOCUMENT TITLE** with **Square Waves from Sine Waves**.
- c Replace **% INTRODUCTORY TEXT** with one or more comments about the overall file, for example:

```
% The Fourier series expansion for a square-wave is  
% made up of a sum of odd harmonics, as shown here  
% using MATLAB(R).
```

The string “(R)” appears as a registered trademark symbol when you publish it.

- d On line 5, insert a blank line for better readability. Notice that the file now contains two cells. The first cell extends from line 6 to the top of the file; the second cell extends from line 6 to the bottom of the file. The cell break at line 6, splits the file into two cells.
- 4 On line 6, where the second cell begins (as indicated by %%), type a title for the cell: Add an Odd Harmonic and Plot It.

When you move from one cell to the next in the file, the highlighting in the file indicates the code cell containing the cursor.

```

1  %% Square Waves from Sine Waves
2  % The Fourier series expansion for a square-wave
3  % made up of a sum of odd harmonics, as shown
4  % using MATLAB(R) .
5
6  %% Add an Odd Harmonic and Plot It.
7  function fourier_demo
8      t = 0:.1:pi*4;
9      y = sin(t);
10     updatePlot(t,y);
11
12     % In each iteration of the for loop add an
13     % harmonic to y. As "k" increases, the out


```

- 5 To display the commented text as explanatory text, rather than MATLAB code, insert a cell break before the explanation. That is:
- a Place the cursor at line 12.
 - b Select **Cell > Insert Cell Break**.
- The code cell that begins on line 12, continues to the end of the `fourier_demo` function. If you insert a cell break anywhere within the code block, MATLAB inserts an implicit cell break at the end of a code block. A *code block* is the body of any programming control statement or function.

6 Remove the quotation marks around the `k` at line 14 and present it in italic instead:

- a** Delete the quotation marks.
- b** Select the letter `k`.
- c** Select **Cell > Insert Text Markup > Italic Text**.

Instead of appearing enclosed in quotation marks, the letter now appears as `_k_`.

- d** To see the effect, click Publish .

7 MATLAB publishes output generated by code immediately after the end of the cell that contains the code. Therefore, the current cell would cause MATLAB to publish the phrase `When k = n Then the plot is:` four times in succession. In addition, only the final plot generated by the `for` loop would be published.

To include every plot generated by the `for` loop, each preceded by the phrase `When k = n . . .`, create a cell within the `for` loop, as follows:


- a** Place the cursor at the end of line 17, after `for k = 3:2:9`.
- b** Select **Cell > Insert Cell Break**.

Now the current cell includes only the body of the `for` loop.

```

15 % approximates a square wave with increasing accuracy
16
17 for k = 3:2:9
18     %%
19     % Perform the following mathematical operation
20     % at each iteration:
21     y = y + sin(k*t)/k;
22
23     display(sprintf('When k = %.1f', k));
24     display('Then the plot is:');
25     updatePlot (t, y)
26 end
27
28 end
29
30 % Even though the approximations are constantly

```

- c To see the effect, click Publish .
- 8 Display equations with symbols and Greek characters (such as pi) using the LaTeX format. In this example, to output a comment containing a polished presentation of the equation, $y = y + \sin(k*t)/k$, use text markup as follows:
- Position the cursor at the end of the comment on line 20, % at each iteration.
 - Select **Cell > Insert Text Markup > LaTeX Equation**.

MATLAB inserts the following lines; the second line is a sample equation with text markup applied:

```

%
% $$e^{\pi i} + 1 = 0$$
%

```



The Editor highlights the sample equation, which is the text between the set of two dollar signs (\$\$).

- c Replace the sample equation with the following LaTeX equation:

$$y = y + \frac{\sin(k*t)}{k}$$

The three lines that display the LaTeX equation now appear as follows in the file:

```
%  
% $$y = y + \frac{\sin(k*t)}{k} $$  
%
```

- d To see the effect, click Publish .
- 9 Reduce the size of the figures in the output by editing the publish configuration for the file:
- a Select **File > Publish Configuration for fourier_demo > Edit Publish Configurations for fourier_demo.m**.
- The Edit Configurations dialog box opens.
- b In the column to the right of **Max image width (pixels)**, double-click **Inf**, and type the value 350.
 - c In the column to the right of **Max image height (pixels)**, double-click **Inf**, and type the value 350.
 - d Click **Save As**. The Save Publish Settings dialog box opens.
 - e In the **Settings name** field, type `small_images`, and then click **Save**.
 - f Click **Close**.
 - g To see the effect, click Publish .
- 10 To create a section header without including a cell break, follow these steps:
- a Position the cursor at the beginning of line 33, where the comment `% Even though the approximations are constantly appears`.
 - b Select **Cell > Insert Text Markup > Section Title without Cell Break**.
 - c Replace `SECTION TITLE` with `Note About Gibbs Phenomenon`.

d Delete line 34, where the comment `% DESCRIPTIVE TEXT` appears.

11 Select **File > Save File and Publish** `fourier_demo`.

When you publish the code to HTML, it appears in the MATLAB Web Browser, as shown in “Published Sample File After Adding Markup” on page 11-7.

By default, MATLAB stores the HTML document, `fourier_demo.html`, and the associated image files in an `/html` subfolder within the folder containing the source file, `fourier_demo.m`.

See “MATLAB Code After Text Markup” on page 11-15 for the resulting code.

MATLAB Code After Text Markup

After adding text markup, the `fourier_demo.m` file appears as follows. When you publish the file to HTML, it appears as shown in “Published Sample File After Adding Markup” on page 11-7.

```
%% Square Waves from Sine Waves
% The Fourier series expansion for a square-wave is
% made up of a sum of odd harmonics, as shown here
% using MATLAB(R).

%% Add an Odd Harmonic and Plot It
function fourier_demo
    t = 0:.1:pi*4;
    y = sin(t);
    updatePlot(t,y);

    %%
    % In each iteration of the for loop add an odd
    % harmonic to y. As _k_ increases, the output
    % approximates a square wave with increasing accuracy.

    for k = 3:2:9
        %%
        % Perform the following mathematical operation
        % at each iteration:
        %
```

```
    % $$ y = y + \frac{\sin(k*t)}{k} $$  
    %  
    y = y + sin(k*t)/k;  
  
    display(sprintf('When k = %.1f',k));  
    display('Then the plot is:');  
    updatePlot (t,y)  
end  
  
end  
  
%% Note About Gibbs Phenomenon  
% Even though the approximations are constantly  
% improving, they will never be exact because of the  
% Gibbs phenomenon, or ringing.  
  
function updatePlot(t,x)  
% Subfunction to update the plot  
    cla  
    plot(t,x)  
end
```

To open the marked up file in the Editor, instead of following the steps in “Adding the Markup for the Example” on page 11-10 run the following command:

```
edit(fullfile(matlabroot,'help','techdoc',...  
    'matlab_env','examples','fourier_demo2.m'))
```

To work with `fourier_demo2.m` on your system, save the file to `fourier_demo.m` in a folder for which you have write permission.

Marking Up MATLAB Comments for Publishing

In this section...

“Overview of Marking Up MATLAB Comments for Publishing” on page 11-18

“Creating Document Titles and Introductory Text for Publishing an Existing File” on page 11-19

“Specifying Preformatted Text in MATLAB Files for Publishing” on page 11-25

“Specifying Bulleted or Numbered Lists in MATLAB Files for Publishing” on page 11-27

“Specifying Graphics in MATLAB Files for Publishing” on page 11-30

“Using HTML Markup Tags in MATLAB Files for Publishing” on page 11-34

“Using LaTeX Markup for Publishing” on page 11-36

“Including Inline LaTeX Math Symbols in MATLAB Files for Publishing” on page 11-39

“Including Blocks of LaTeX Math Symbols in MATLAB Files for Publishing” on page 11-40

“Forcing a Snapshot of Output in MATLAB Files for Publishing” on page 11-42

“Including Bold, Italic, and Monospaced Text in MATLAB Files for Publishing” on page 11-43

“Including Trademarks in MATLAB Files for Publishing” on page 11-45

“Including Hyperlinks in MATLAB Files for Publishing” on page 11-46

“Cleaning Up Text Markup Before Publishing MATLAB Files” on page 11-54

“Summary of Markup for Publishing MATLAB Files” on page 11-57

Note Many examples in this section show the effects of publishing to HTML. For information on how to publish to HTML, see “Specifying Output Preferences for Publishing” on page 11-64.

Overview of Marking Up MATLAB Comments for Publishing

This section describes how to mark up comments in your MATLAB files so that when you publish the code, it appears polished, rather than as a text file of code. You can single-source your MATLAB code with documentation that describes what the code is doing.

You can mark up MATLAB comments in either of the following ways to specify the appearance of the file when you publish it:

- Use **Cell > Insert Text Markup** menu options to format the comments.

This method automatically inserts the text markup for you.

- Type the markup directly in the comments.

The markup symbols you type are the same as the text markup that results when you use the equivalent menu item. See “Summary of Markup for Publishing MATLAB Files ” on page 11-57 for details.

You can use text markup as you create a file, to mark up an existing file, or a combination of the two. When you use the **Cell** menu options, the Editor might insert more comment lines and other markup that you want.

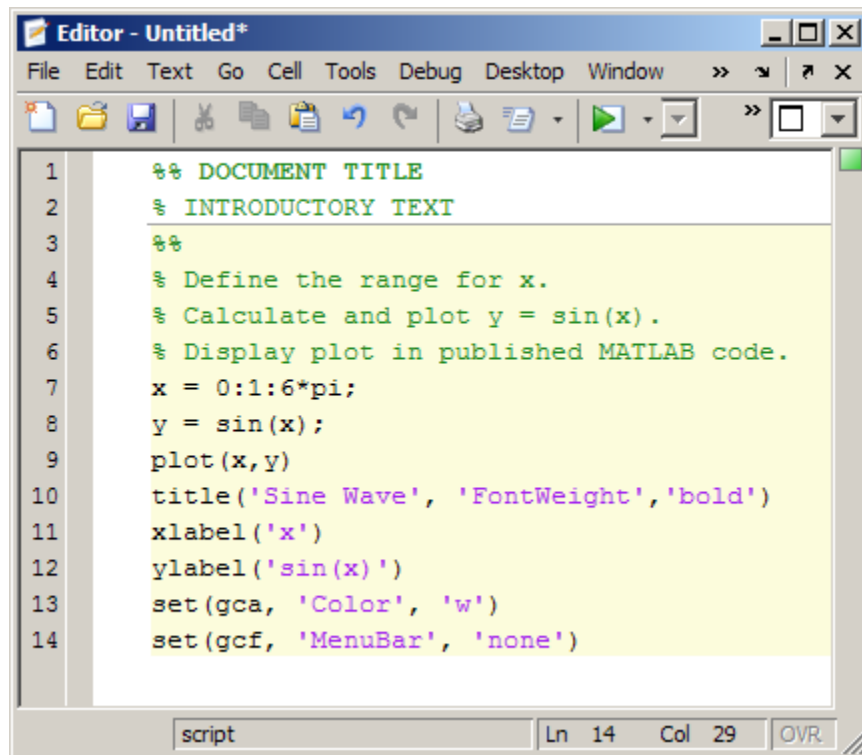
Several examples in the sections that follow use this file, `sine_wave.m`:

```
% Define the range for x.
% Calculate and plot y = sin(x).
% Display plot in published MATLAB code.
x = 0:1:6*pi;
y = sin(x);
plot(x,y)
title('Sine Wave', 'FontWeight','bold')
xlabel('x')
ylabel('sin(x)')
set(gca, 'Color', 'w')
set(gcf, 'MenuBar', 'none')
```


Creating Document Titles and Introductory Text for Publishing an Existing File

To specify a document title and introductory text for a MATLAB file, follow these steps:

- 1 In the Editor, position the cursor anywhere in the file.
- 2 Select **Cell > Insert Text Markup > Document Title and Introduction**. The first three lines of the file appear as shown in the following image.



The screenshot shows the MATLAB Editor window titled "Editor - Untitled*". The menu bar includes File, Edit, Text, Go, Cell, Tools, Debug, Desktop, and Window. The toolbar contains icons for file operations and execution. The script content is as follows:

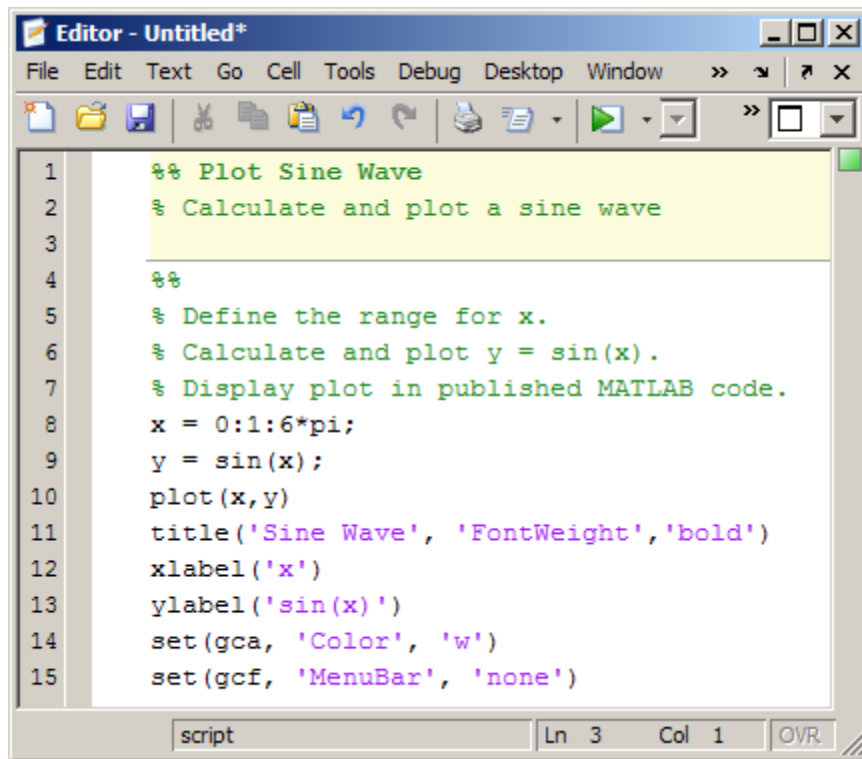
```
1 %% DOCUMENT TITLE
2 % INTRODUCTORY TEXT
3 %%
4 % Define the range for x.
5 % Calculate and plot y = sin(x).
6 % Display plot in published MATLAB code.
7 x = 0:1:6*pi;
8 y = sin(x);
9 plot(x,y)
10 title('Sine Wave', 'FontWeight','bold')
11 xlabel('x')
12 ylabel('sin(x)')
13 set(gca, 'Color', 'w')
14 set(gcf, 'MenuBar', 'none')
```

The status bar at the bottom indicates "script", "Ln 14", "Col 29", and "OVR.".

- 3 Replace **DOCUMENT TITLE** with the cell heading that you want to use; for example, **Plot Sine Wave**.

- 4 Replace INTRODUCTORY TEXT with text that introduces the file; for example, Calculate and plot a sine wave.
- 5 Insert a blank comment line to increase readability.

If you specify the example text suggested in the previous list, then the resulting file appears as follows.

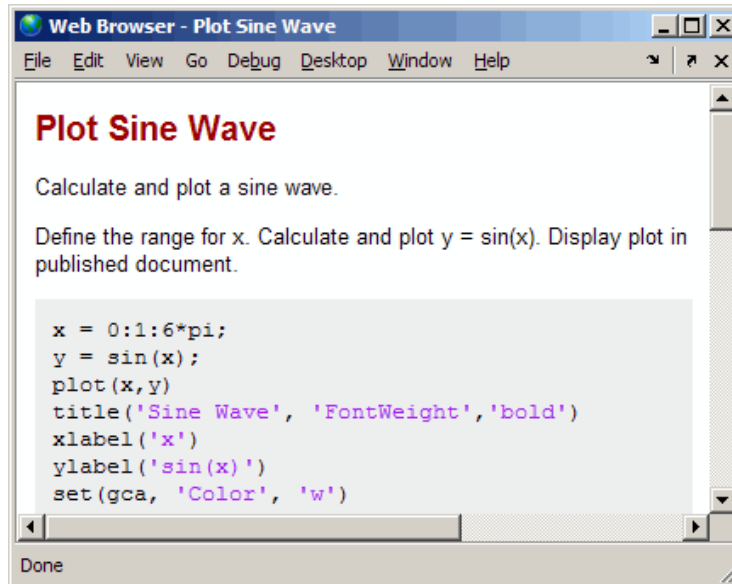


```
1 %% Plot Sine Wave
2 % Calculate and plot a sine wave
3
4 %%
5 % Define the range for x.
6 % Calculate and plot y = sin(x).
7 % Display plot in published MATLAB code.
8 x = 0:1:6*pi;
9 y = sin(x);
10 plot(x,y)
11 title('Sine Wave', 'FontWeight','bold')
12 xlabel('x')
13 ylabel('sin(x)')
14 set(gca, 'Color', 'w')
15 set(gcf, 'MenuBar', 'none')
```

Notice that a horizontal rule, which indicates a cell break, ends the title and introductory text. When you insert a document title and introduction, the Editor also adds a cell break in preparation for the first section within the file.

When you publish the file, the document title is displayed as a top-level heading (h1 in HTML), using a large size, bold font. The introductory text

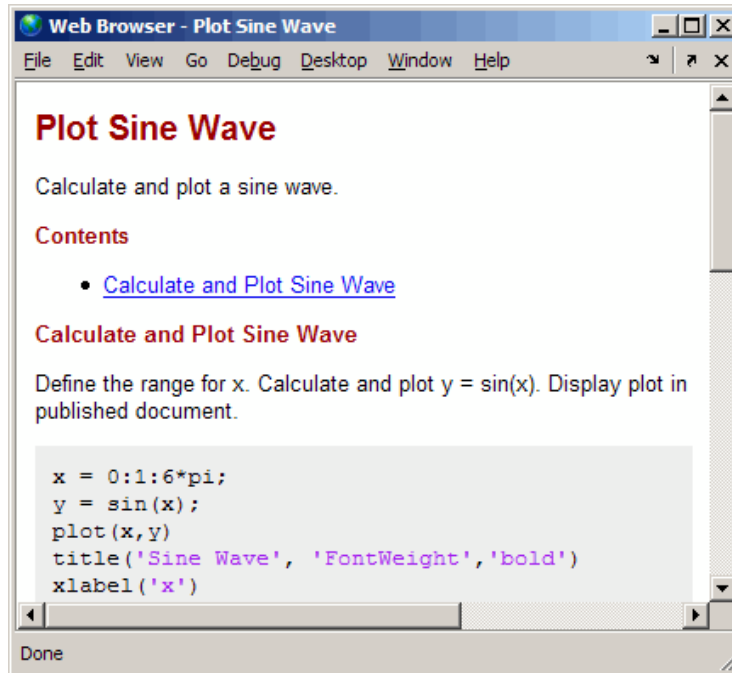
appears polished. The following figure shows the file published to HTML and presented in the MATLAB Web Browser.



Specifying a Title for the New Section that the Editor Inserts with the Document Title

When you follow the steps in the previous section, “Creating Document Titles and Introductory Text for Publishing an Existing File” on page 11-19, the first cell, demarcated in the Editor with the horizontal rule followed by a line with double percent signs (%%), does not appear when you publish the file. It is not evident because the section does not have a title.

To provide a title for the section, insert text after the double percent signs on line 4—for example, Calculate and Plot Sine Wave. When you republish the file to HTML, it appears in the MATLAB Web Browser as shown in the following image. Notice that MATLAB automatically inserts the Contents heading and the link to the section when you publish the file to HTML.



The file now has a document title, introductory text, and a first section. You can add more sections, as described in “Creating New Section Titles” on page 11-22.

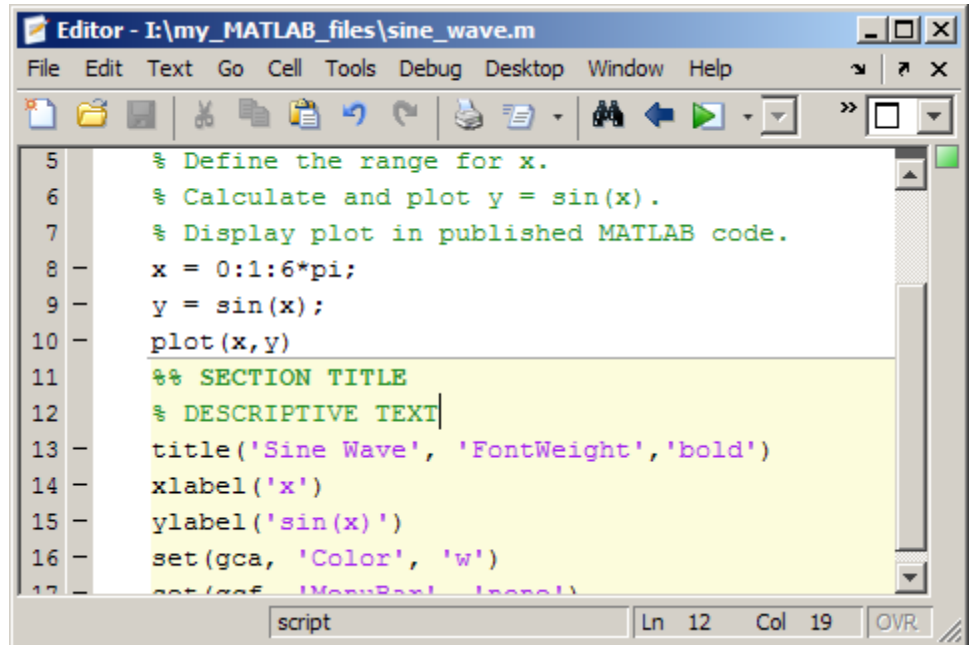
Note You can add any comments in the lines immediately following the title. However, if you want the title to appear as the overall document title, you cannot add any other text before the next cell (a line starting with %%).

Creating New Section Titles

To insert a new section title and descriptive text within a MATLAB file, follow these steps:

- 1 Position the cursor where you want to insert a new cell—before the title function shown in the example, for instance.

- 2 Select **Cell > Insert Text Markup > Section Title with Cell Break**.
The file appears as follows.



The screenshot shows the MATLAB Editor window for a file named 'sine_wave.m'. The code is as follows:

```

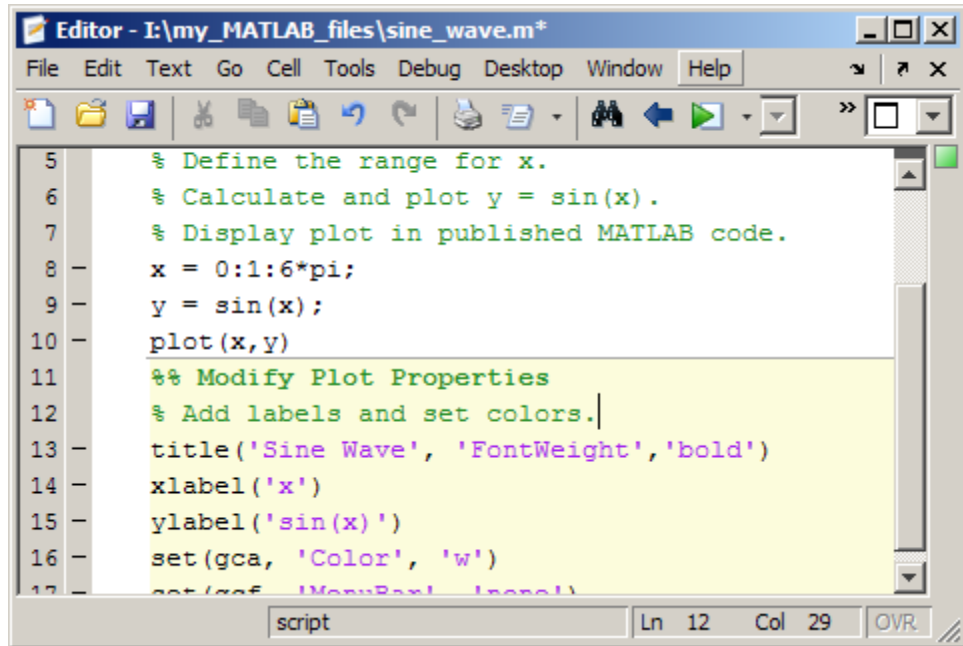
5      % Define the range for x.
6      % Calculate and plot y = sin(x).
7      % Display plot in published MATLAB code.
8      x = 0:1:6*pi;
9      y = sin(x);
10     plot(x,y)
11     %% SECTION TITLE
12     % DESCRIPTIVE TEXT
13     title('Sine Wave', 'FontWeight', 'bold')
14     xlabel('x')
15     ylabel('sin(x)')
16     set(gca, 'Color', 'w')
17     set(gcf, 'MenuBar', 'none')

```

The editor interface includes a menu bar (File, Edit, Text, Go, Cell, Tools, Debug, Desktop, Window, Help), a toolbar with various icons, and a status bar at the bottom showing 'script', 'Ln 12', 'Col 19', and 'OVR'.

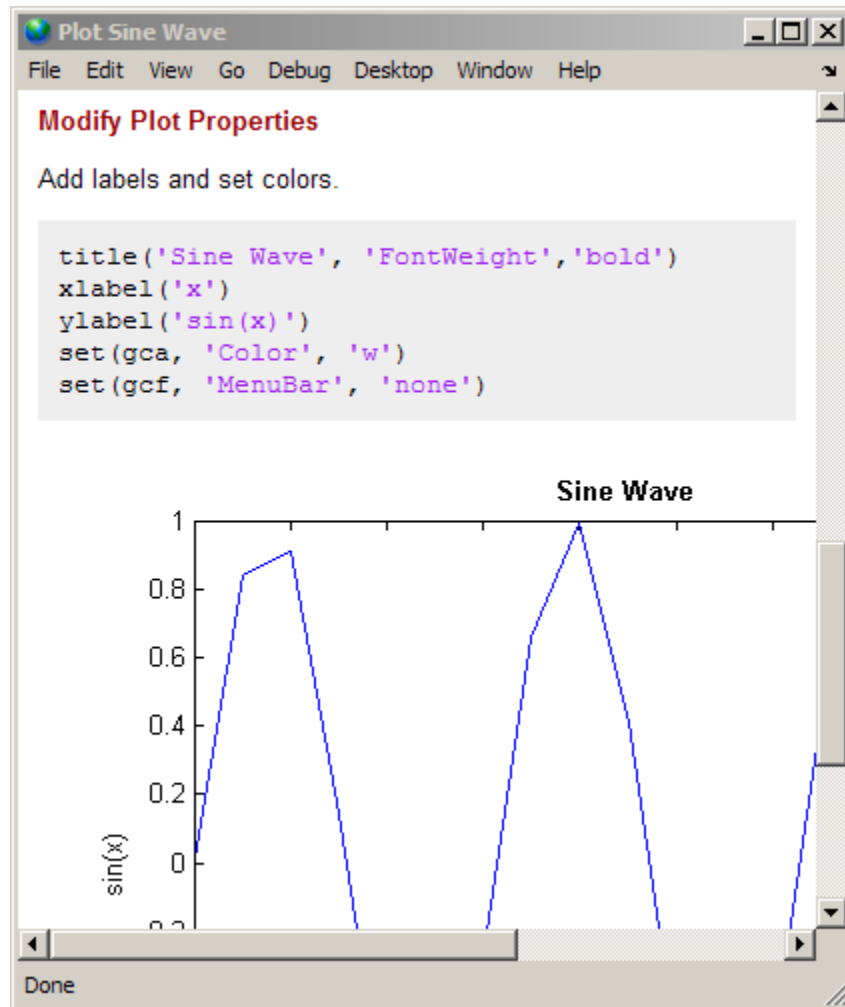
- 3 Replace **SECTION TITLE** with your title—**Modify Plot Properties**, for example.
- 4 Replace **DESCRIPTIVE TEXT** with text that describes the cell—Add labels and set colors., for example.

If you specify the example text suggested in the previous list and “Specifying a Title for the New Section that the Editor Inserts with the Document Title” on page 11-21, then the resulting file appears as follows.



```
5 % Define the range for x.
6 % Calculate and plot y = sin(x).
7 % Display plot in published MATLAB code.
8 x = 0:1:6*pi;
9 y = sin(x);
10 plot(x,y)
11 %% Modify Plot Properties
12 % Add labels and set colors.
13 title('Sine Wave', 'FontWeight', 'bold')
14 xlabel('x')
15 ylabel('sin(x)')
16 set(gca, 'Color', 'w')
17 set(gcf, 'MenuBar', 'none')
```

When you publish the file to HTML, the section title appears as a heading, using a medium size, bold font. Comments appear as polished text. The following figure shows the results when you publish the updated `sine_wave.m` file to HTML output. Notice that the `Modify Plot Properties` heading is appears as an `h2`.



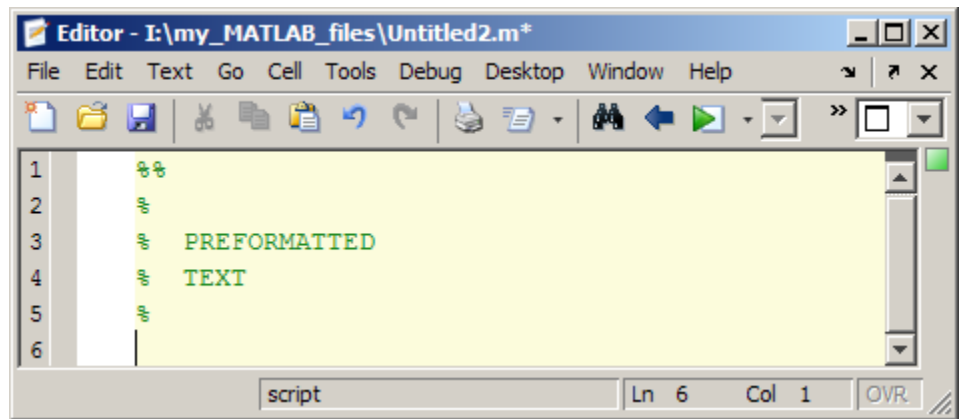
Specifying Preformatted Text in MATLAB Files for Publishing

MATLAB software enables you to specify preformatted text in a MATLAB file. Preformatted text appears in monospace font, maintains white space, and does not wrap long lines.

To insert preformatted text, follow these steps:

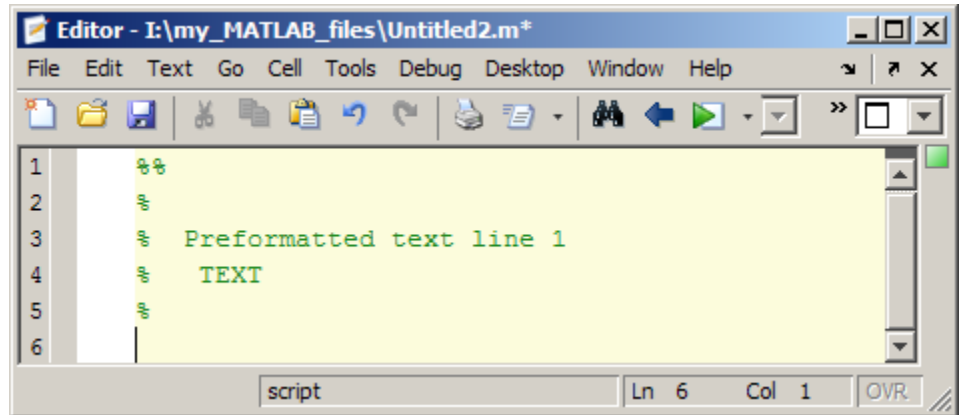
- 1** Position the cursor within the file where you want to insert preformatted text.
- 2** Select **Cell > Insert Text Markup > Preformatted Text**.

Five lines of text are inserted.

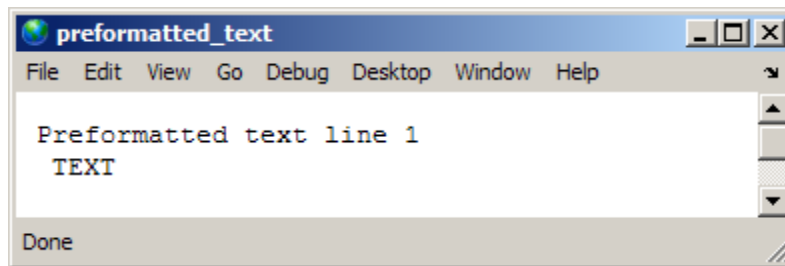


- 3** Being careful not to delete the two blank spaces before the word PREFORMATTED, replace the words PREFORMATTED and TEXT with your text. Your text can include tabs, spaces, and additional comment lines. For example:
 - a** Replace PREFORMATTED with Preformatted text line 1.
 - b** Insert a tab before TEXT on line 4.

The resulting comments appear as follows.

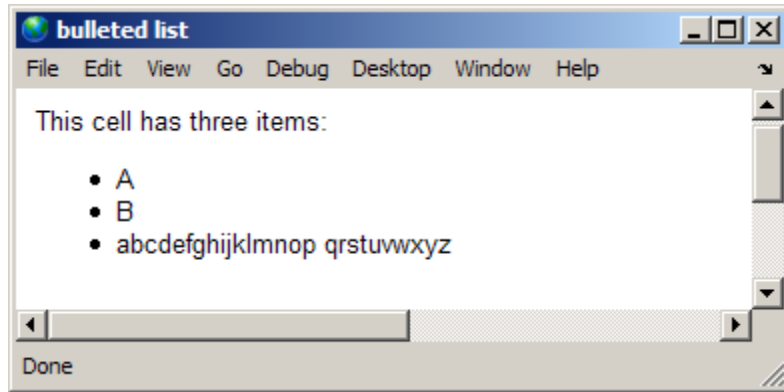


If you save the file to `preformatted_text.m` and then publish the file to HTML, the output appears as shown in the following figure.



Specifying Bulleted or Numbered Lists in MATLAB Files for Publishing

The following steps describe how to specify text markup for a bulleted or numbered list, so that it appears as shown when you publish it. .

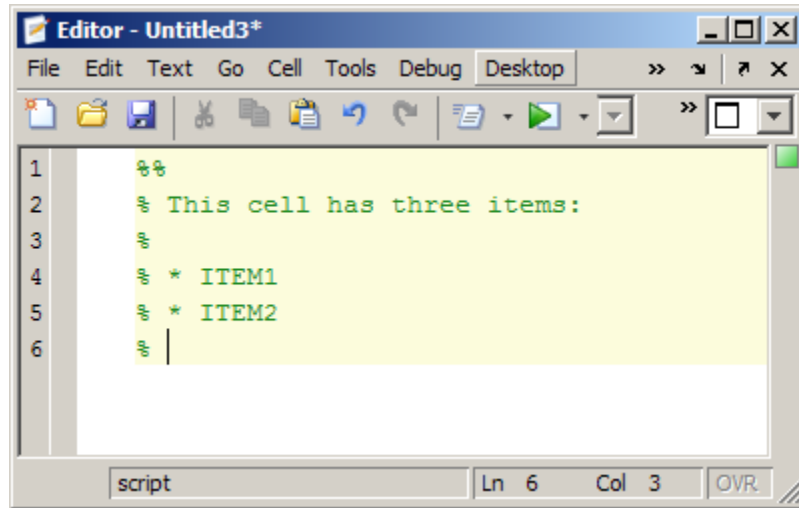


- 1 Position the cursor at the end of the line that precedes the location where you want to add a list. For example, if your file contains the following lines, position the cursor after the colon:

```
%%  
% This cell has three items:
```

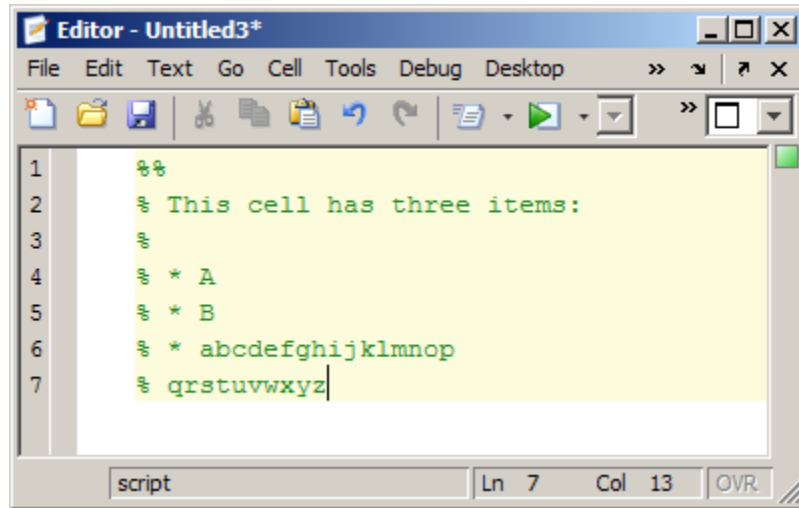
- 2 Select **Cell > Insert Text Markup > Bulleted List** or **Cell > Insert Text Markup > Numbered List**, depending on the type of list you want.

MATLAB adds four lines of comments to the file. The following figure shows the result when you insert a bulleted list.



If you insert a numbered list, the text markup is the same, except a number sign (#) indicates a numbered list item.

- 3** Replace the sample text, ITEM1 and ITEM2, with your text. For example, replace ITEM1 with A and replace ITEM2 with B.
- 4** To create a multiline list item, break the line as desired, but do not insert the list item symbol (* or #) before the second line. For example, to insert the alphabet as a multiline list item, breaking the line at the letter p, type the alphabet as shown in the following figure.



Notice that the third list item breaks over two comment lines in the source, yet maintains the appearance of a list when published (as shown at the beginning of this section).

Specifying Graphics in MATLAB Files for Publishing

You can insert text markup to publish an image that the MATLAB code did not generate. This is shown in the following example. (By default, the publishing includes images generated by the MATLAB code.)

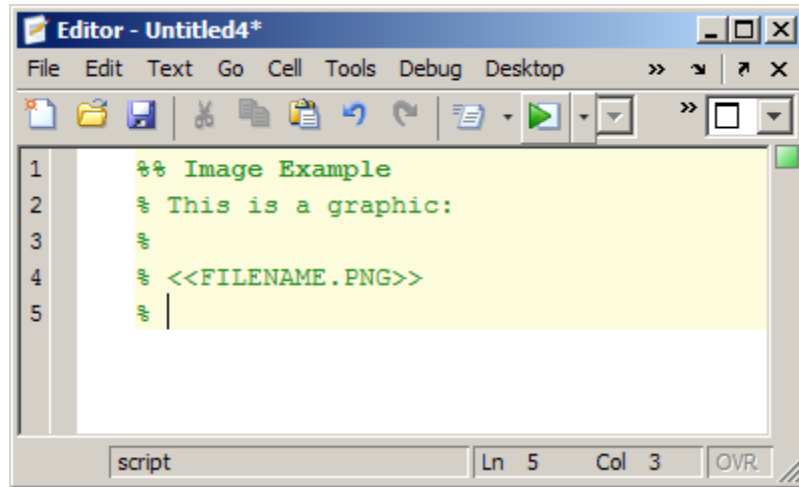
- 1 Position the cursor where you want to add a graphic. For example, if your file contains the following lines, position the cursor after the colon:

```

%% Image Example
% This is a graphic:

```

- 2 Select **Cell > Insert Text Markup > Image**. MATLAB adds text markup, as shown in the following figure.



- 3 Replace FILENAME.PNG, with the file name of the graphic you want to insert, relative to the folder where MATLAB publishes the file.

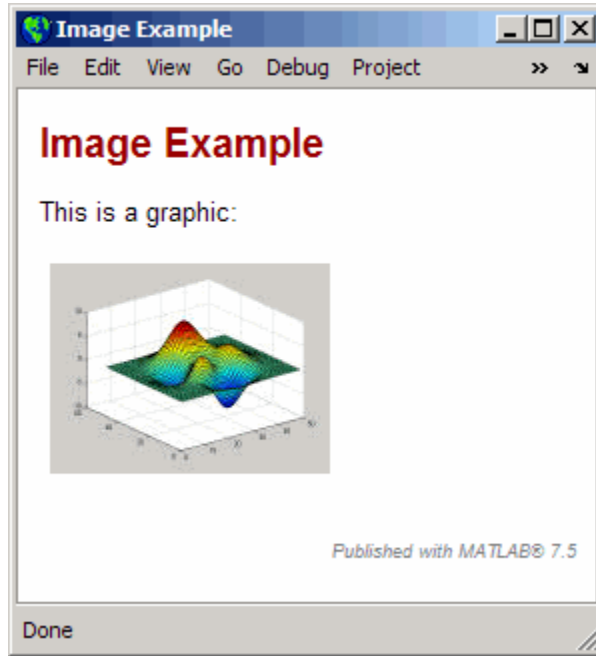
For example, if you want to include the graphic, `surfpeaks.png`, and it is in the folder into which MATLAB publishes the MATLAB code file, then replace FILENAME.PNG with `surfpeaks.png`. See “Creating the `surfpeaks.png` Image” on page 11-33.

By default, MATLAB publishes the file to an `/html` subfolder of the folder containing the file. You can change this folder, referred to as the output folder, by changing the publish configuration settings, as described in “Specifying Output Preferences for Publishing” on page 11-64.

If the graphic is not in the folder to which you publish the file, then you must specify the location of the graphic file as a relative path from the location of the file that results from publishing. The following table summarizes how to specify such graphics, assuming the folder containing the MATLAB code is `I:\my_MATLAB_files`.

Folder Where publish Saves the Output	Image File Location	How to Specify the Image in the MATLAB Comment
<code>I:\my_MATLAB_files/html</code>	<code>I:\my_MATLAB_files/html</code>	<code>% <<surfpeaks.jpg>></code>
<code>I:\my_MATLAB_files/doc</code>	<code>I:\my_MATLAB_files/images</code>	<code>% <<../images/surfpeaks.jpg>></code>

When you publish the file to HTML, the output appears as shown in the following figure.



Valid Image Types for Output File Formats

The type of images you can include when you publish depends on the output type of that document as indicated in the table that follows. For greatest compatibility, MathWorks recommends using the default image format for each output type.

Output File Format	Types of Images You Can Include
doc	Any format that your installed version of Microsoft Office supports.
html	Any format publishes successfully. Ensure that the tools you use to view and process the output files can display the output format you specify.

Output File Format	Types of Images You Can Include
latex	Any format publishes successfully. Ensure that the tools you use to view and process the output files can display the output format you specify.
pdf	bmp and jpg .
ppt	Any format that your installed version of Microsoft Office supports.
xml	Any format publishes successfully. Ensure that the tools you use to view and process the output files can display the output format you specify.

Creating the surfpeaks.png Image

To create the surfpeaks.png image used in the preceding example, follow these steps:

1 Create an html subfolder in the folder where the file that references the graphic is located.

2 Enter the following in the Command Window:

```
>> surf(peaks)
```

A Figure window opens and displays the surfpeaks figure.

3 Save the figure as surfpeaks.jpg in the html subfolder that you created in step 1.

Note Unless you reduce the size of surfpeaks.jpg, it will appear larger than shown in the previous example.

Using HTML Markup Tags in MATLAB Files for Publishing

You can use the **Cell** menu to insert HTML code into your MATLAB file. When you do so, the Editor inserts HTML code for a one-column, two-row table. You can use the inserted code as a guideline for inserting other HTML code.

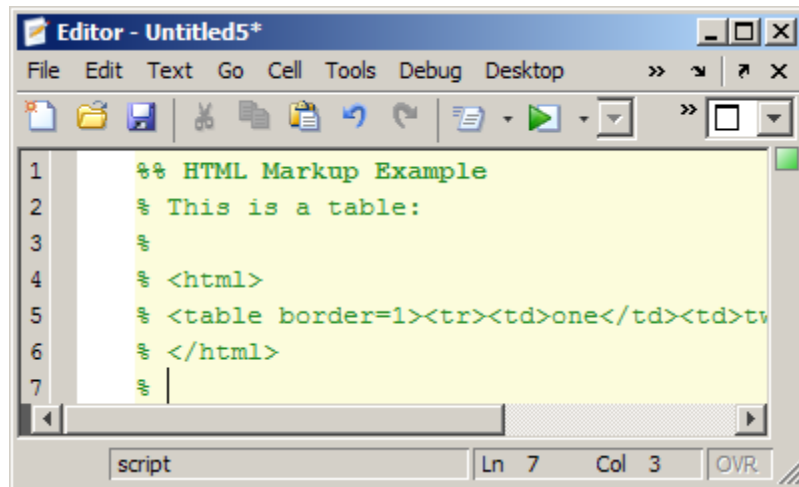
Note When you insert text markup for HTML code, the HTML code publishes only when the specified output file format is HTML. For example, if you add HTML markup, but then specify LaTeX as the output file format, publishing excludes the text enclosed within the HTML markup. See “Specifying Output Preferences for Publishing” on page 11-64 for information on specifying the output file format.

To insert the text markup for HTML code, follow these steps:

- 1** Position the cursor at the end of the comment that precedes the location where you want to insert HTML code. For example, if the file contains the following lines, position the cursor after the colon:

```
%% HTML Markup Example  
% This is a table:
```

- 2** Select **Cell > Insert Text Markup > HTML Markup**. MATLAB adds HTML markup, as shown in the following figure.



The screenshot shows the MATLAB Editor window titled "Editor - Untitled5*". The menu bar includes File, Edit, Text, Go, Cell, Tools, Debug, and Desktop. The toolbar contains icons for file operations and execution. The editor area shows the following code:

```
1 %% HTML Markup Example
2 % This is a table:
3 %
4 % <html>
5 % <table border=1><tr><td>one</td><td>two</td></tr></table>
6 % </html>
7 %
```

The status bar at the bottom indicates "script", "Ln 7", "Col 3", and "OVR".

- 3 Edit the inserted HTML code to specify the HTML code that you want to use.

If you publish the file to HTML and leave the inserted HTML code as is, MATLAB creates a single-row table with two columns. The table contains the values one and two, as shown in the following figure.



Using LaTeX Markup for Publishing

You can use the **Cell** menu to insert LaTeX code. You can use the inserted code as a guideline for inserting other LaTeX code.

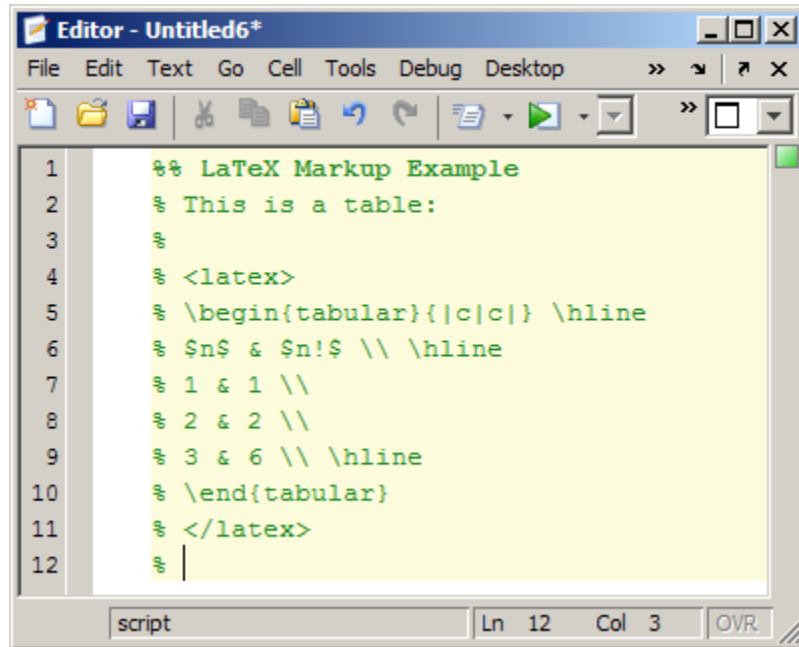
Note When you insert text markup for LaTeX code, that code publishes only when the specified output file format is LaTeX. For example, if you add LaTeX markup, but specify HTML as the output file format, publishing excludes the code enclosed within the LaTeX markup. See “Specifying Output Preferences for Publishing” on page 11-64 for information on specifying the output file format.

To insert the text markup for LaTeX code, follow these steps:

- 1** Position the cursor at the end of the comment that precedes the location where you want to insert LaTeX code. For example, if the file contains the following lines, position the cursor after the colon:

```
%% LaTeX Markup Example  
% This is a table:
```

- 2** Select **Cell > Insert Text Markup > LaTeX Markup**. MATLAB adds LaTeX markup, as shown in the following figure.

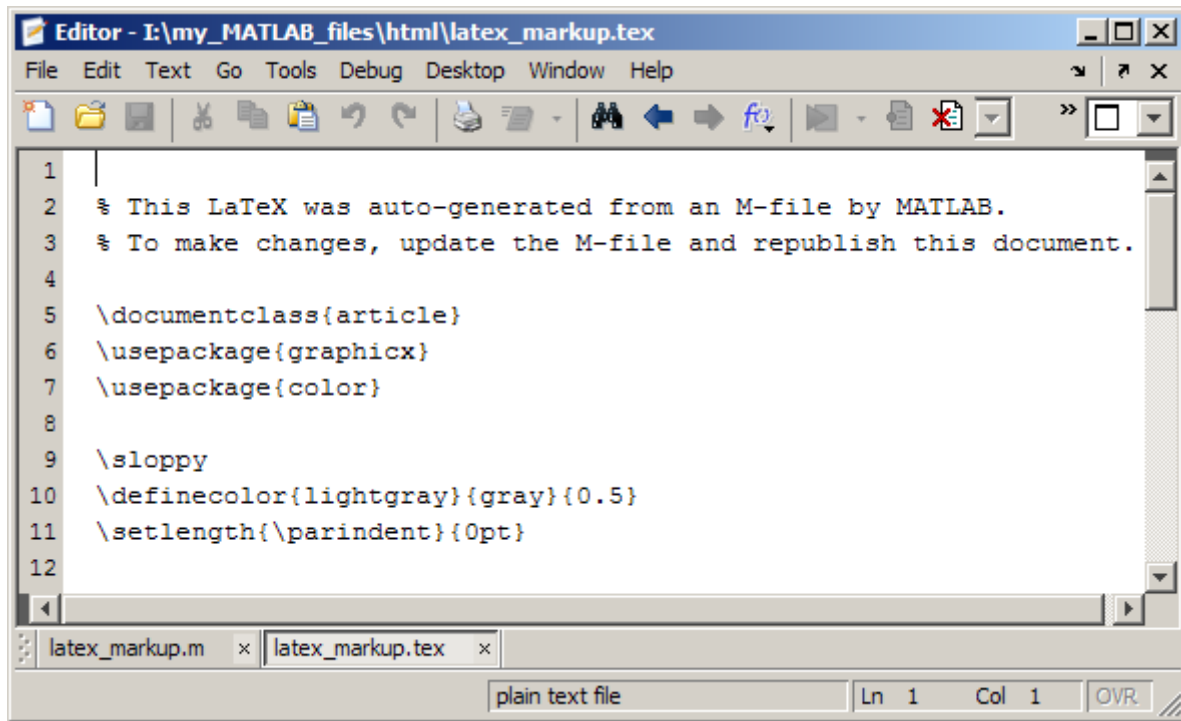
The image shows a screenshot of the MATLAB Editor window titled "Editor - Untitled6*". The window has a menu bar with "File", "Edit", "Text", "Go", "Cell", "Tools", "Debug", and "Desktop". Below the menu bar is a toolbar with various icons for file operations and editing. The main editor area has a light yellow background and contains the following text:

```
1 %% LaTeX Markup Example
2 % This is a table:
3 %
4 % <latex>
5 % \begin{tabular}{|c|c|} \hline
6 % $n$ & $n!$ \\ \hline
7 % 1 & 1 \\
8 % 2 & 2 \\
9 % 3 & 6 \\ \hline
10 % \end{tabular}
11 % </latex>
12 %
```

The status bar at the bottom of the window shows "script", "Ln 12", "Col 3", and "OVR".

- 3 Edit the inserted LaTeX code to specify the LaTeX code that you want to use.

Suppose you publish the file to LaTeX, and leave the inserted markup text as is. In that case, the Editor opens a new file with the LaTeX code, as shown in the following figure. (See “Creating a Publish Configuration for a MATLAB File” on page 11-66 for information on specifying LaTeX as the output format.)



The screenshot shows a MATLAB Editor window titled "Editor - I:\my_MATLAB_files\html\latex_markup.tex". The window contains the following LaTeX code:

```
1 |
2 | % This LaTeX was auto-generated from an M-file by MATLAB.
3 | % To make changes, update the M-file and republish this document.
4 |
5 | \documentclass{article}
6 | \usepackage{graphicx}
7 | \usepackage{color}
8 |
9 | \sloppy
10 | \definecolor{lightgray}{gray}{0.5}
11 | \setlength{\parindent}{0pt}
12 |
```

The status bar at the bottom indicates "plain text file", "Ln 1 Col 1", and "OVR".

If you compile the published LaTeX code, it appears as follows.

LaTeX Markup Example

This is a table:

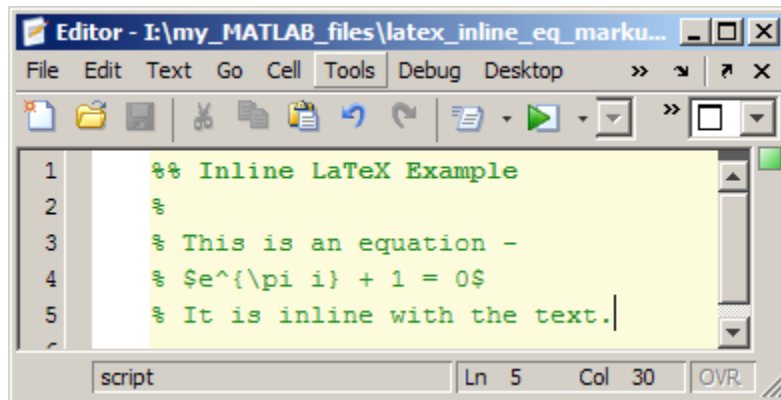
n	$n!$
1	1
2	2
3	6

Including Inline LaTeX Math Symbols in MATLAB Files for Publishing

You can make LaTeX math symbols appear inline when you publish a MATLAB file to any format, except Microsoft PowerPoint®. For Microsoft PowerPoint output, consider “Using LaTeX Markup for Publishing” on page 11-36 instead.

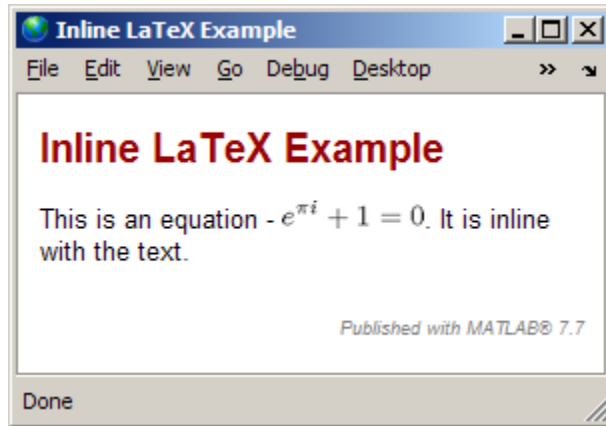
To make a LaTeX math symbol appear inline, enclose the string within dollar sign symbols (\$) within a formatted comment block.

For example, suppose your file appears as follows:



```
1 %% Inline LaTeX Example
2 %
3 % This is an equation -
4 %  $e^{i\pi} + 1 = 0$ 
5 % It is inline with the text.
```

When you publish the file to HTML, it appears as in the image that follows. Notice that the LaTeX math symbols are inline with the rest of the text:



For a list of symbols you can display, and the character sequence to create them, see the MATLAB String property.

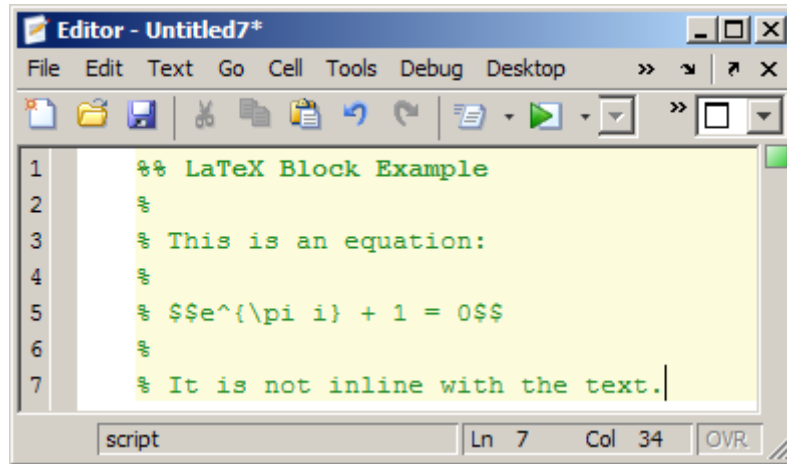
Including Blocks of LaTeX Math Symbols in MATLAB Files for Publishing

You can use the **Cell** menu to insert LaTeX symbols in blocks offset from the main comment text. You can use the inserted code as a guideline for inserting other LaTeX code.

- 1 Position the cursor before the line where you want to add an equation or symbols. For example, if your file contains the following lines, position the cursor after the colon on the end of the third line:

```
%% LaTeX Block Example
%
% This is an equation:
% It is not inline with the text.
```

- 2 Select **Cell > Insert Text Markup > LaTeX Equation** to insert the sample equation markup.



```

1 %% LaTeX Block Example
2 %
3 % This is an equation:
4 %
5 % $$e^{\pi i} + 1 = 0$$
6 %
7 % It is not inline with the text.

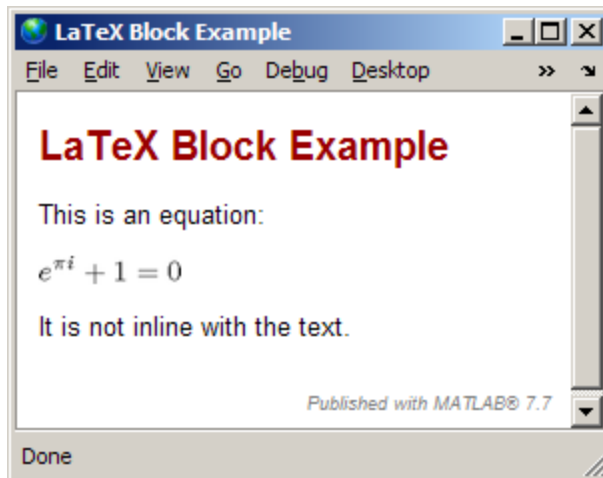
```

The screenshot shows the MATLAB Editor window titled "Editor - Untitled7*". The menu bar includes File, Edit, Text, Go, Cell, Tools, Debug, and Desktop. The toolbar contains icons for file operations and execution. The script content is as follows:

- 3 Replace the inserted sample markup $e^{\pi i} + 1 = 0$ with the LaTeX math symbols that you want.

For a list of symbols you can display, and the character sequence to create them, see the MATLAB String property.

Publishing the code to HTML, and leaving the inserted sample text markup as is, results in this:



Forcing a Snapshot of Output in MATLAB Files for Publishing

You can use the **Cell** menu to insert code that forces a snapshot of output, such as a figure. This is useful, for example, if you have a **for** loop that generates numerous figures and you want to publish them all, after the **for** loop end statement.

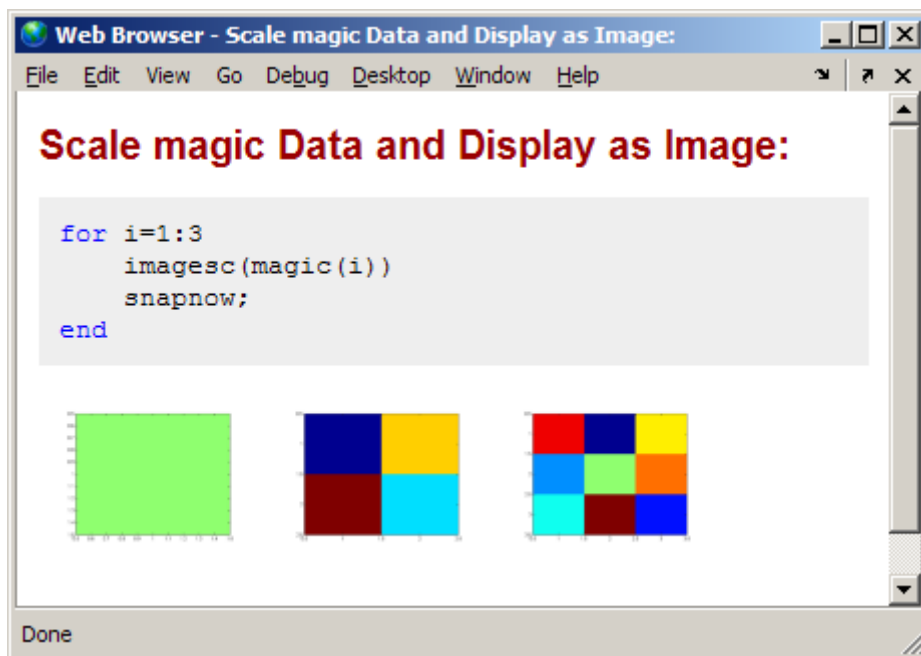
- 1 Position the cursor at the end of the line where you want to force a snapshot of the output. For example, if your file contains the following lines, position the cursor after the line containing the `imagesc` function:

```
%% Scale magic Data and Display as Image:  
  
for i=1:3  
    imagesc(magic(i))  
end
```

- 2 Select **Cell > Insert Text Markup > Force Snapshot** to insert the `snapshot` function:

```
%% Scale magic Data and Display as Image:  
  
for i=1:3  
    imagesc(magic(i))  
    snapshot;  
end
```

- 3 If you publish the file to HTML, it resembles the following figure. The images in your HTML will be larger than shown in the figure. To resize images generated by MATLAB code, use the **Max image width** and **Max image height Publish settings**, as described in “Specifying Output Preferences for Publishing” on page 11-64.



Including Bold, Italic, and Monospaced Text in MATLAB Files for Publishing

You can mark selected strings in the MATLAB comments so that they display in bold, italic, or monospaced text when you publish the file. The following sections provide instructions.

Marking Up Existing Comments with Font Formats

To mark up existing comments, follow these steps:

- 1 Within a comment, select text that you want to be bold, italic, or monospaced.
- 2 Select **Cell > Insert Text Markup**, and then select **Bold Text**, **Italic Text**, or **Monospaced Text**.

Inserting New Comments with Font Formats

To insert sample text that you will replace with your new comment text, follow these steps:

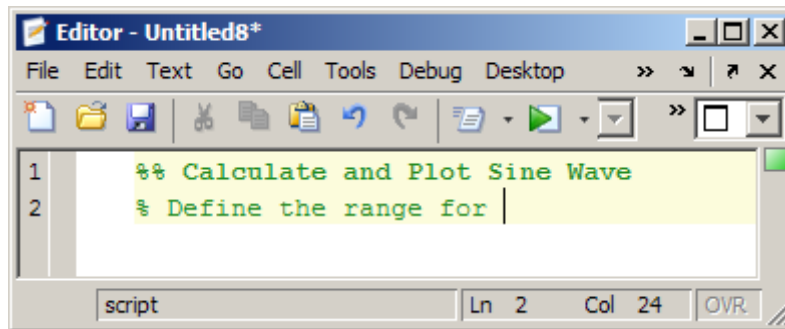
- 1 Select **Cell > Insert Text Markup**, and then select **Bold Text**, **Italic Text**, or **Monospaced Text**.
- 2 Replace the inserted text with the text that you want.

When the Editor inserts sample text, the inserted text appears as follows:

```
% *BOLD TEXT*  
% _ITALIC TEXT_  
% |MONOSPACED TEXT|
```

Example of Font Formats

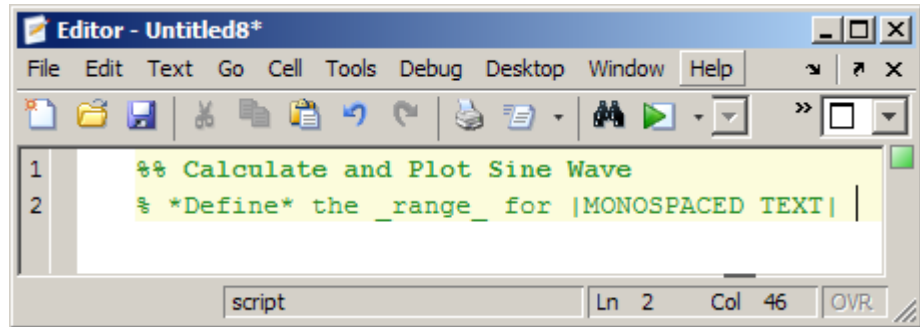
Suppose your file appears as follows.



Mark up the comments as follows:

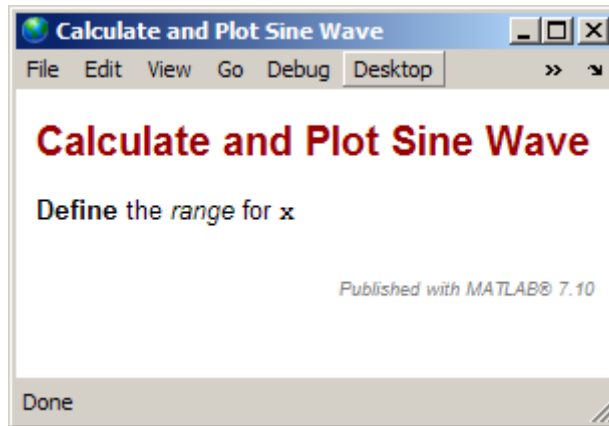
- 1 Select the word **Define**, and then select **Cell > Insert Text Markup > Bold Text**.
- 2 Select the word **range**, and then select **Cell > Insert Text Markup > Italic Text**.
- 3 Position the cursor after the word **for**, insert a space, and then select **Cell > Insert Text Markup > Monospaced Text**.

The file appears as follows.



- 4 Replace |MONOSPACED TEXT| with |x|.

If you publish the file to HTML, the output appears as shown in the following figure.



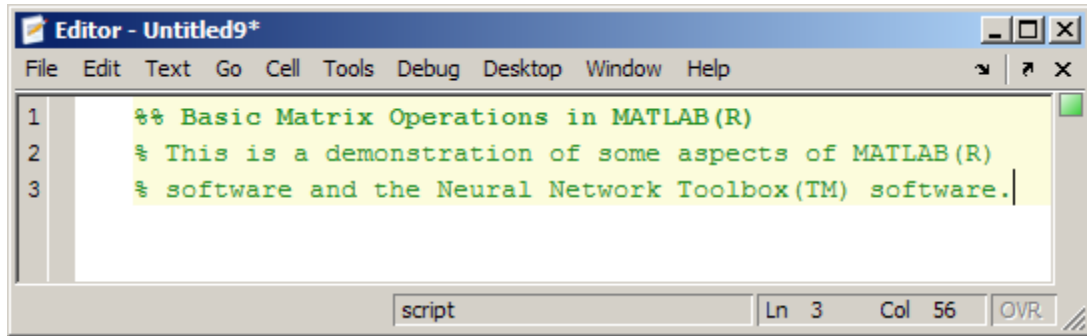
Including Trademarks in MATLAB Files for Publishing

If the comments in your MATLAB file include trademarked terms, you can include text to produce a trademark symbol (™) or registered trademark symbol (®) in the output.

- To produce the trademark symbol, enter (™) in a MATLAB comment.

- To produce the registered trademark symbol, enter (R) in a MATLAB comment.

For example, suppose you enter lines in a file as shown in the following image.



If you publish the file to HTML, it appears as follows in the MATLAB Web Browser.



Including Hyperlinks in MATLAB Files for Publishing

You can insert dynamic or static hyperlinks within a MATLAB comment, and then publish the MATLAB file to HTML or XML. You can also publish static hyperlinked text to Microsoft Word.

When you specify a *dynamic* hyperlink, MATLAB evaluates the hyperlinked code when someone clicks it in the output. This practice is useful when, for example, you want to point the reader to MATLAB code or documentation or you want the link to run code. When you specify a *static* hyperlink to a Web location, you specify a complete URL within the code. This is useful when you want to point the reader to a location on the Web. For both static and dynamic hyperlinks, the output contains active hyperlinks.

You can include or exclude the URL from a static hyperlink. Consider including the URL, for example, when you anticipate that readers of your output file might view it in printed form, and therefore need the URL. Consider excluding the URL, when you are confident that readers will view your output online and therefore be able to click the hyperlink.

This section includes the following topics:

- “Inserting Static Hyperlinks and Publishing URLs” on page 11-47
- “Inserting Static Hyperlinks Without Publishing URLs” on page 11-48
- “Inserting Dynamic Hyperlinks” on page 11-49
- “Effect of Copying Hyperlinked Text from the MATLAB Command Window” on page 11-53

Inserting Static Hyperlinks and Publishing URLs

You can insert a URL, such as `www.mathworks.com`, that you know at the time you write the file by following these steps:

- 1** Within a comment, position the cursor where you want to insert the hyperlinked text. For example, suppose you want to specify a link to more information about a topic and you have the following comment within the file:

```
%%  
% For more information, see our Web site:
```

Position your cursor after the colon (:).

- 2** Select **Cell > Insert Text Markup > Hyperlinked Text**.

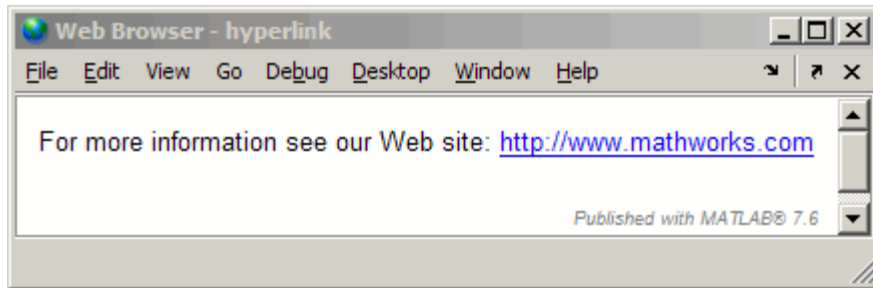
The Editor inserts the following:

```
<http://www.mathworks.com MathWorks>
```

3 Replace `www.mathworks.com` with the URL you want to use.

4 Delete the string, `MathWorks`.

When you publish the file to HTML, the results resemble the following figure (except the URL in this image is still `http://www.mathworks.com`).



Inserting Static Hyperlinks Without Publishing URLs

To insert hyperlinked text without a printed URL, follow these steps:

1 Within a comment, position the cursor where you want to insert the hyperlinked text. For example, suppose you want to specify a hyperlink to the MathWorks Web site and you have the following lines within your file:

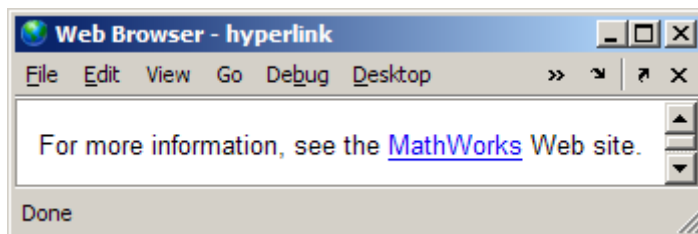
```
%%
% For more information, see the MathWorks Web site.
```

Select the text you want to replace with a hyperlink. For example, select "MathWorks"

2 Select **Cell > Insert Text Markup > Hyperlinked Text**. The Editor replaces the selected text with the following:

```
<http://www.mathworks.com MathWorks>
```

If you publish the file to HTML, the results are as shown in the following figure.



- 3 Replace `www.mathworks.com` with the URL that you want to use.
- 4 Replace `MathWorks` with the text that you want to appear as the hyperlinked text.

Inserting Dynamic Hyperlinks

You can insert a hyperlink which MATLAB evaluates at the time a reader clicks that link. You implement these links using `matlab:` syntax.

You cannot insert dynamic links in Microsoft Word files.

Note A reader must have MATLAB running for dynamic links to work.

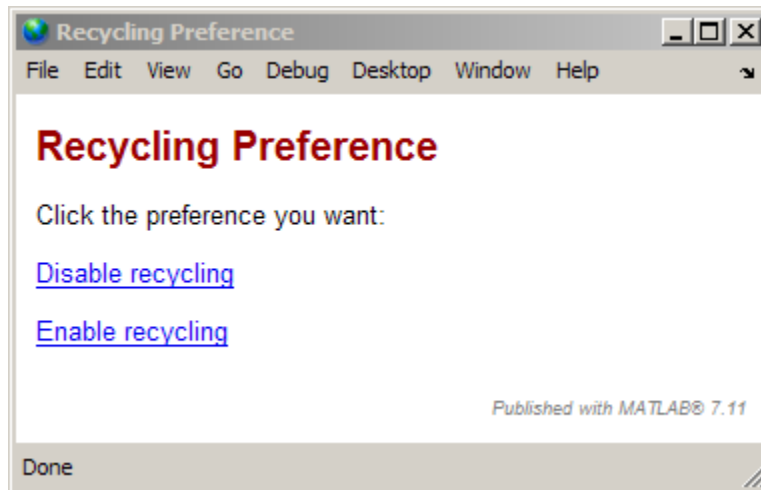
The following sections provide examples:

- “Inserting a Dynamic Link To Run Code” on page 11-49
- “Inserting a Dynamic Link to a File” on page 11-50
- “Inserting a Dynamic Link to a MATLAB Function Reference Page” on page 11-52

Inserting a Dynamic Link To Run Code. You can specify a dynamic hyperlink to run code when a user clicks the hyperlink. For example, the following `matlab:` syntax creates hyperlinks in the output, which when clicked either enable or disable recycling:

```
%% Recycling Preference
% Click the preference you want:
%
% <matlab:recycle('off') Disable recycling>
% <matlab:recycle('on') Enable recycling>
```

When you publish the file to HTML, the results resemble the following figure:



When you click one of the hyperlinks, MATLAB sets the `recycle` command accordingly. After clicking a hyperlink, run `recycle` in the Command Window to confirm the setting is as you expect.

Inserting a Dynamic Link to a File. You can specify a link to a file that you know is in your readers' `matlabroot`. You do not need to know where each reader installed MATLAB. To insert a dynamic link to a file, follow these steps:

- 1 Within a comment, position the cursor where you want to insert the link. For example, suppose you want to specify a link to the MATLAB help topic for the `publish` function. Also suppose you have the following comment within the file:

```
%%
% See the code
```



```
% for the publish function.
```

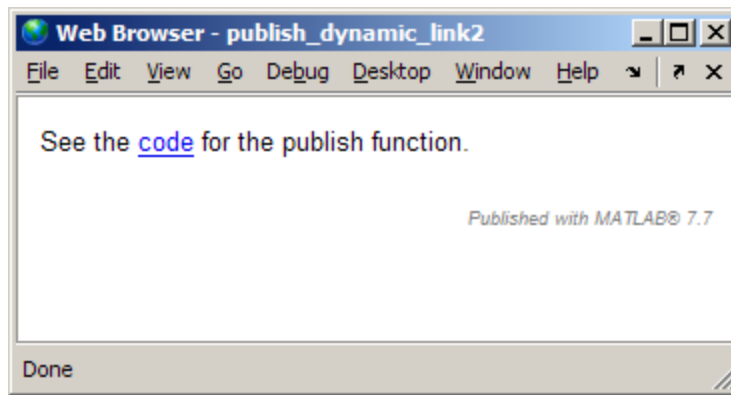
2 Replace the word `code` with the following markup:

```
<matlab:web(fullfile(matlabroot,'toolbox','matlab','codetools','publish.m')) code>
```

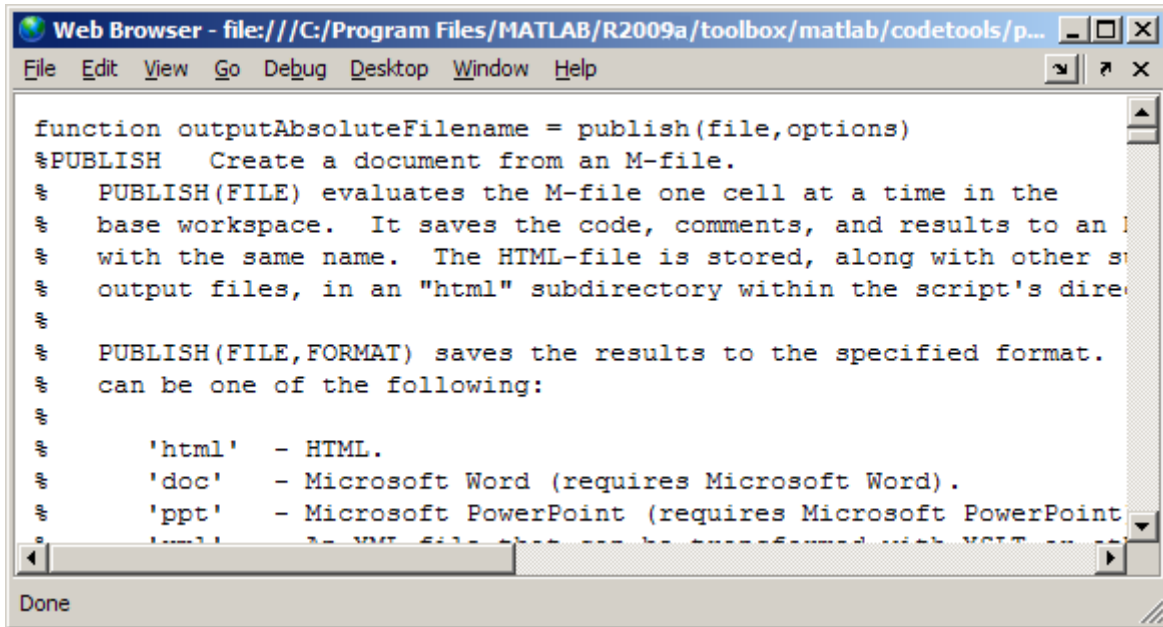
The resulting code now appears as follows:

```
%%
% See the <matlab:web(fullfile(matlabroot,'toolbox','matlab','codetools','publish.m')) code>
% for the publish function.
```

When you publish the file to HTML, the results resemble the following figure.



When you click the `code` link, the MATLAB Web browser opens and displays the code for the `publish` function. On the reader's system, MATLAB issues the command (although the command does not appear in the reader's Command window). Notice in the Web browser title bar that the `matlabroot` specification in the file resolves to the reader's installation folder for MATLAB.



```

function outputAbsoluteFilename = publish(file,options)
%PUBLISH Create a document from an M-file.
% PUBLISH(FILE) evaluates the M-file one cell at a time in the
% base workspace. It saves the code, comments, and results to an
% with the same name. The HTML-file is stored, along with other
% output files, in an "html" subdirectory within the script's dire
%
% PUBLISH(FILE,FORMAT) saves the results to the specified format.
% can be one of the following:
%
% 'html' - HTML.
% 'doc' - Microsoft Word (requires Microsoft Word).
% 'ppt' - Microsoft PowerPoint (requires Microsoft PowerPoint)
% 'html' - An HTML file that can be transformed with XSLT into

```

Inserting a Dynamic Link to a MATLAB Function Reference Page.

You can specify a link to a MATLAB function reference page using `matlab:` syntax. For example, suppose your readers have MATLAB installed and running. To provide a link to the `publish` reference page, follow these steps:

- 1 Within a comment, position the cursor where you want to insert the hyperlinked text. For example, suppose you want to specify a link to the MATLAB help topic for the `publish` function. Furthermore, suppose you have the following comment within the file:

```

%%
% See the help for the publish function.

```

- 2 Replace the word `publish` with the following markup:

```

<matlab:doc('publish') publish>

```

The resulting file code now appears as follows:

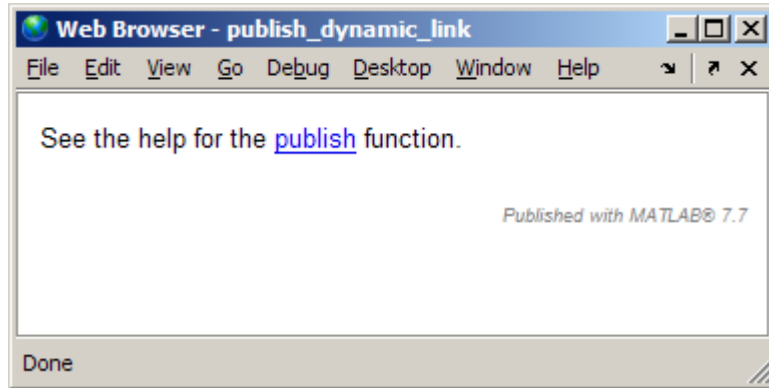
```

%%

```

```
% See the help for the <matlab:doc('publish') publish> function.
```

When you publish the file to HTML, the results resemble the following figure.



When you click the `publish` hyperlink, the MATLAB help browser opens and displays the reference page for the `publish` function. On the reader's system, MATLAB issues the command (although the command does not appear in their Command window).

Effect of Copying Hyperlinked Text from the MATLAB Command Window

If you copy statements that display hyperlinked text from the MATLAB Command Window to a file and then publish that file, results might not be as you expect. When you do so, the output shows the code rather than the hyperlink.

For example, suppose you enter the following code in the Command Window:

```
disp('<a href="http://www.mathworks.com">Link to MathWorks</a>')
```

When you press **Return**, the Command Window displays a link to the MathWorks Web site:

[Link to MathWorks](http://www.mathworks.com)

However, if you include the preceding `disp` statement in a file that you publish, the HTML tag and the included text appears in the output, rather than a link:

```
disp(' <a href="http://www.mathworks.com">Link to MathWorks</a>')
```

Instead, use one of the methods described in these sections:

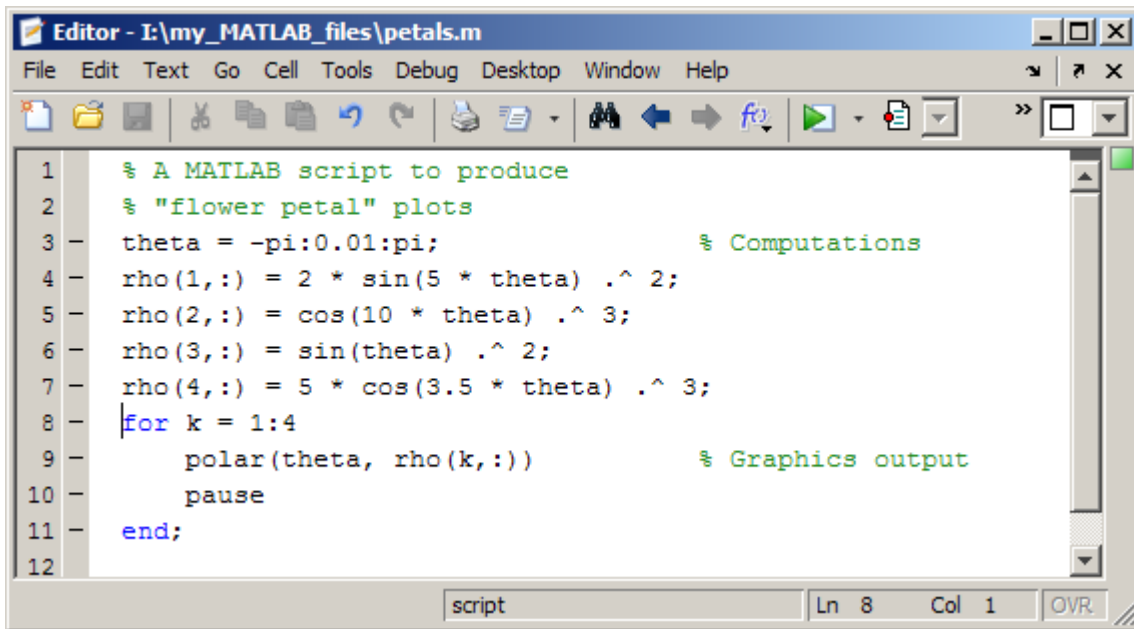
- “Inserting Static Hyperlinks and Publishing URLs” on page 11-47
- “Inserting Static Hyperlinks Without Publishing URLs” on page 11-48
- “Inserting Dynamic Hyperlinks” on page 11-49

Cleaning Up Text Markup Before Publishing MATLAB Files

When you insert text markup into an existing file using the **Cell** menu options, sometimes more comment lines than you need are inserted. You can adjust the inserted comments as needed for your purposes. If you delete blank comment lines that the **Cell** menu options insert there might be unintended consequences, however. See “Specifying Preformatted Text in MATLAB Files for Publishing” on page 11-25 for details.

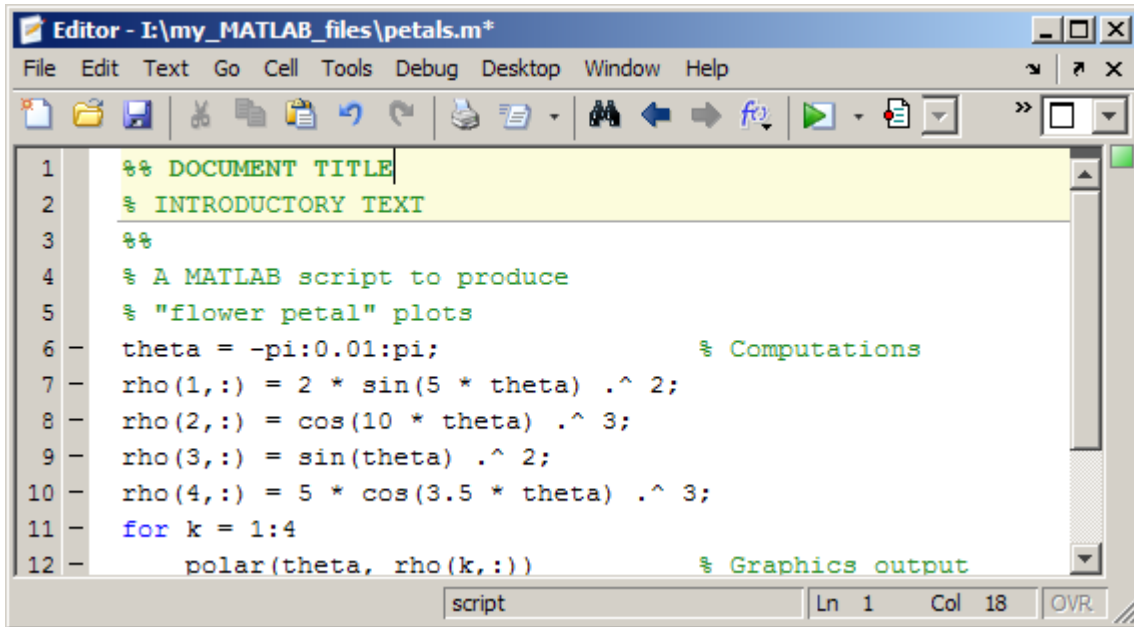
The following example shows how you can use **Cell** menu options with an existing file.

Suppose a file currently appears in the Editor as shown in the following image.



```
1  % A MATLAB script to produce
2  % "flower petal" plots
3  - theta = -pi:0.01:pi;           % Computations
4  - rho(1,:) = 2 * sin(5 * theta) .^ 2;
5  - rho(2,:) = cos(10 * theta) .^ 3;
6  - rho(3,:) = sin(theta) .^ 2;
7  - rho(4,:) = 5 * cos(3.5 * theta) .^ 3;
8  - for k = 1:4
9  -     polar(theta, rho(k,:))     % Graphics output
10 -     pause
11 - end;
12
```

If you position the cursor anywhere within the file and select **Cell > Insert Text Markup > Document Title and Introduction**, the file looks like the following.



The screenshot shows the MATLAB Editor window with the following code:

```
1 %% DOCUMENT TITLE
2 % INTRODUCTORY TEXT
3 %%
4 % A MATLAB script to produce
5 % "flower petal" plots
6 - theta = -pi:0.01:pi; % Computations
7 - rho(1,:) = 2 * sin(5 * theta) .^ 2;
8 - rho(2,:) = cos(10 * theta) .^ 3;
9 - rho(3,:) = sin(theta) .^ 2;
10 - rho(4,:) = 5 * cos(3.5 * theta) .^ 3;
11 - for k = 1:4
12 -     polar(theta, rho(k,:)) % Graphics output
```

The status bar at the bottom indicates the file is a 'script' and the cursor is at 'Ln 1 Col 18'.

The file already contains a comment with introductory text, so you can delete the % INTRODUCTORY TEXT line and the double percent sign (%%) line. When you do so, the code appears as follows.

```

Editor - I:\my_MATLAB_files\petals.m*
File Edit Text Go Cell Tools Debug Desktop Window Help
1 %% DOCUMENT TITLE
2 | A MATLAB script to produce
3 | "flower petal" plots
4 - theta = -pi:0.01:pi; % Computations
5 - rho(1,:) = 2 * sin(5 * theta) .^ 2;
6 - rho(2,:) = cos(10 * theta) .^ 3;
7 - rho(3,:) = sin(theta) .^ 2;
8 - rho(4,:) = 5 * cos(3.5 * theta) .^ 3;
9 - for k = 1:4
10 -     polar(theta, rho(k,:)) % Graphics output
11 -     pause
12 - end;
script Ln 2 Col 1 OVR

```

Summary of Markup for Publishing MATLAB Files

The following table provides a summary of the text markup that you can type into a file to achieve the same results as using the **Cell > Insert Text Markup** menu options. These tables are useful if you are not using the MATLAB Editor, or if you do not want to use menus to apply markup. For a description of the **Cell** menu options, see “Marking Up MATLAB Comments for Publishing” on page 11-17.

Note The blank comment lines preceding and following some items, such as bulleted lists, are required.

Result in Output	Example of Corresponding File Markup
Document title and introduction	<code>%% DOCUMENT TITLE % INTRODUCTORY TEXT</code>
Section title and description	<code>%% SECTION TITLE % DESCRIPTIVE TEXT</code>
Section title without cell break	<code>%% SECTION TITLE % DESCRIPTIVE TEXT</code>
Bold text	<code>% *BOLD TEXT*</code>
Italic text	<code>% _ITALIC TEXT_</code>
Monospaced text	<code>% MONOSPACED TEXT </code>
Hyperlinked text	<code>% <http://www.mathworks.com MathWorks></code>
Trademark symbol	<code>% TEXT(TM)</code>
Registered trademark symbol	<code>% TEXT(R)</code>
Code to force a snapshot of the current output	<code>snapshot;</code>
Image	<code>% % <<FILENAME.PNG>> % %</code>
Bulleted list	<code>% % * ITEM % * ITEM %</code>

Result in Output	Example of Corresponding File Markup
Numbered list	<pre data-bbox="605 331 742 447">% % # ITEM1 % # ITEM2 %</pre>
HTML markup	<pre data-bbox="605 501 1006 713">% % <html> % <table border=1><tr> % <td>one</td> % <td>two</td></tr></table> % </html> %</pre>
LaTeX markup	<pre data-bbox="605 765 1006 1043">% % <latex> % \begin{tabular}{ r r} % \hline \$n\$&\$n!\$\\ % \\hline 1&1\\ 2&2\\ 3&6\\ % \\hline % \end{tabular} % </latex> %</pre>
LaTeX equation	<pre data-bbox="605 1090 946 1116">% \$\$e^{\pi i} + 1 = 0\$\$</pre>
Preformatted text block	<pre data-bbox="605 1199 1194 1347">% % This is a block of preformatted text. % The first line of commented text must % begin with two blank spaces. %</pre>

Marking Up MATLAB Code for Publishing

In this section...

“Overview of Marking Up MATLAB Code for Publishing” on page 11-60

“Specifying the Display of Code Output” on page 11-60

“Example of Marking Up Code” on page 11-60

Overview of Marking Up MATLAB Code for Publishing

This section describes ways to control how output that MATLAB generates when it publishes executable MATLAB code. For example, you can direct MATLAB to include the last, or all plots generated by a `for` loop. You can interweave comments, code, and code output throughout your output document to draw your readers’ attention to certain areas of interest.

Specifying the Display of Code Output

The method you use to specify how output displays in the document is the same method you use to specify document titles and section headers; namely the cell break (`%%`). When you insert a cell break into a file, it directs MATLAB to publish the code and output contained in the cells created by the break. Because the entire file is a cell, inserting a cell break results in the file containing two cells. The first cells is the one above the cell break, and the second is the one below the cell break. The examples in the remaining topics demonstrate how you can use this behavior to control the output produced by MATLAB code.

Example of Marking Up Code

This section provides an example to demonstrate how a MATLAB file appears when published. It demonstrates how the published example file appears before and after cell breaks are added.

Sample MATLAB File Before Inserting Mark Up in Code

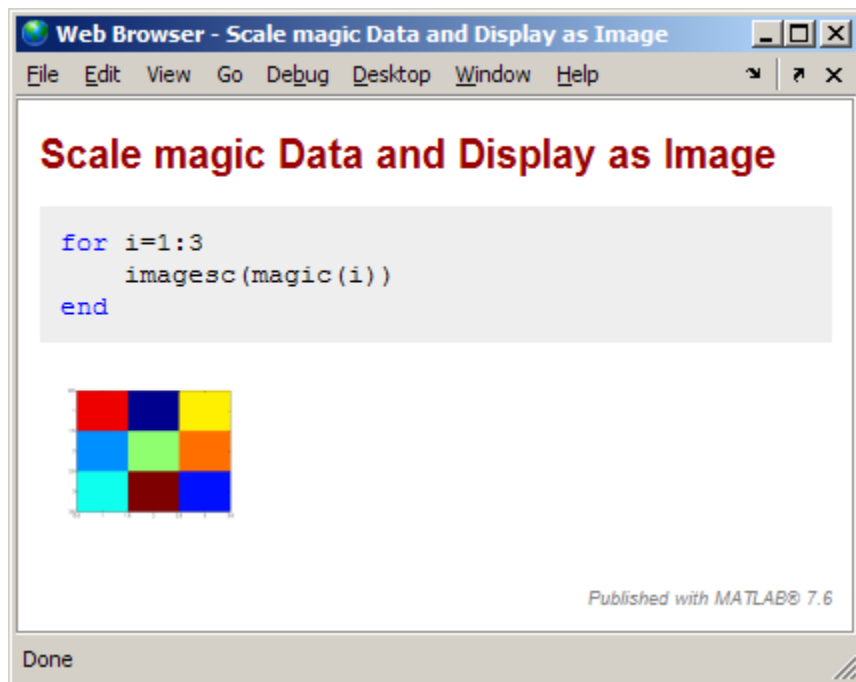
Suppose your file contains the following code:

```
%% Scale magic Data and Display as Image
```

```
for i=1:3
    imagesc(magic(i))
end
```

The following image illustrates how the code presented appears when you publish it to HTML. The plot in the figure is smaller than it appears if you publish the code using factory default settings. For information on setting publishing properties for images, see “Specifying Output Preferences for Publishing” on page 11-64.

Notice that the output displays the plot after the end of the `for` loop and that only the last plot generated by the code displays.



Sample MATLAB File After Inserting Cell Breaks in Code

By placing cell breaks within a loop, you can display the output that the MATLAB code generates when iterating a loop.

To include the plot generated by each iteration of the loop in the output file, insert a cell break after the opening for statement. Position the cursor at the end of the first line of the `for` loop, and then select **Cell > Insert Cell Break**.

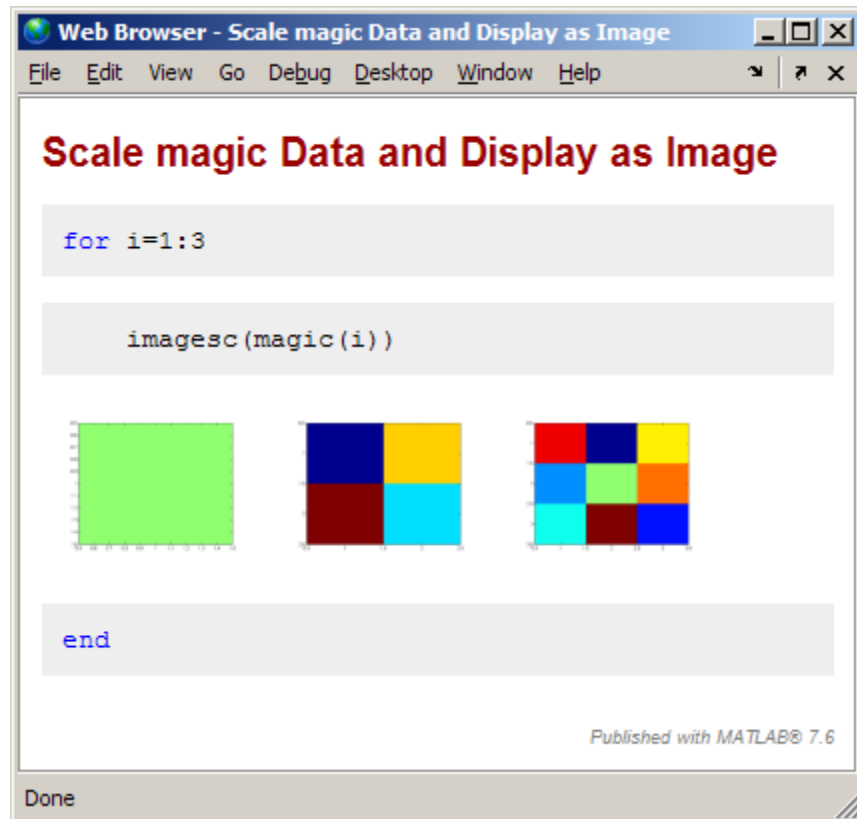
The code now appears like this:

```
%% Scale magic Data and Display as Image

for i=1:3
    %%
    imagesc(magic(i))
end
```

Now when you publish the code to HTML, it appears as follows. The plots in the figure are smaller than they appear if you publish the code using factory default settings. For information on setting publishing properties for images, see “Specifying Output Preferences for Publishing” on page 11-64.

Notice that the output file displays the plot within the `for` loop code. You can also use text markup for similar results with figures. See “Marking Up MATLAB Code for Publishing” on page 11-60 for details.



Specifying Output Preferences for Publishing

In this section...

- “About Publishing Configurations” on page 11-64
- “Creating a Publish Configuration for a MATLAB File” on page 11-66
- “Running an Existing Publish Configuration” on page 11-92
- “Creating Multiple Publish Configurations for a File” on page 11-93
- “About the publish_configurations.m File” on page 11-104
- “Finding Publish Configurations” on page 11-105
- “Removing Publish Configurations” on page 11-105
- “Reassociating and Renaming Publish Configurations” on page 11-105

About Publishing Configurations

Once you have marked up MATLAB code for publishing, as described in “Marking Up MATLAB Comments for Publishing” on page 11-17 and “Marking Up MATLAB Code for Publishing” on page 11-60 you are ready to publish it. The easiest method for publishing a MATLAB code file is to use the factory default publishing configuration. This method is appropriate if your code requires no input arguments and you want to publish to HTML. However, if your code requires input arguments, or if you want to specify preferences for publishing (such as the output folder, output format, image format, and so on), specify a custom configuration.

Publishing MATLAB Files Using No Input Arguments and Factory Default Settings

To publish a MATLAB script or MATLAB function file that requires no input arguments:

- 1 Open the file in the Editor.
- 2 Click the Publish button  on the Editor toolbar.

By default, the Editor publishes the file using factory default settings. Factory default settings specify that the output file format is HTML, that the code is evaluated and included in the output file, and so on.

If the file is not in a folder on the search path or in the current folder, a dialog box opens and presents you with options that allow you to publish the file. Either change the current folder to the folder containing the file, or add the folder containing the file to the MATLAB search path.

If the file has unsaved changes, publishing it from the Editor automatically saves the changes before publishing.

Using Publish Configurations to Publish MATLAB Files Using Input Arguments or Custom Settings

Using a publish configuration, you can specify custom settings, including input arguments for a MATLAB function file in the Editor. You can associate multiple publish configurations with a file for different publish settings, input arguments, or both. MATLAB saves the configuration between sessions.

For example, the function `collatzplot_new.m`, which computes and plots the Collatz sequence for any given positive integer, requires you to specify the integer as an input value. You cannot simply publish `collatzplot_new.m` because the input value is not defined. A publish configuration enables you to publish `collatzplot_new(specific value)`.

You can also use publish configurations to provide preparatory or setup information before publishing a file, whether it takes input arguments or not.

Note Publish configurations use the base MATLAB workspace. Therefore, a value that you assign to a variable in a publish configuration overwrites the value for that variable (assuming that it currently exists) in the base workspace.

Function Alternative to Publishing

From the Command Window, execute the `publish` function to run the file and publish the results. See the `publish` function reference page for options you can set.

Creating a Publish Configuration for a MATLAB File

- “Specifying File Input Using a Publish Configuration” on page 11-66
- “Specifying Publish Configuration Settings” on page 11-70
- “Specifying Values for the Publish Settings Property Table” on page 11-74
- “Creating a Template for Typical Publish Settings” on page 11-90

Specifying File Input Using a Publish Configuration

Follow these steps to create a publish configuration for a MATLAB code file in the Editor. The example in this section shows how to create and use a publish configuration to specify input arguments to a MATLAB function file.

These steps specify Editor toolbar buttons, but you can also use equivalent items in the **File** menu.

- 1 Open the file that you want to publish in the Editor. This example uses the code that follows. This code is like the `sine_wave.m` file, after it has been marked up as described in “Marking Up MATLAB Comments for Publishing” on page 11-17, but it is slightly altered to make it a MATLAB function file. Save the code as `sine_wave_f.m`

```
%% Plot Sine Wave
% Calculate and plot a sine wave.

%% Calculate and Plot Sine Wave
% Calculate and plot |y = sin(x)|.


function sine_wave_f(x)

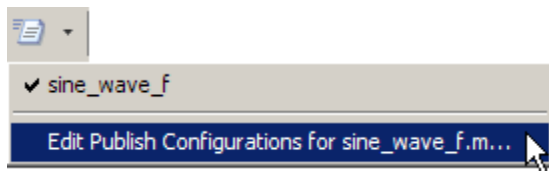
y = sin(x);
plot(x,y)
```



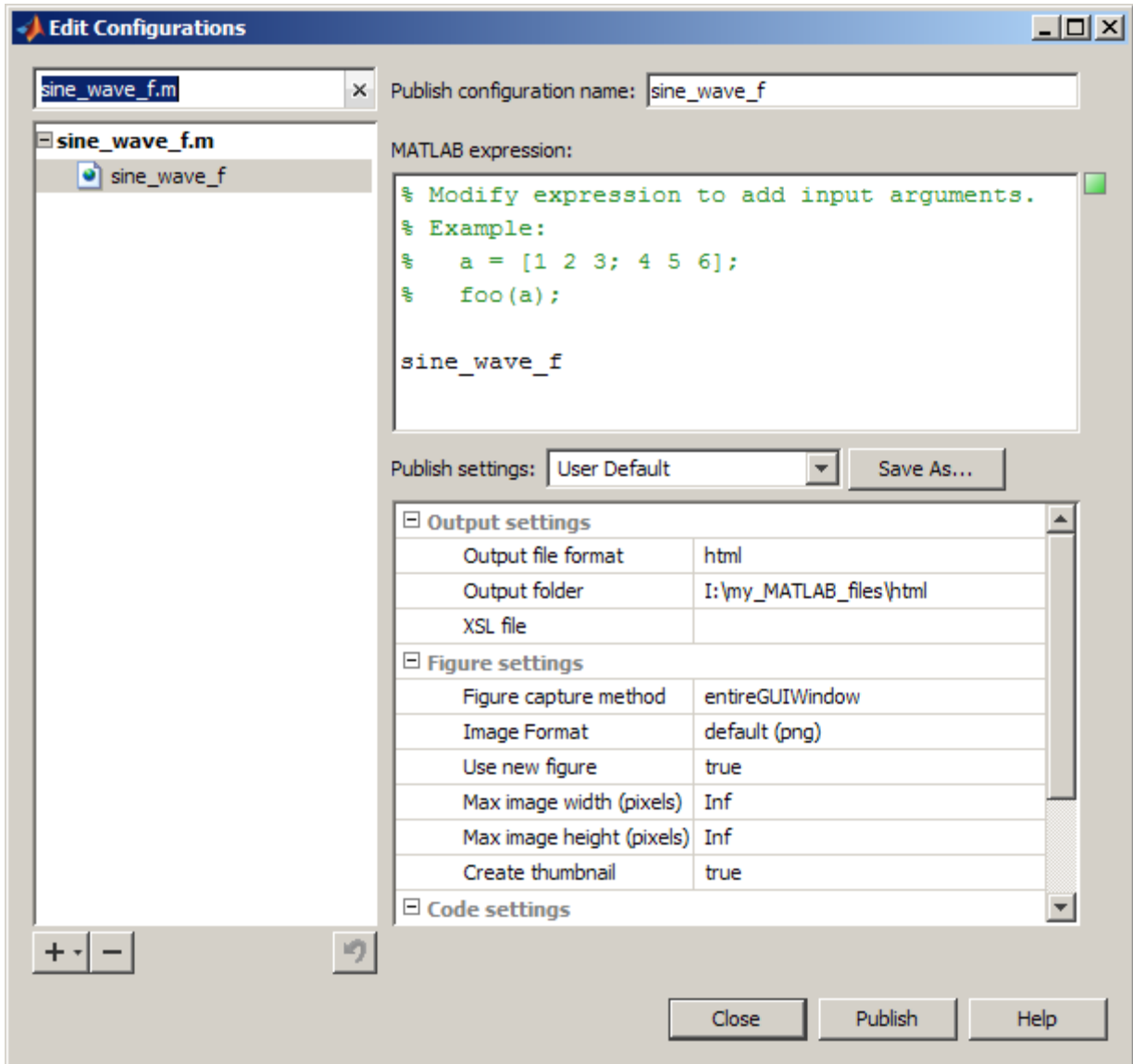
```
%% Modify Plot Properties

title('Sine Wave', 'FontWeight','bold')
xlabel('x')
ylabel('sin(x)')
set(gca, 'Color', 'w')
set(gcf, 'MenuBar', 'none')
```

- 2 Click the down arrow on the **Publish** button  on the Editor toolbar, and select **Edit Publish Configuration for *file name***, where file name in this example is `sine_wave_f.m`.



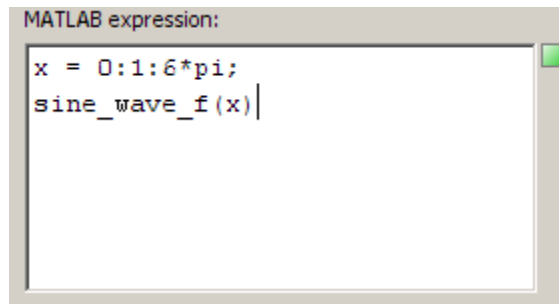
The Edit Configurations dialog box opens, with the default publish configuration template for `sine_wave_f.m`, as shown in the following figure.



- 3 In the **Publish configuration name** field, type a name for the publish configuration, or accept the default name.

If you expect to create multiple configurations for a file, assign each a name that helps you identify the configuration. In this figure, the default name of the configuration is `sine_wave_f`.

- 4 In the **MATLAB expression** field, type the expression that you want the Editor to evaluate when it publishes the file. In this example, delete the commented statements and replace them as shown in the following figure.



You can modify the statements in the **MATLAB expression** area of the dialog box, and then click **Publish** to see the results of the changes. If you clear the **MATLAB expression** area, MATLAB publishes the file without evaluating any code. This is equivalent to setting the **Evaluate code** property in the **Publish settings** properties table to `false`.


- 5 In the **Publish settings** properties table, change the property values if you do not want to use the current settings.

You can modify the property settings, and then click **Publish** to see the results of the changes.

See “Specifying Values for the Publish Settings Property Table” on page 11-74 for details.

- 6 Do one of the following:
 - To publish the file using the settings and MATLAB expression that you have specified, click **Publish**.

For this example MATLAB creates the following files in `I:\my_matlab_files\my_mfiles\html`, which is a subfolder in the folder where `sine_wave_f.m` is located:

- An output file, `sine_wave_f.html`
- A thumbnail file for the last image generated by the MATLAB code, `sine_wave_f.png`
- Image files created by the executable MATLAB code, `sine_wave_f_##.png`
- To create another publish configuration for the same file, click the plus button , and then select **Publish Configuration**.


See “Creating Multiple Publish Configurations for a File” on page 11-93 for details.

- To close the Edit Configurations dialog box, click **Close**. MATLAB saves the configuration and its association with the file.

After creating a configuration, you can view the MATLAB expression and use the configuration to publish the file without opening the Edit Configurations dialog box. See “Running an Existing Publish Configuration” on page 11-92 for details.

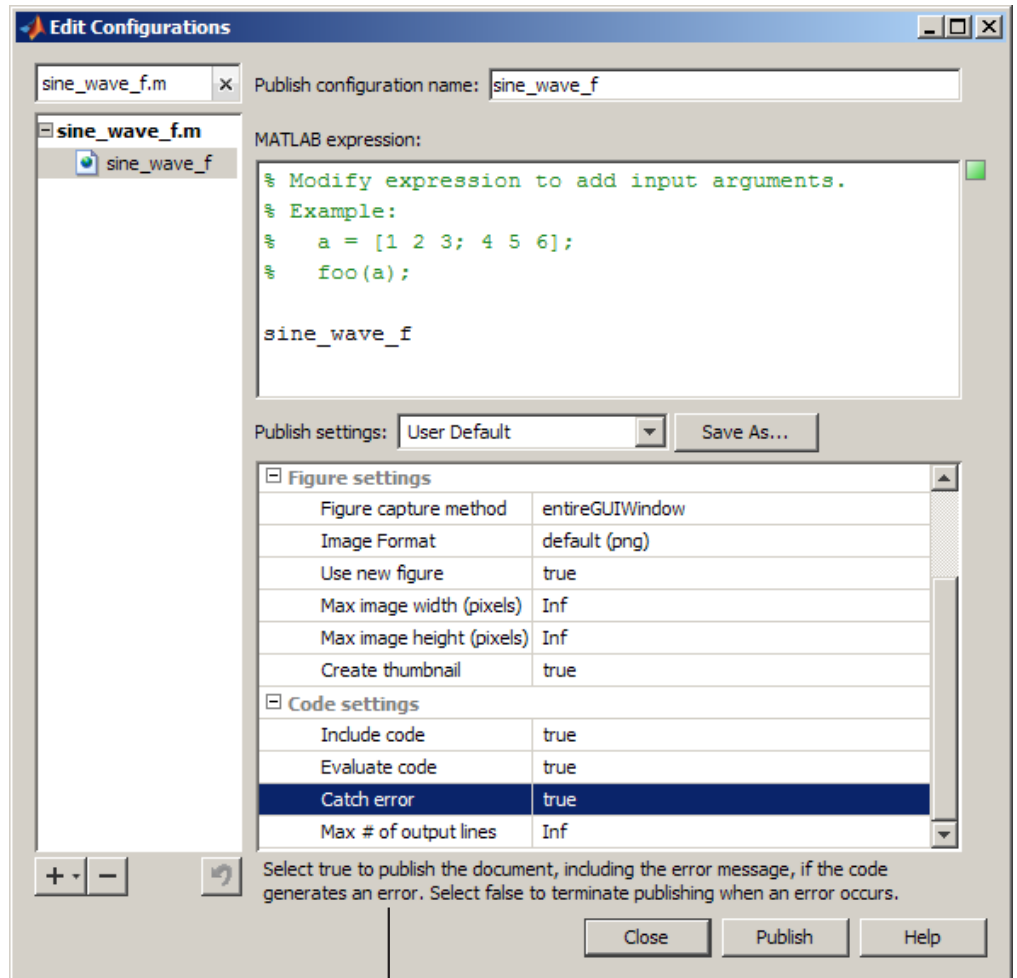
Specifying Publish Configuration Settings

This section describes how to specify new publish settings for a configuration. Publish settings enable you to specify the folder to which an output file is saved, how images generated by the code are captured and included in the output, and so on.

- 1** If the Edit Configurations dialog box is not already open, click the down arrow on the **Publish** button, , and then click the configuration that you want to change.

This example uses the `sine_wave_f` publish configuration as described in “Creating a Publish Configuration for a MATLAB File” on page 11-66.

- 2** View the properties table below the **Publish settings** field to see the current publish property values.
- 3** For information about a property, click the property name. A brief description of that property displays below the publish settings property table. For example, if you click **Catch error**, the dialog box appears as shown in the following image.



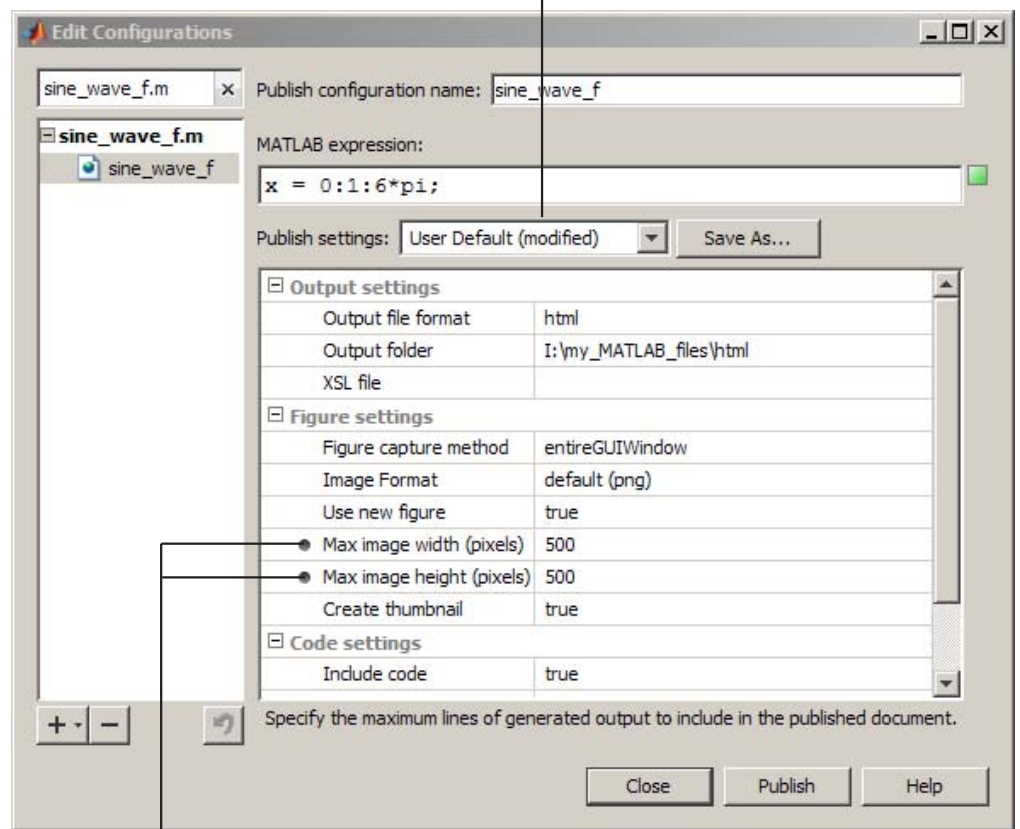
Description of Catch error property

- Optionally, you can change publish setting values by clicking in the column to the right of the property name, and then entering or selecting a property value. This example changes **Max image width** and **Max image height** to 400.

The Editor marks each property that you change with a dot (●) and adds the string, (modified), next to User Default in the **Publish settings** field.

See “Specifying Values for the Publish Settings Property Table” on page 11-74 for information about the various properties you can set.

Indicates that one or more settings differ from the saved User Default property settings



Indicates that these properties' values differ from the settings saved in the User Default publish settings.

- 5 Click **Publish** to preview the publication of the file that is open in the Editor using the new settings.
- 6 When you are satisfied with the results, click **Save As**.

The Save Publish Options dialog box opens and displays the names of all the currently defined publish settings. By default the following publish settings install with MATLAB:

- **Factory Default**

The MATLAB installation includes this set of publishing properties for you to get started with publishing. It enables you to publish a MATLAB file to HTML quickly and view the results. You can use it to test the effect of changing settings. If you determine that the test settings produce undesirable results, you can restore the **Factory Default** publish settings by selecting it from the **Publish settings** drop-down list.

- **User Default**

The MATLAB installation includes this set of publishing properties in anticipation that you will have a set of publishing properties that are common to most or all of your publishing configurations. Initially, **User Default** settings are identical to the **Factory Default** settings. See “Creating a Template for Typical Publish Settings” on page 11-90 for an example of changing the **User Default** settings to best suit your publishing needs.

- 7** In the **Settings Name** field, enter a meaningful name for the settings. For example, `reduce_image`. Then click **Save**.

You can now use the `reduce_image` publish settings with other publish configurations.

You can also overwrite the publishing properties saved in an existing publish settings name. Select it from the **Publish settings** drop-down list, and then click **Overwrite**. However, you cannot overwrite the **Factory Default** publish settings.

Note When you overwrite a publish settings name, publish configurations that currently specify the old name do not have their publish properties updated to reflect the new settings. Instead, properties that have different values from the updated publish settings appear with a dot next to each of them.

- 8** In the Edit Configurations dialog box, do one of the following:
 - Click **Publish** to publish the file that is open in the Editor using the new settings.
 - Click **Close** to close the dialog box.

Specifying Values for the Publish Settings Property Table

The sections that follow describe each of the publish settings properties that you can adjust when you create or update a publish configuration. To access a publish configuration, open the file for which you want to create or update a publish configuration, and then select **File > Publish Configuration for *file name* > Edit Publish Configurations for *file name***.

You can set or adjust values for the following properties:

- “Output file format” on page 11-74
- “Output folder” on page 11-75
- “XSL file” on page 11-75
- “Figure capture method” on page 11-75
- “Image format” on page 11-80
- “Use new figure” on page 11-80
- “Max image width” on page 11-86
- “Max image height” on page 11-86
- “Create thumbnail” on page 11-87
- “Include code” on page 11-87
- “Evaluate code” on page 11-87
- “Catch error” on page 11-89
- “Max # of output lines” on page 11-89

Output file format. Select one of the choices from the drop-down list to publish the document to one of the following file formats:

- `html` — Publishes to an HTML document.

- `xml` — Publishes to an XML document.
- `latex` — Publishes to a LaTeX document.
- `doc` — Publishes to a Microsoft Word document, if your system is a PC.
- `ppt` — Publishes to a Microsoft PowerPoint document, if your system is a PC.
- `pdf` — Publishes to a PDF document.

MATLAB names the output file with the same name as the publish configuration that produced it and stores it, along with images that MATLAB generates from code, in the folder specified with the **Output folder** property.

Output folder. Type the full path of the folder to which you want MATLAB to publish the output document and its associated image files. For example, if your file is in `I:\my_matlab_files\my_mfiles`, you might specify `I:\my_matlab_files\my_word_files` if you are creating a publish configuration for documents that you publish to Word.

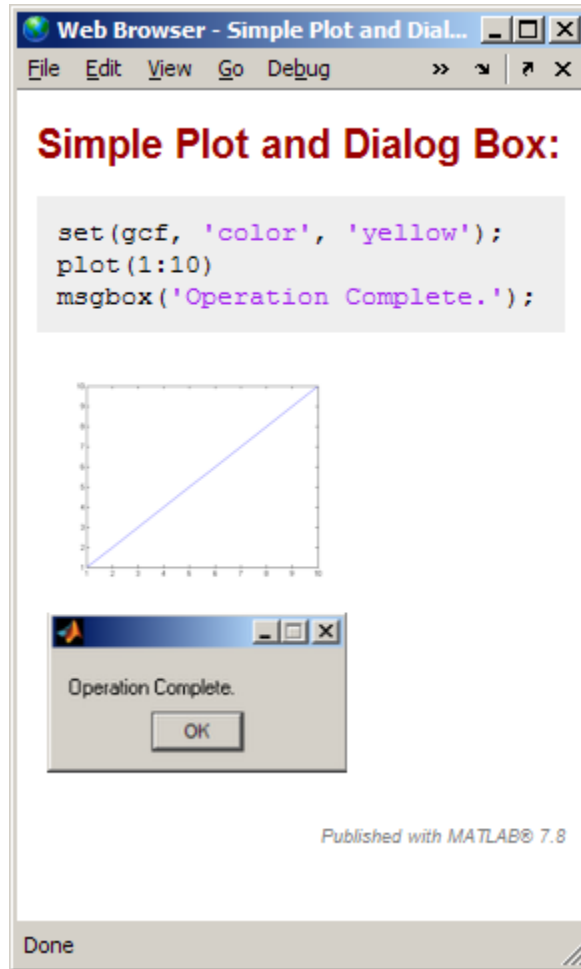
XSL file. Type the full path of the Extensible Stylesheet Language (XSL) file, that you want to use when you specify the **Output file format** as **HTML**, **XML**, or **LaTeX**. If you leave this field blank, MATLAB uses a default stylesheet that installs with the MATLAB software.

Figure capture method. Specify a figure capture method to indicate how you want figures and dialog boxes that the MATLAB code creates to appear when published.

The following list provides some suggestions on how to set this property, depending on the type of document you are publishing. (The document shown in the images that follow was published with the **Max image width** property set to 150 pixels.)

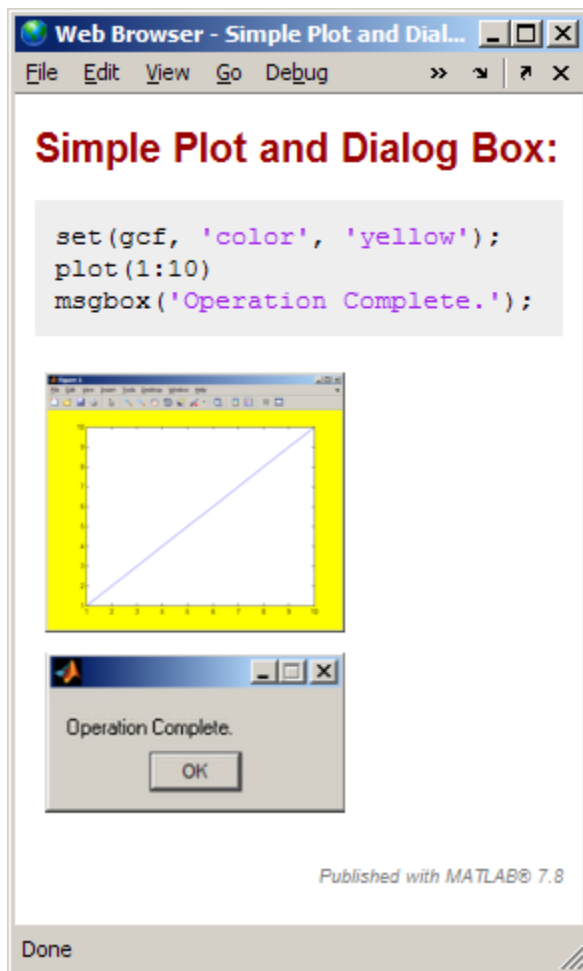
- To output the Figure data and complete dialogs boxes that the MATLAB code creates, set the **Figure capture method** to **entireGUIWindow**.

This method sets the figure background to white, presents just the plot for the figure, but includes the window decorations (title bar, toolbar, menu bar, and window border) for the dialog box in the output.

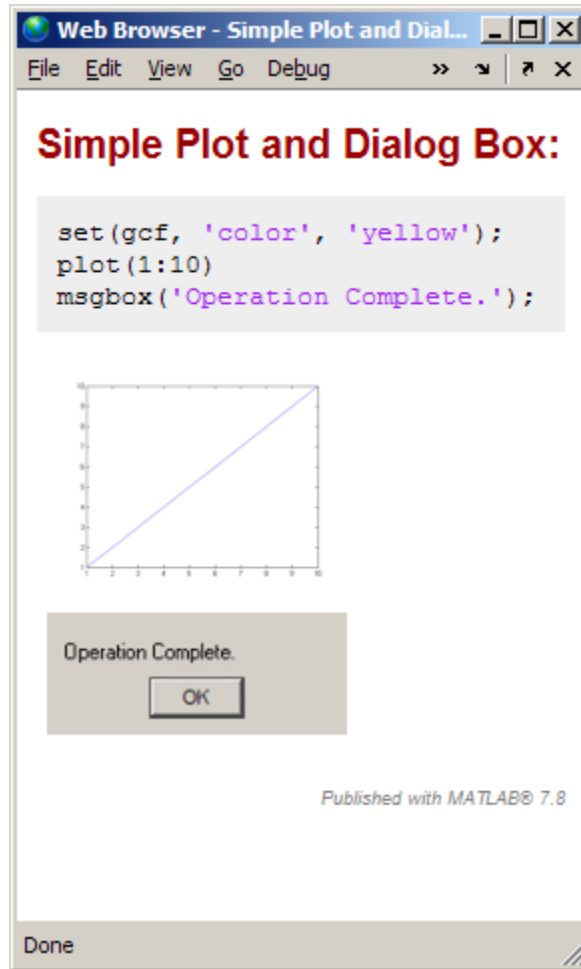


- To output a tutorial on using MATLAB, set the **Figure capture method** to **entireFigureWindow**.

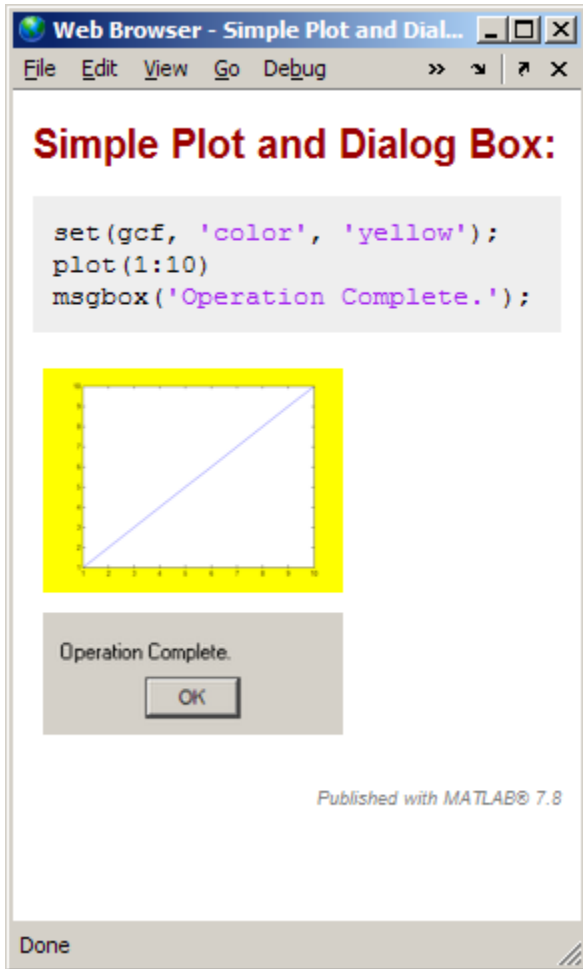
Notice that this method preserves the background color for figures and includes the window decorations for both figures and dialog boxes in the output.



- To output figures with the plot background set to white and dialog boxes without window decorations, set the **Figure capture method** to **print**.



- To output figures and dialog boxes excluding window decorations, but including the background color for figures, set the **Figure capture method** to `getframe`.



The following table summarizes the effects of the various Figure capture methods.

Use this Figure Capture Method	To Get Figure Captures with these Appearance Details	
	Window Decorations	Plot Backgrounds
<code>entireGUIWindow</code>	Included for dialog boxes; Excluded for figures	Set to white for figures; matches the screen for dialog boxes
<code>print</code>	Excluded for dialog boxes and figures	Set to white
<code>getframe</code>	Excluded for dialog boxes and figures	Match the screen plot background
<code>entireFigureWindow</code>	Included for dialog boxes and figures	Match the screen plot background

Note Typically, MATLAB figures have the `HandleVisibility` property set to `on`. Dialog boxes are figures with the `HandleVisibility` property set to `off` or `callback`. If your results are different from the results listed in the preceding list and table, the `HandleVisibility` of your figures or dialog boxes might be atypical. For more information, see “`HandleVisibility` Property” in the MATLAB Graphics documentation.

Image format. Select the file type for images produced when publishing MATLAB files. The image file types available in the drop-down list depend on the **Figure capture method** you specify. You can choose any type available in the drop-down list, but for greatest compatibility select the default.

Use new figure. Set to `true` if you want MATLAB to create a Figure window with a white background and at the default size before publishing if the MATLAB code generates a figure. After publishing finishes, MATLAB closes the Figure window.

To use a figure with different properties for publishing, set this property to `false`. Then open a Figure window, change the size and background color, for example, and then publish. Figures in your output use the characteristics of the figure you opened before publishing.

Note This preference applies to executable MATLAB code that generates a figure. It does not apply to figures included using the **Cell > Insert Text Markup > Image** menu option.

The following example demonstrates how to specify new Figure window properties for output images by setting the **Use new figure** publish settings property to **false**:

- 1 Create `sine_wave_f.m`, as described in “Creating a Publish Configuration for a MATLAB File” on page 11-66.
- 2 Create a Figure window by saving the following code in a file and then running it:

```
function createfigure
%CREATEFIGURE

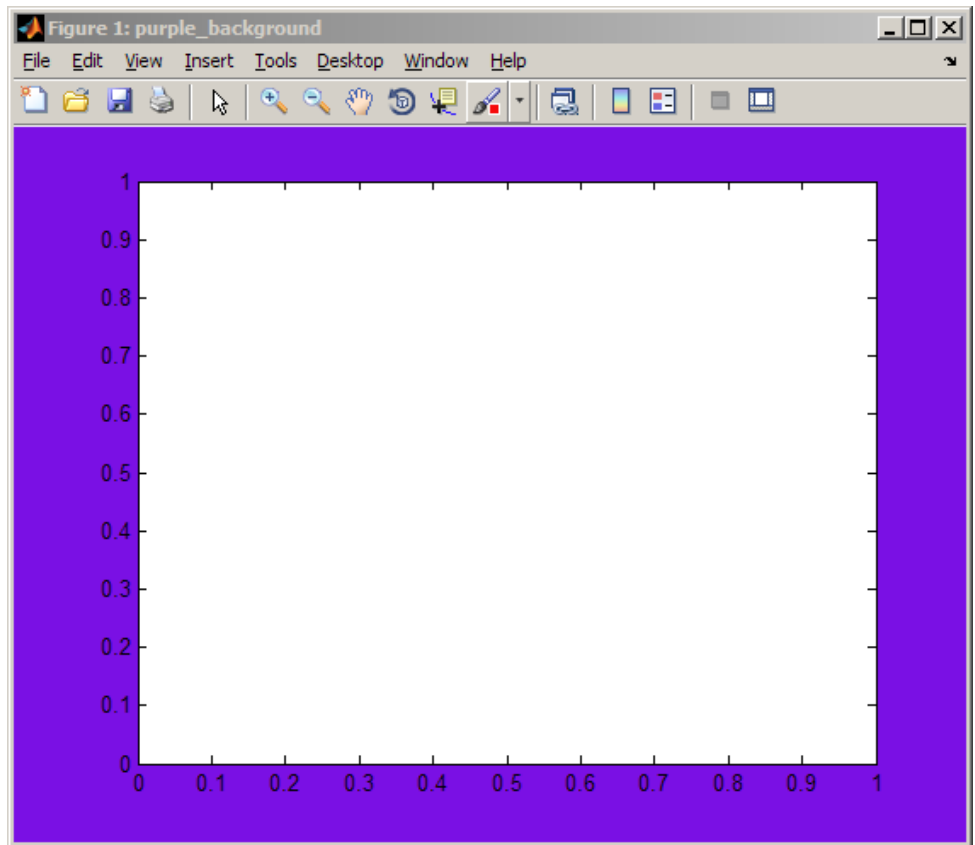
% Create figure
figure1 = figure('Name','purple_background',...
'Color',[0.4784 0.06275 0.8941]);
colormap('hsv');

% Create subplot
subplot(1,1,1,'Parent',figure1);
box('on');

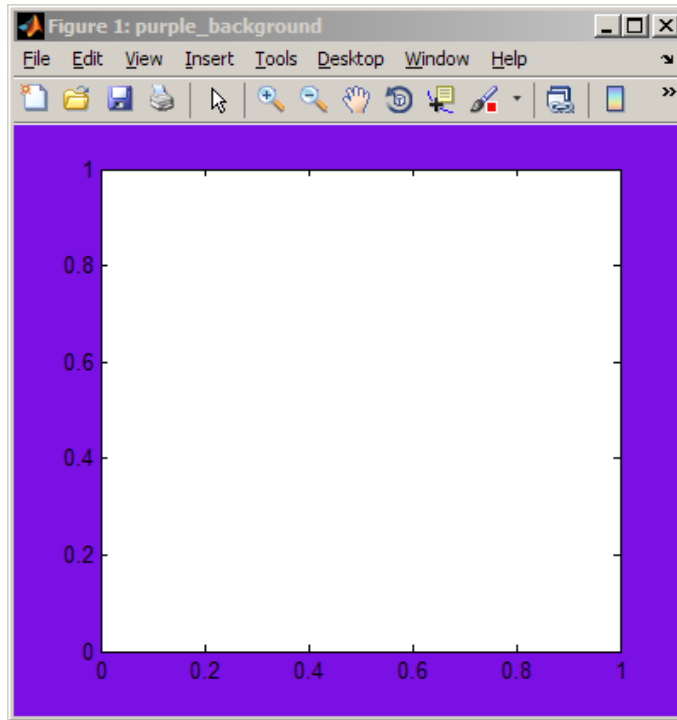
% Create xlabel
xlabel({' '});

% Create title
title({' '});
```

The following figure appears.



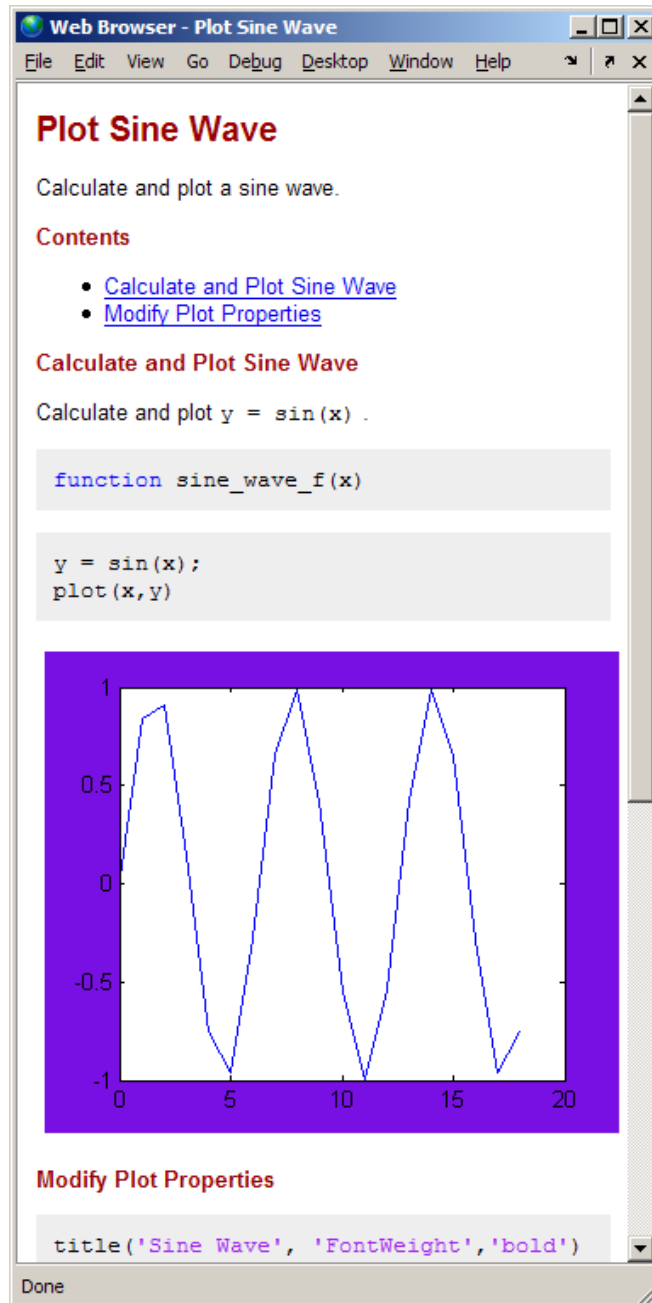
- 3 Reduce the size of the figure by dragging and dropping the edges. For example:



- 4** Do not close the window.
- 5** Make `sine_wave_f.m` the active file in the Editor, and then select **File > Publish Configurations for sine_wave_f.m > Edit Publish Configurations for sine_wave_f.m**.
- 6** In the **Publish settings** drop-down list, select **Factory Default**.
- 7** If you have previously set **Publish settings** for `sine_wave_f.m`, the **Change Publish Settings** dialog box opens. Click **Change to Factory Default**.
- 8** In the **Publish settings** properties table, set **Use new figure** to **false**.

<input checked="" type="radio"/> Use new figure	false
---	-------

- 9 Click **Publish**. MATLAB publishes `sine_wave_f.m` as shown in the following figure.



The screenshot shows a web browser window with the title "Web Browser - Plot Sine Wave". The browser's address bar and menu bar are visible. The main content area displays the following:

Plot Sine Wave

Calculate and plot a sine wave.

Contents

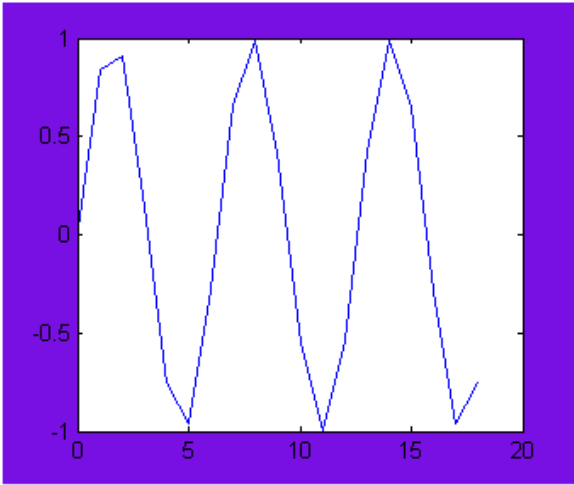
- [Calculate and Plot Sine Wave](#)
- [Modify Plot Properties](#)

Calculate and Plot Sine Wave

Calculate and plot $y = \sin(x)$.

```
function sine_wave_f(x)
```

```
y = sin(x);  
plot(x,y)
```



The plot shows a sine wave oscillating between -1 and 1. The x-axis is labeled from 0 to 20 in increments of 5. The y-axis is labeled from -1 to 1 in increments of 0.5. The plot is enclosed in a purple border.

Modify Plot Properties

```
title('Sine Wave', 'FontWeight','bold')
```

Done

Max image width. Overwrite the current value to restrict the width of images in the output. Note the following about this property:

- It applies only to images that the code generates. It does not apply to images you include using the method described in “Specifying Graphics in MATLAB Files for Publishing” on page 11-30.
- It applies when you select an **Image Format** property setting that is a bitmap, such as `.png` or `.jpg`.
- It does not apply when the **Image Format** property setting is a vector format, such as `.eps`.
- The image’s aspect ratio is maintained. If you restrict both height and width using the **Max image width** and **Max image height** properties to resize the image, then MATLAB maintains the aspect ratio. It does so by using the maximum you specified for one dimension and something less than the maximum for the other dimension.
- MATLAB ignores this property when the **Output file format** is `pdf`.

Max image height. Overwrite the current value to restrict the height of images in the output. Note the following about this property:

- It applies only to images that the code generates. It does not apply to images you include using the method described in “Specifying Graphics in MATLAB Files for Publishing” on page 11-30.
- It applies when you select an **Image Format** property setting that is a bitmap, such as `.png` or `.jpg`.
- It does not apply when the **Image Format** property setting is a vector format, such as `.eps`.
- The image’s aspect ratio is maintained. If you restrict both width and height using the **Max image width** and **Max image height** properties to resize the image, then MATLAB maintains the aspect ratio. It does so by using the maximum you specified for one dimension and something less than the maximum for the other dimension.
- MATLAB ignores this property when the **Output file format** is `pdf`.

Create thumbnail. Set to **true** to direct MATLAB to create a thumbnail image if the **Image Format** preference is a bitmap, such as `.png` or `.jpg`. For example, you can use this thumbnail to represent your file in HTML pages. If you create your own demos and include them in the Help browser **Demos** pane using a `demos.xml` file, MATLAB automatically creates a list of your demos that includes the thumbnail for each.

Set to **false** to direct MATLAB to not create a thumbnail image.

Include code. Set to **true** to have MATLAB include the MATLAB code in the output. Set to **false** to have MATLAB exclude the code from all output file formats except HTML. When the output file format is HTML, MATLAB inserts the MATLAB code in the output file as an HTML comment. Therefore, when viewed in a Web browser, for example, the MATLAB code does not display.

Use the MATLAB `grabcode` function if you want to extract the MATLAB code from the output HTML file.

For example, suppose you publish `I:/my_matlabfiles/my_mfiles/sine_wave_f.m` to HTML using a publish configuration with the **Include code** property set to **false**. If you share the output with colleagues, they can view it in a Web browser. If your colleagues want to see the MATLAB code that generated the output, they can issue the following command from the folder containing `sine_wave_f.html`:

```
grabcode('sine_wave_f.html')
```

MATLAB opens the file that created `sine_wave_f.html` in the Editor.

See “Creating a Publish Configuration for a MATLAB File” on page 11-66 for the `sine_wave_f.m` code.

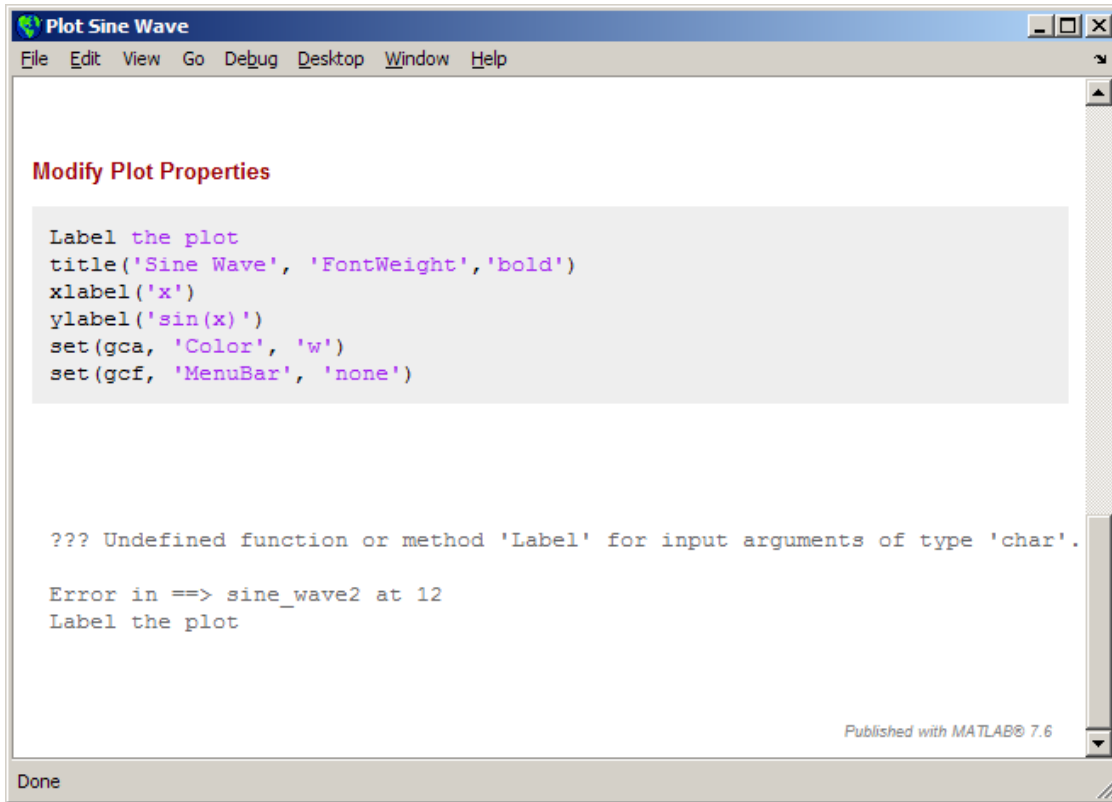
Evaluate code. Set to **true** to direct MATLAB to evaluate the MATLAB code while publishing the results and include the results in the output document. Also specify the **Max # of output lines** property to specify the maximum number of lines you want to include in the output. This property is helpful when you have code that produces much output and you only want to include a sample of it.

Set to **false**, to:

- Direct MATLAB to not evaluate the code, nor include code results when publishing a file.
- Use the `publish` function to publish the file that contains the `publish` function. Otherwise, MATLAB attempts to publish the file recursively.

Because MATLAB does not evaluate the code when you set this property to **false**, there can be invalid code in the file. Therefore, consider first running the file with this property set to **true**.

For example, suppose you include comment text, `Label the plot`, in a file, but forget to preface it with the comment character. If you publish the document to HTML, and set **Evaluate code** to **true**, the output includes the error, such as shown in the following figure.



Catch error. Set to **true** to direct MATLAB to publish and include the error message text in the output if an error occurs when it evaluates the code.

Set to **false** to direct MATLAB to terminate the publish operation if an error occurs when it evaluates the code.

This property has no effect if you set the **Evaluate code** property to false.

Max # of output lines. Type a value to specify the maximum number of output lines that you want to include after each cell break in the output.

For example, suppose your MATLAB code includes a loop, such as the following:

```
for n = 1:100
    disp(x)
end;
```

If you publish the code, then by default, all 100 lines generated by the preceding code appears in the output. If you want to include a smaller representative sample, set **Max # of output lines** to a small value, such as 10.

Creating a Template for Typical Publish Settings


Use the **User Default** publish settings installed with MATLAB to create a template for all or most of your publish configurations.

Initially, the **User Default** publish setting has the same property values as the **Factory Default** publish settings. Update and save your most commonly used property settings to avoid having to reset the same settings each time you create a publish configuration.

For example, suppose that you frequently publish your files using the factory installed **User Default** settings, with a few exceptions. You want to change the factory installed **User Default** settings to:

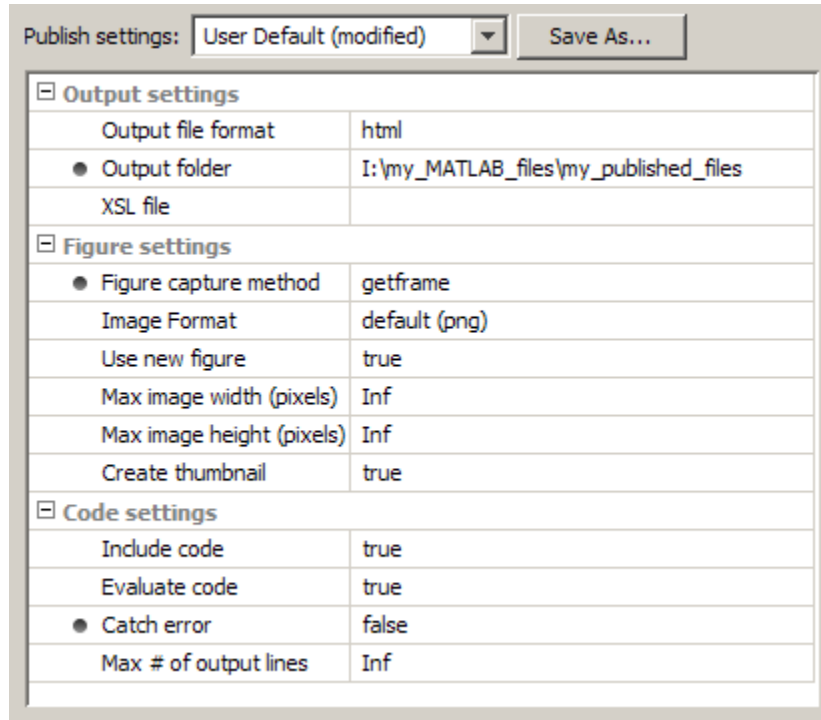
- Save the output files to `I:\my_MATLAB_files\my_published_files`
- Use the `getframe` figure capture method
- Terminate publishing if an error occurs while the MATLAB code is being evaluated

Update the **User Default** publish settings, as follows:

- 1** If the **Edit Configurations** dialog box is not already open, click the down arrow on the **Publish** button , and then click the configuration for which you want to set the properties as described in the preceding list.
- 2** From the **Publish settings** drop-down list, select **User Default**.

If the **Change Publish Settings** dialog box opens, click **Change to User Default**.

- 3** Adjust the values in the publish settings properties table, so that the **Publish settings** appear as shown in the following figure.



4 Click **Save As**.


The Save Publish Settings dialog box opens.

5 In the **Publish settings** drop-down list, select **User Default**, and then click **Overwrite**.

The **User Default** publish settings are saved with the specified property values.

Now, suppose you want to create a publish configuration using all the same settings, except you want to publish your file to a Microsoft Word document. Follow these steps:

1 In the Editor, open the file that you want to publish to a Word document.

- 2 Click the down arrow next to the Publish button  on the Editor toolbar and click **Edit Publish Configuration for file name**, where the file name is the name of the file that you want to publish to a Word document.

The Edit Configurations dialog box opens. Notice that the **Publish settings** is **User Default** and the publish settings properties table contains the values you set in the preceding list of steps.

- 3 If you want, adjust the MATLAB expression.
- 4 Change the **Output file format** from **html** to **doc**.
- 5 Click **Save As**.

The Save Publish Settings dialog box opens.


- 6 In the **Settings name** box, type a name for the new group of publish settings. For example, `WordDefault`.
- 7 Click **Save**.

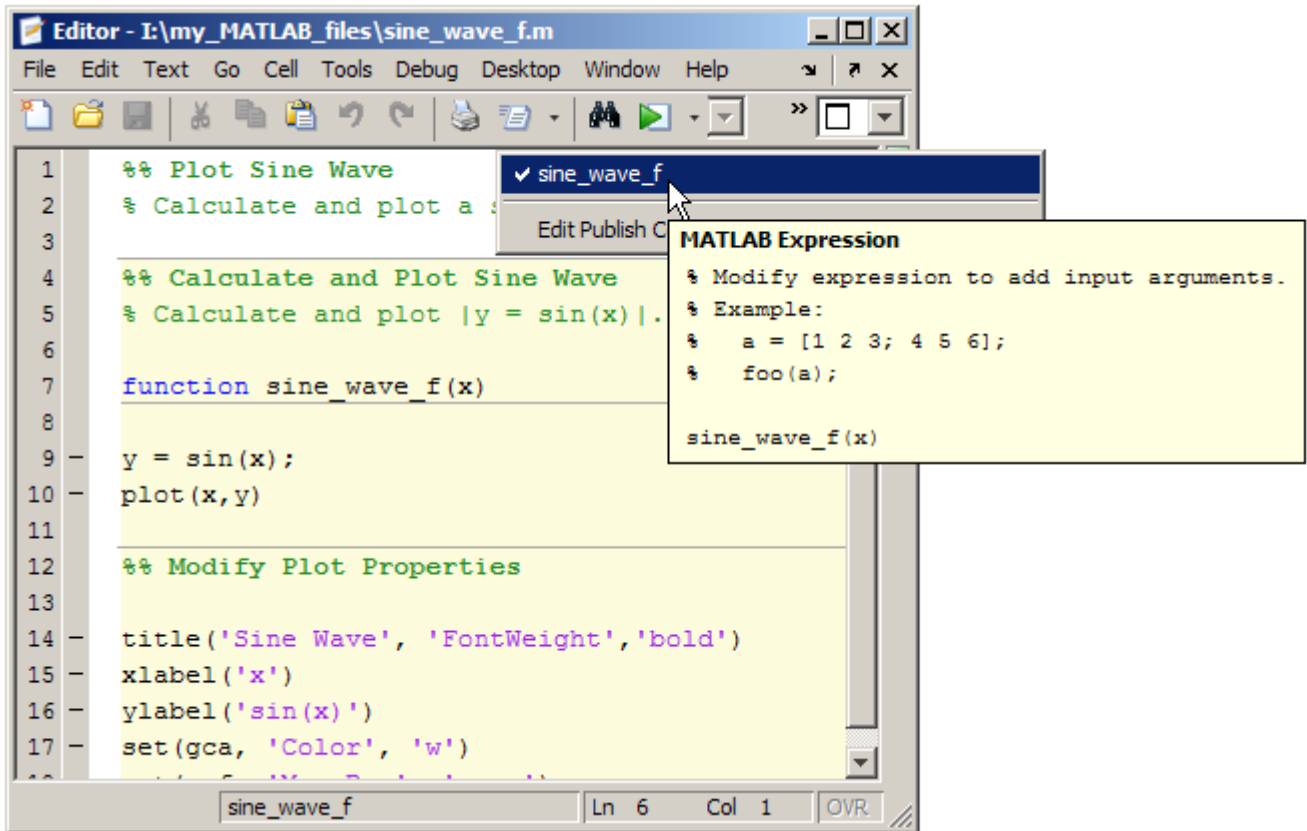
Now you can use any one of the following publish settings as the basis for new publish settings, for the next publish configuration you create:

- Factory Default
- Your customized User Default
- Word Default
- Any other publish settings that you create and save with a unique name

Running an Existing Publish Configuration

After creating a publish configuration, you can run the configuration without opening the Edit Configurations dialog box, as follows:

- 1 In the Editor toolbar, click the down arrow on the **Publish** button , and position the mouse pointer on a publish configuration name. MATLAB displays a Tooltip showing the publish configuration's MATLAB expression so you can see what will be evaluated when you publish the file using the named configuration.



- 2 To use the publish configuration, select a configuration name. MATLAB publishes the file using the MATLAB expression you specified in the publish configuration. For example, if you select `sine_wave_f`, MATLAB sets the value of the input argument, `x`, to `0:1:6*pi` and passes it to the MATLAB function before evaluating and publishing the file. (To see how to set the MATLAB expression, see “Creating a Publish Configuration for a MATLAB File” on page 11-66.)

Creating Multiple Publish Configurations for a File

You can create multiple publish configurations for a given file. You might do this to publish the file with different values for input arguments, with different publish setting property values, or both. Create a named

configuration for each purpose, all associated with the same file. Then, any time you publish the file, you can choose and run whichever particular publish configuration that you want. For example, for `sine_wave_f(x)` you might use different values for `x` and adjust publishing properties for these purposes:

- For reviewing with colleagues, publish the document to Word. Use publish settings to adjust the size of images generated by the code so they are not cropped in the document. Evaluate and include the code, as well as any errors generated by the code in the Word document.
- For inclusion in a blog, publish the document to HTML. Use publish settings to:
 - Specify an argument value.
 - Set publishing properties to evaluate and include the code in the output published to HTML.
 - Set publishing properties to exclude errors generated by the code from the output published to HTML.
- For presentation at a meeting, use the same settings as used for publishing to the blog, but publish to Microsoft PowerPoint.
- For a polished presentation, publish to PDF.

The following sections provide instructions for creating multiple configurations for `sine_wave_f.m`.

- “Example of Publishing `sine_wave_f.m` to Microsoft Word” on page 11-94
- “Steps for Publishing `sine_wave_f.m` to HTML” on page 11-97
- “Steps for Publishing `sine_wave_f.m` to Microsoft® PowerPoint” on page 11-100
- “Steps for Publishing `sine_wave_f.m` to PDF” on page 11-102

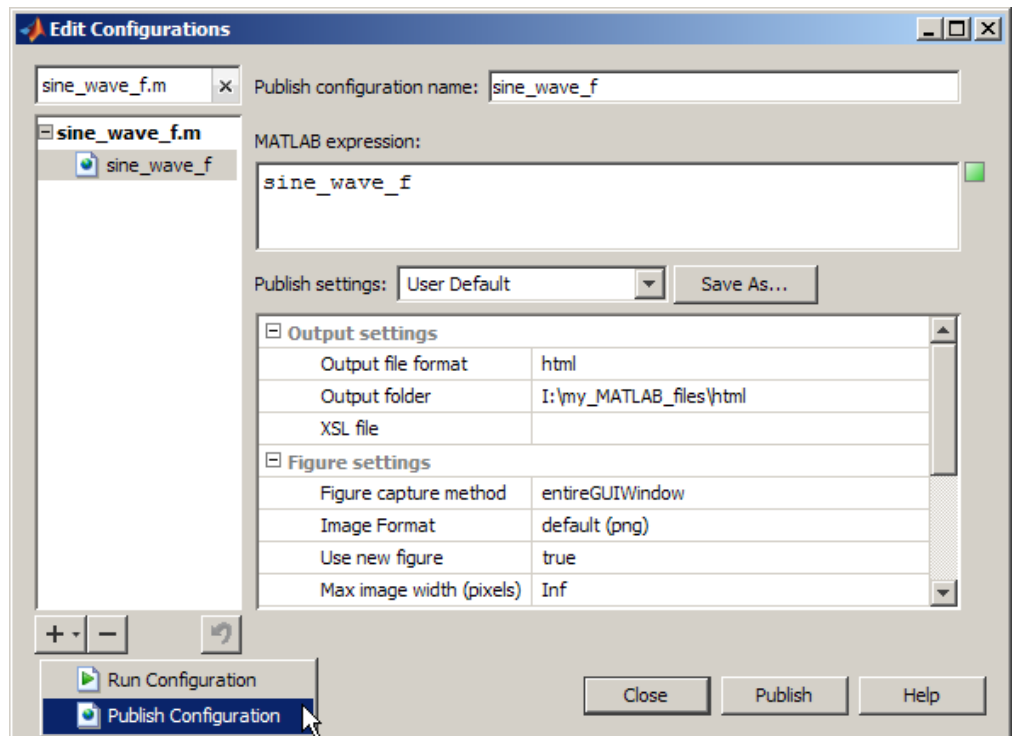
Example of Publishing `sine_wave_f.m` to Microsoft Word

The following steps provide an example of settings you might use when you want to publish a file to Word. This example uses the `sine_wave_f.m` file, the code for which appears in “Creating a Publish Configuration for a MATLAB File” on page 11-66.

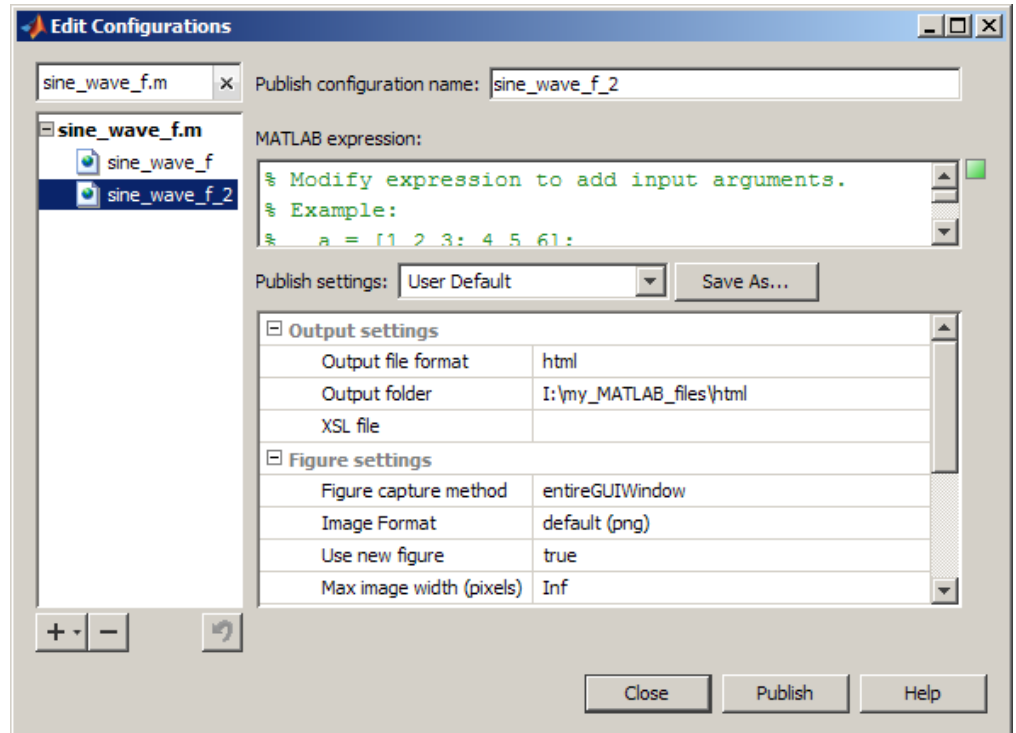
- 1 Copy `sine_wave_f.m` to your current folder. If you have write permission to your current folder, you can type the following in the Command Window to copy the file from the MATLAB root folder:

```
copyfile(fullfile(docroot, 'techdoc', 'matlab_env', 'examples', ...  
'sine_wave_f.m'), '.', 'f')
```

- 2 In the Editor, open `sine_wave_f.m`.
- 3 Select **File > Publish Configuration for sine_wave_f.m > Edit Publish Configurations for sine_wave.m**.
- 4 Select `sine_wave_f` in the list of files and configurations, click the down arrow next to the **Add** button **+ ▾**, and then select **Publish Configuration**.



MATLAB creates a publish configuration, `sine_wave_f_n` where the value of n depends on the number of publish configurations you have previously created for `sine_wave_f`.



- 5 Rename `sine_wave_f_n` to `sine_wave_word`, and replace the default template expression with the following code:

```
x = 0:1:rand*pi;
sine_wave_f(x)
```

- 6 Change the values for **Publish settings**, as follows so that the output is a Word document that includes the code, its output and any errors the code might generate. The maximum values for the image height and width are set so that the images are not cropped in the Word document:

- a For **Output file format**, select `doc` from the drop-down list.

- b** For **Image format**, select jpeg from the drop-down list.
 - c** For **Max image width**, type 400.
 - d** For **Max image height**, type 400.
- 7** Click **Publish** to test how the settings affect the Word document.

You can continue to test and change publish settings until you achieve the results that you want.

Tip In addition to testing that your MATLAB code evaluates as expected and publishes to Word as expected, consider running the spelling and grammar checker in Word to be sure that the comments in your code do not contain typographical or grammatical errors.

- 8** Optionally, if you plan to reuse these publish settings later, click **Save As**. In the Save Publish Settings dialog box, in the **Settings name** field, type `word_settings`, and then click **Save**.


Steps for Publishing `sine_wave_f.m` to HTML

These steps provide an example of creating a configuration for `sine_wave_f.m`, that publishes the file to HTML. You might do this to publish output for inclusion in a blog, for example.

- 1** Copy `sine_wave_f.m` to your current folder. If you have write permission to your current folder, you can type the following in the Command Window to copy the file from the MATLAB root folder:

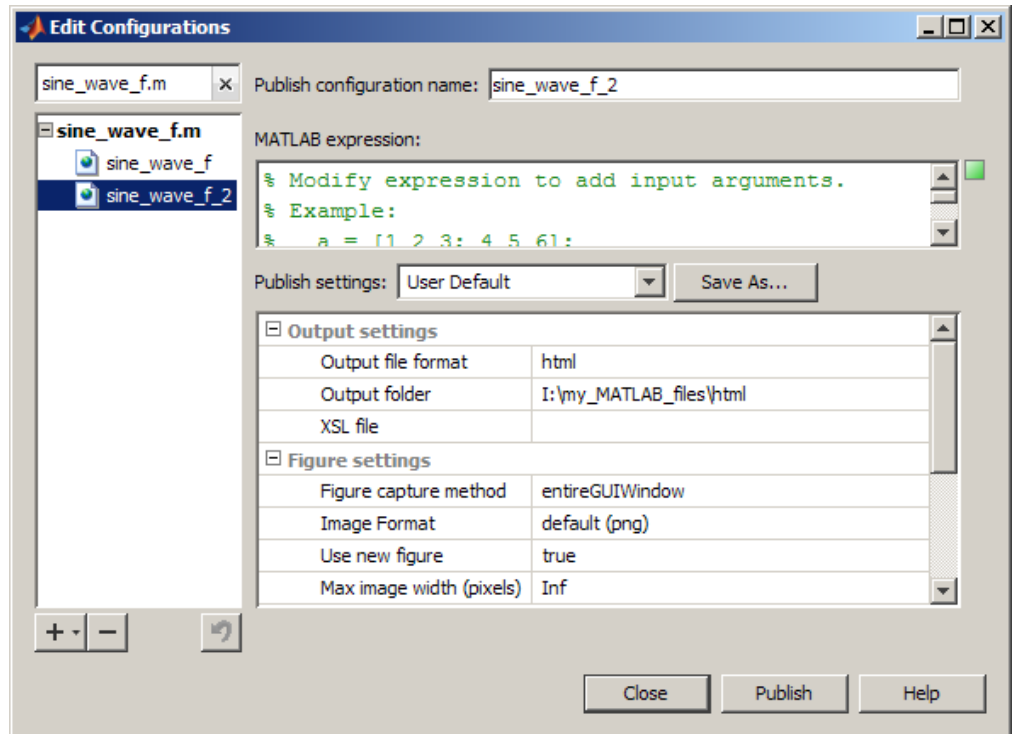
```
copyfile(fullfile(docroot, 'techdoc', 'matlab_env', 'examples', ...
    'sine_wave_f.m'), '.', 'f')
```

- 2** In the Editor, open `sine_wave_f.m`.
- 3** Select **File > Publish Configuration for sine_wave_f.m > Edit Publish Configurations for sine_wave_f.m**.

Select `sine_wave_f` in the list of files and configurations, click the down arrow next to the **Add** button , and then select **Publish Configuration**.

- 4 Select `sine_wave_f` in the list of files and configurations, click the down arrow next to the **Add** button \downarrow , and then select **Publish Configuration**.

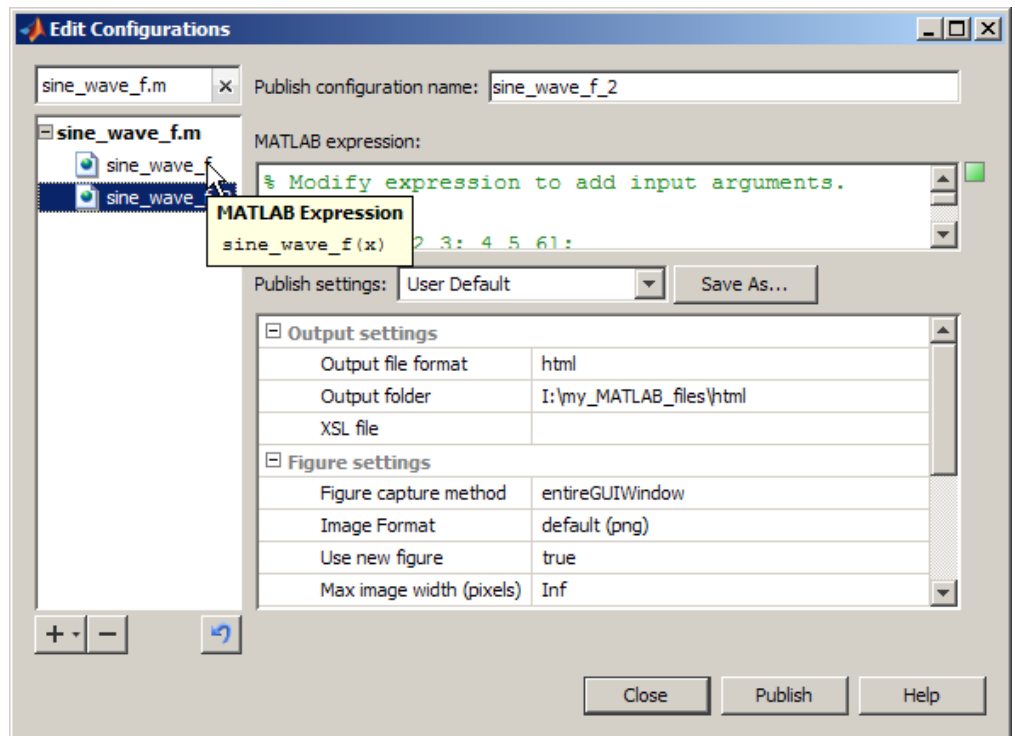
MATLAB creates a publish configuration, `sine_wave_f_n` where the value of n depends on the number of publish configurations you have previously created for `sine_wave_f`.



- 5 In the **Publish configuration name** field, replace `sine_wave_f_n` with `sine_wave_html`.
- 6 In the **MATLAB expression** field, replace the default expression with the following:

```
x = 0:1:rand*pi;
sine_wave_f(x)
```


Tip To get a quick view of the expression used in a different configuration, position the mouse pointer on the name of a different publish configuration without selecting it. In the following figure, `sine_wave_html` is selected, but the mouse pointer is positioned on `sine_wave_f`. You can see the MATLAB expression specified for the `sine_wave_f` configuration in the Tooltip.



- 7 Change the values for **Publish settings**, as follows so that the file publishes to an HTML document, including the code, its output and any errors the code might generate. The maximum values for the image height and width are set so that the images are not cropped in the Word document:
 - a For **Output file format**, select `html` from the drop-down list.
 - b For **Max image width**, type 400.

- c** For **Max image height**, type 400.
- 8** Click **Publish** to test how the settings affect the HTML document.


You can continue to test and change publish settings until you achieve the results that you want.
- 9** Optionally, if you plan to reuse these publish settings later, click **Save As**. In the Save Publish Settings dialog box, in the **Settings name** field, type `html_settings`, and then click **Save**.

Steps for Publishing `sine_wave_f.m` to Microsoft PowerPoint

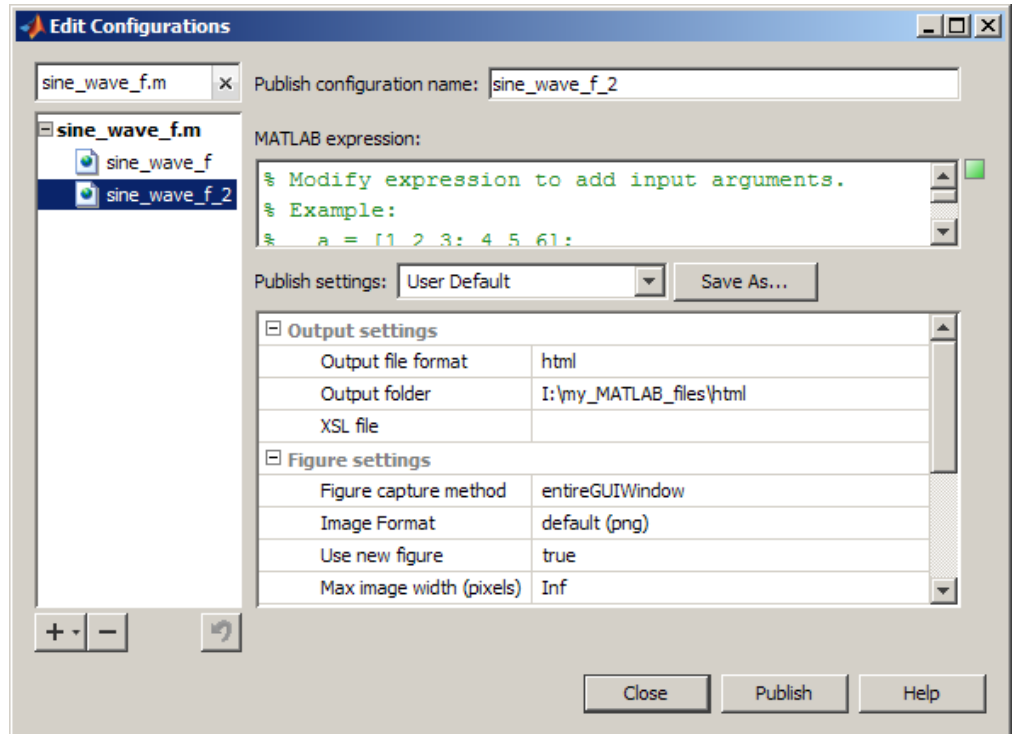
These steps provide an example of creating a configuration for `sine_wave_f.m`, that publishes the file to Microsoft PowerPoint. You might do this to publish output for presentation in a meeting, for example.

- 1** Copy `sine_wave_f.m` to your current folder. If you have write permission to your current folder, type the following in the Command Window to copy the file from the MATLAB root folder:

```
copyfile(fullfile(docroot, 'techdoc', 'matlab_env', 'examples', ...  
'sine_wave_f.m'), '.', 'f')
```

- 2** In the Editor, open `sine_wave_f.m`.
- 3** Select **File > Publish Configuration for `sine_wave_f.m` > Edit Publish Configurations for `sine_wave_f.m`**.
- 4** Select `sine_wave_f` in the list of files and configurations, click the down arrow next to the **Add** button , and then select **Publish Configuration**.

MATLAB creates a publish configuration, `sine_wave_f_n` where the value of n depends on the number of publish configurations you have previously created for `sine_wave_f`.



5 In the **Publish configuration name** field, replace `sine_wave_f_n` with `sine_wave_ppt`.

6 In the **MATLAB expression** field, replace the default expression with the following:

```
x = 0:1:6*pi;
sine_wave_f(x)
```

7 Assume for the purposes of a PowerPoint presentation, you do not want to include the code.

Change the **Output file format** to **ppt** and **Include code** to **false**.

8 Click **Publish** to test how the PowerPoint output appears.

- 9 Optionally, if you plan to reuse these publish settings later, click **Save As**. In the Save Publish Settings dialog box, in the **Settings name** field, type `ppt_settings`, and then click **Save**.

Steps for Publishing `sine_wave_f.m` to PDF

This example uses the `sine_wave_f.m` file, the code for which appears in “Creating a Publish Configuration for a MATLAB File” on page 11-66.


- 1 Copy `sine_wave_f.m` to your current folder. If you have write permission to your current folder, type the following in the Command Window to copy the file from the MATLAB root folder:

```
copyfile(fullfile(docroot, 'techdoc', 'matlab_env', 'examples', ...  
'sine_wave_f.m'), '.', 'f')
```

- 2 Open `sine_wave_f.m` in the Editor.

```
edit sine_wave_f.m
```

- 3 Select **File > Publish Configuration for sine_wave_f.m > Edit Publish Configurations for sine_wave_f.m**.

- 4 Select `sine_wave_f` in the list of files and configurations, click the down arrow next to the **Add** button , and then select **Publish Configuration**.

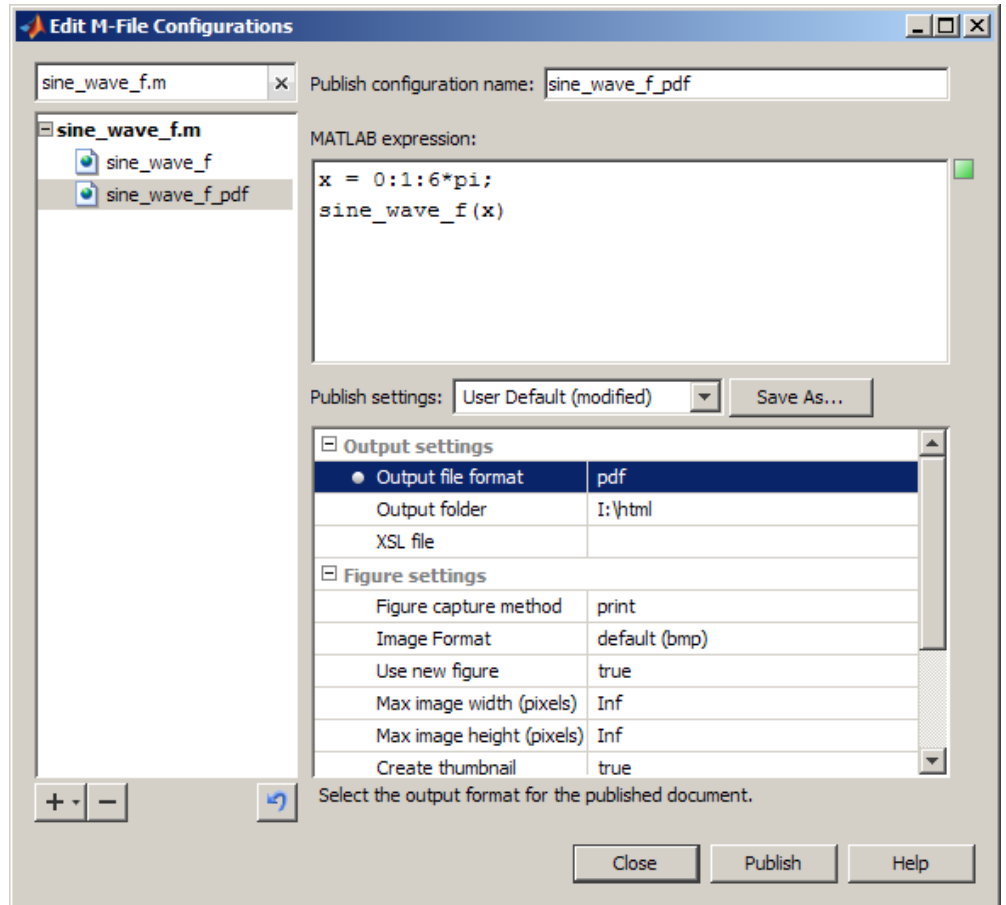
MATLAB creates a publish configuration, `sine_wave_f_n`, where the value of n depends on the number of publish configurations you have previously created for `sine_wave_f`.

- 5 In the **Publish configuration name** field, replace `sine_wave_f_n` with `sine_wave_pdf`.

- 6 In the **MATLAB expression** field, replace the default expression with the following:

```
x = 0:1:6*pi;  
sine_wave_f(x)
```

- 7 Change the **Output file format** to pdf.



- 8 Click **Publish** to test how the PDF output appears.

Notice that when you publish to PDF, unlike other publishing output formats, the introductory text appears after the table of contents:

Plot Sine Wave	
Table of Contents	
Calculate and Plot Sine Wave	1
Modify Plot Properties	1
Introductory text → Calculate and plot a sine wave.	

- 9 Optionally, if you plan to reuse these publish settings later, click **Save As**. In the Save Publish Settings dialog box, in the **Settings name** field, type `pdf_settings`, and then click **Save**.

About the `publish_configurations.m` File

When you create one or more publish configurations using the Edit Configurations dialog box, the Editor updates the `publish_configurations.m` file in your preferences folder. (This is the folder that MATLAB returns when you run the MATLAB `prefdir` function.)

Although you can port this file from the preferences folder on one system to another, there can only be one `publish_configurations.m` file on a system. Therefore, only do this if you have not already created configurations on the second system. In addition, because this file might contain references to file paths, be sure that the specified files and paths exist on the second system.

MathWorks recommends that you not update `publish_configurations.m` in the MATLAB Editor or a text editor. Changes that you make using tools other than the Edit Configurations dialog box might be overwritten later. Each time you save a configuration using the Edit Configurations dialog box, MATLAB updates the `publish_configurations.m` file, as well as the `run_configurations.m` file. For more information, see “About the `run_configurations.m` File” on page 9-96.

Finding Publish Configurations

The method you use to find publish configurations is the same as the one you use to find run configurations. For details, see “Find Configurations” on page 9-96.

Removing Publish Configurations

If you no longer need a publish configuration because you do not use it or because you deleted the file with which it is associated, it is a good practice to delete the publish configuration. The method you use to delete publish configurations is the same as the one you use to delete run configurations. For details, see “Remove Configurations” on page 9-98 for details.

Reassociating and Renaming Publish Configurations

Each publish configuration is associated with a specific file. If you move or rename the file, redefine the association. If you delete a file, consider deleting the associated configurations, or associating them with a different file. You might also need to modify the statements in the configurations so they will run. The method you use to reassociate and rename publish configurations is the same as the one you use to reassociate and rename run configurations. See “Reassociate and Rename Configurations” on page 9-99 for details.

Summary of Options for Presenting Your Code to Others

In addition to publishing, MATLAB provides other options for presenting your code to others. The following table summarizes the options and presents them in order of ease of implementing.

Method	Description	Output Formats	Details
Command line help	Using comments at the start of a MATLAB program, enable the display of help comments when you type <code>help function-name</code> or <code>help class-name</code> in the Command Window.	<ul style="list-style-type: none"> • ASCII text 	See “Adding Help for Your Program Files” on page 5-9.
Publish	Using comments and basic markup, publish a document that includes comment text, MATLAB code, and code output. You can produce numerous output formats from the same code.	<ul style="list-style-type: none"> • XML • HTML • LaTeX • Microsoft Word (.doc) • Microsoft PowerPoint (ppt) • PDF 	See “Overview of Publishing MATLAB Code” on page 11-2.
Notebook	Using Microsoft Word, create electronic or printed records of MATLAB sessions, class notes, textbooks or technical reports. You must have Microsoft Word software installed.	<ul style="list-style-type: none"> • Microsoft Word (.doc) 	See Chapter 12, “Creating a MATLAB Notebook to Publish to Microsoft Word”
Help Browser Topics	Create HTML and XML files to provide your own MATLAB help topics for viewing from the MATLAB Help browser or the Web.	<ul style="list-style-type: none"> • HTML 	See “Adding HTML Help Files to the Help Browser” on page 5-17

Method	Description	Output Formats	Details
Help Browser Demos	Create HTML files to provide your own demos, including videos for viewing from the MATLAB Help browser.	<ul style="list-style-type: none">• Text• HTML	See “Adding Demos to the Help Browser” on page 5-52
MATLAB Report Generator	Using MATLAB Report Generator build complex reports. You must have MATLAB Report Generator software installed.	<ul style="list-style-type: none">• RTF• PDF• Word• HTML• XML	See <i>MATLAB Report Generator User’s Guide</i>

Creating a MATLAB Notebook to Publish to Microsoft Word

You can use the `notebook` command to create electronic or printed records of MATLAB sessions, class notes, textbooks, or technical reports to Microsoft Word. This document is known as a MATLAB Notebook. As an alternative, consider using the `MATLAB publish` command. For more information, see Chapter 11, “Publishing MATLAB Code”.

Note The `notebook` command is available only on Windows systems that have Microsoft Word installed. For supported versions of Word, see “Configuring MATLAB notebook” on page 12-28.

- “About MATLAB Notebooks” on page 12-2
- “Defining MATLAB Commands as Input Cells for a MATLAB Notebook” on page 12-12
- “Evaluating MATLAB Commands in a MATLAB Notebook” on page 12-17
- “Printing and Formatting a MATLAB Notebook” on page 12-23
- “Configuring MATLAB notebook” on page 12-28
- “Notebook Feature Reference” on page 12-30

About MATLAB Notebooks

In this section...

“Contents of MATLAB Notebooks” on page 12-2

“Creating or Opening a MATLAB Notebook” on page 12-2

“Entering Commands in a MATLAB Notebook” on page 12-9

“Protecting the Integrity of Your Workspace in MATLAB Notebooks” on page 12-10

“Ensuring Data Consistency in MATLAB Notebooks” on page 12-10

“Debugging and MATLAB Notebooks” on page 12-11

Contents of MATLAB Notebooks

Using the `notebook` command, you can create a Microsoft Word document, that contains text, MATLAB commands, and the output from MATLAB commands.

You can think of this document as a record of an interactive MATLAB session annotated with text, or as a document embedded with live MATLAB commands and output. This documentation refers to this Microsoft Word document as a MATLAB Notebook.

Creating or Opening a MATLAB Notebook

This section includes information on performing the following tasks:

- “Issuing the notebook Command from the MATLAB Desktop” on page 12-2
- “Creating a MATLAB Notebook” on page 12-5
- “Opening an Existing MATLAB Notebook” on page 12-6
- “Converting a Word Document to a MATLAB Notebook” on page 12-7

Issuing the notebook Command from the MATLAB Desktop

To create a MATLAB Notebook from within MATLAB desktop, type the following in the Command Window:

notebook

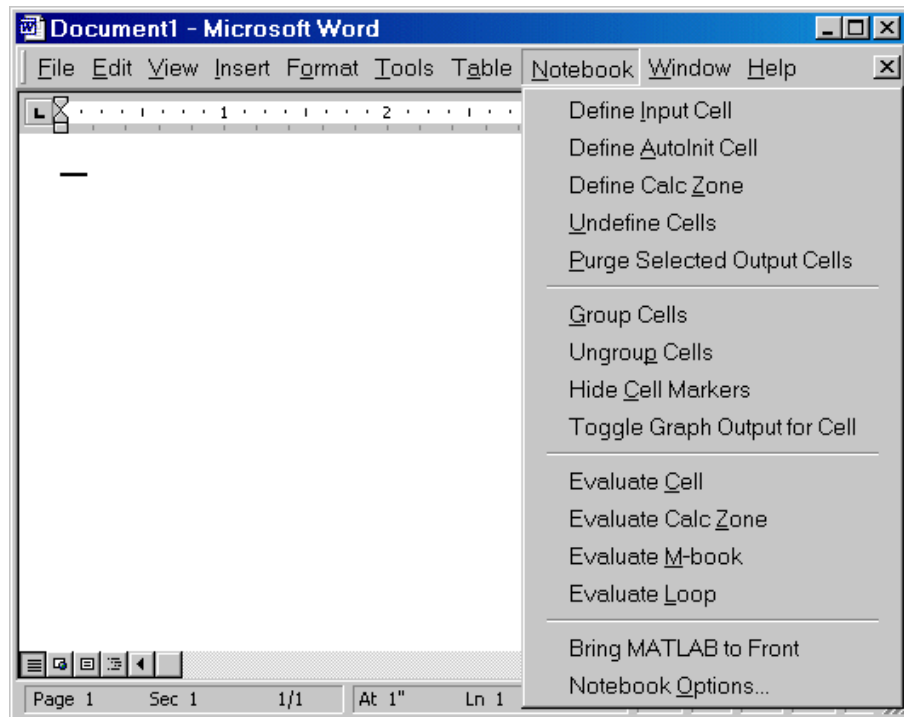
If you are running `notebook` for the first time, you might need to configure it. See “Configuring MATLAB notebook” on page 12-28 for more information.

The `notebook` command starts Word on your system and creates a MATLAB Notebook, called `Document1`.

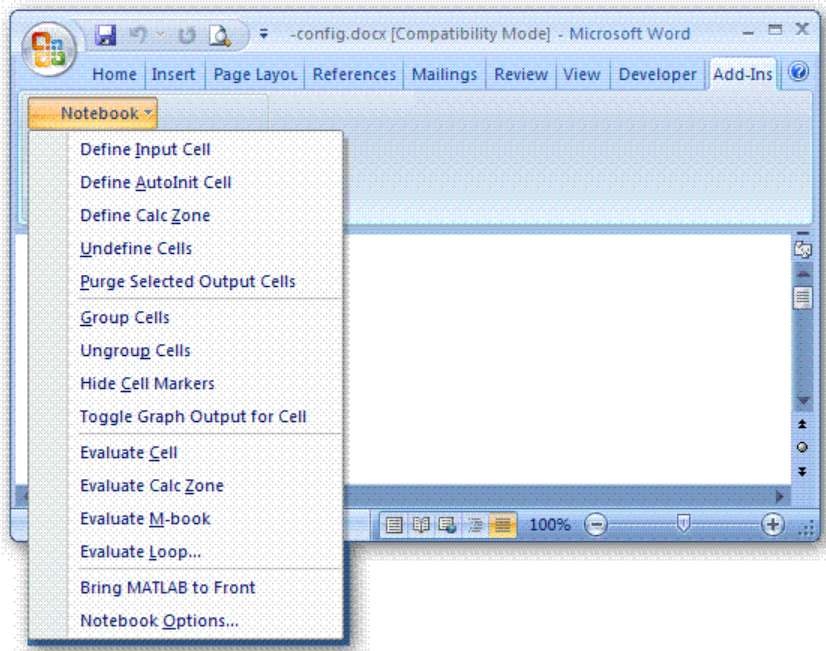
When Word is opening, if a dialog box appears asking you to enable or disable macros, choose to enable macros. The `notebook` command defines Microsoft Word macros that enable MATLAB to interpret the different types of cells that hold MATLAB commands and their output. For more information on macro security, see “Configuring MATLAB notebook” on page 12-28.

Depending on the version of Word you are using, one of the following occurs:

- In Word 2002, and 2003, `notebook` adds the **Notebook** menu to the Word menu bar, as shown in the following illustration.



- In Word 2007, notebook adds the **Notebook** menu to the Word **Add-Ins** tab, as shown in the following illustration.

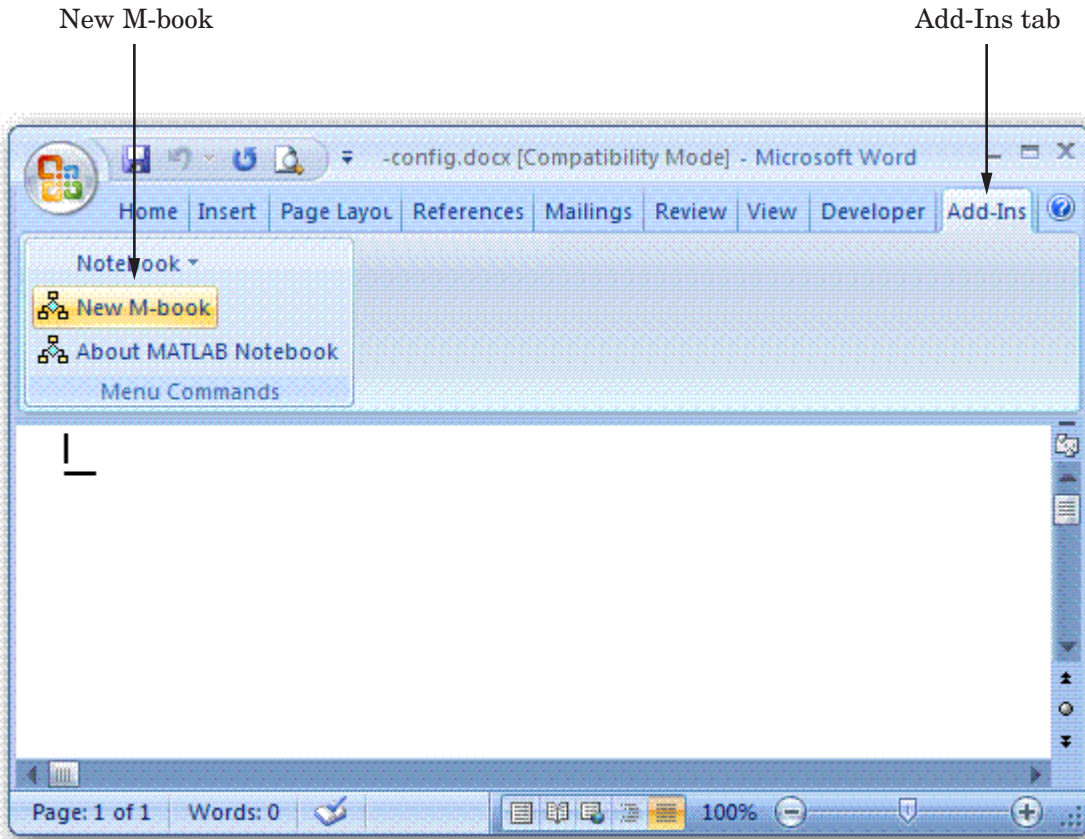


Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Creating a MATLAB Notebook

After issuing the notebook command, you can create a MATLAB Notebook as follows:

- In Word 2002, and 2003, select **File > New M-book**
- In Word 2007, select **Add-Ins > New M-book**, as shown in the following figure.



Microsoft product screen shot reprinted with permission from Microsoft Corporation.

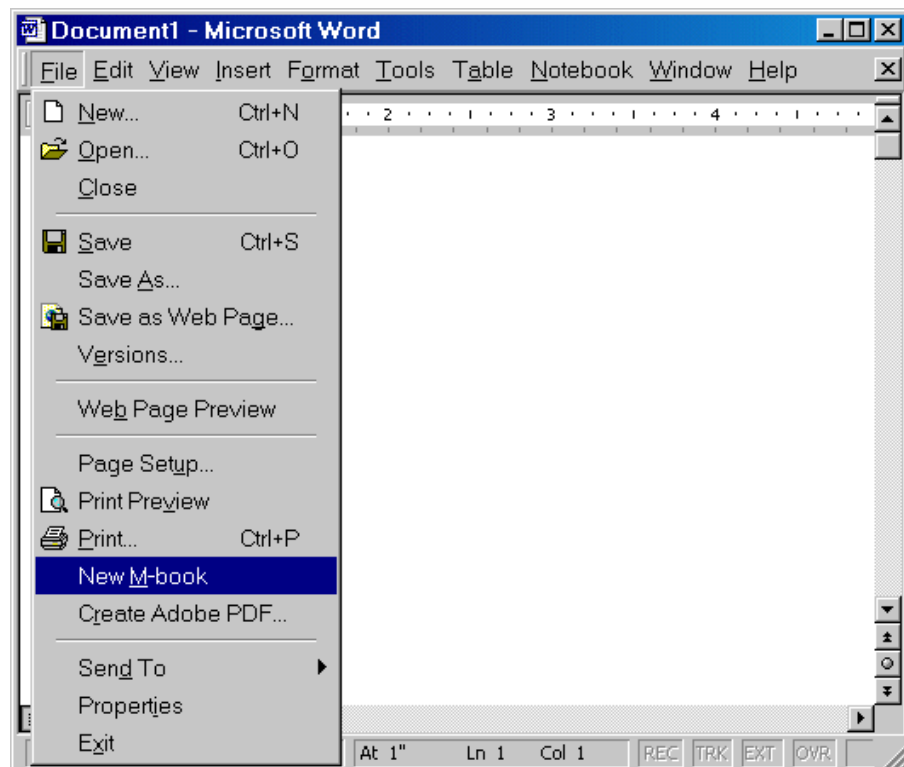
Opening an Existing MATLAB Notebook

You can use the `notebook` command to open an existing MATLAB Notebook, as shown in the following code, where *filename* is the notebook you want to open.

```
notebook filename
```


Alternatively, you can double-click a notebook file in a Windows file management tool, such as Explorer.

When you double-click a notebook, Microsoft Word opens it and starts MATLAB if it is not already running. The **Notebook** menu appears on the Word menu bar and **New M-book** appears on the **File** menu, as shown in the figure that follows.



Converting a Word Document to a MATLAB Notebook

To convert a Word document to a MATLAB Notebook, you insert it into a notebook file, as described in the steps that follow. Choose the set of steps that corresponds to the version of Word you are using:

- “Microsoft Word 2002, or 2003” on page 12-8

- “Microsoft Word 2007” on page 12-8

Microsoft Word 2002, or 2003.

- 1 Create a MATLAB Notebook.

For details, see “Creating or Opening a MATLAB Notebook” on page 12-2.

- 2 From the **Insert** menu, select **File**.
- 3 Select the file you want to convert.
- 4 Click **OK**.

Microsoft Word 2007.

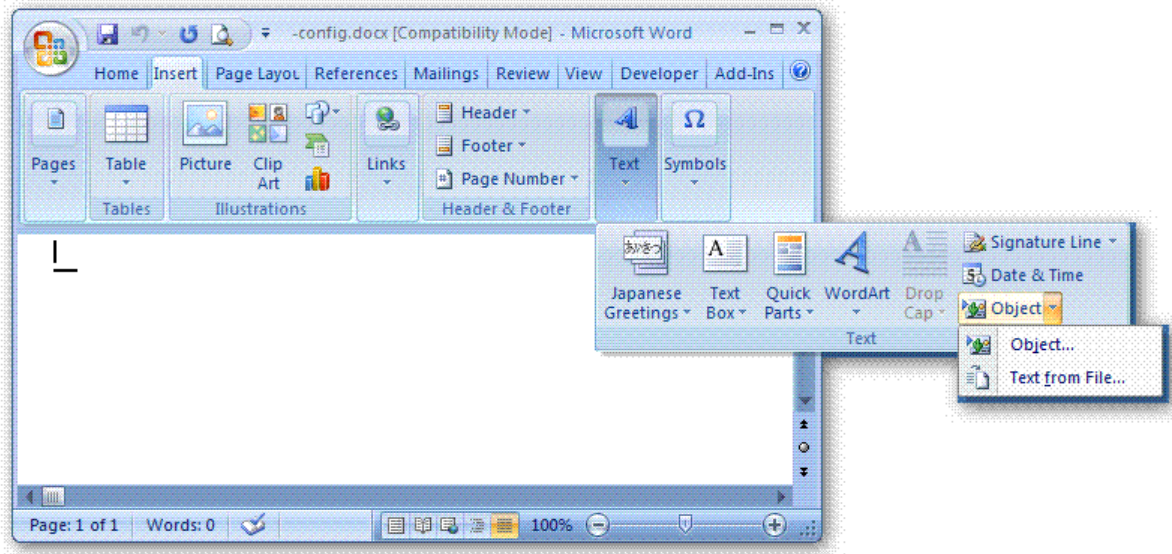
- 1 Create a MATLAB Notebook.

For details, see “Creating or Opening a MATLAB Notebook” on page 12-2.

- 2 From the **Insert** tab, in the **Text** group, click the arrow next to **Object**.
- 3 Select **Text from File**, as shown in the image that follows.

The Insert File dialog box opens.

- 4 In the Insert File dialog box, select the file that you want to convert, and then click **OK**.



Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Entering Commands in a MATLAB Notebook

Note A good way to learn how to use notebook is to open the sample MATLAB Notebook, `Readme.doc`, and try out the various techniques described in this section. You can find this file in the `matlabroot/notebook/pc` folder.

You enter MATLAB commands in a notebook 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 MATLAB Notebook.

```
a = magic(3)
```

To execute the MATLAB magic command in this document, follow the steps described in these sections:

- “Defining MATLAB Commands as Input Cells for a MATLAB Notebook” on page 12-12
- “Evaluating MATLAB Commands in a MATLAB Notebook” on page 12-17

MATLAB displays the output of the command in the Word document in an output cell.

Protecting the Integrity of Your Workspace in MATLAB Notebooks

When you work on more than one MATLAB Notebook in a single word-processing session, note that:

- Each notebook uses the same “copy” of MATLAB.
- All notebooks share the same workspace.

If you use the same variable names in more than one notebook, data used in one notebook can be affected by another notebook. You can protect the integrity of your workspace by specifying the `clear` command as the first autoinit cell in the notebook.

Ensuring Data Consistency in MATLAB Notebooks

You can think of a MATLAB Notebook as a sequential record of a MATLAB session. When executed in order, from the first MATLAB command to the last, the notebook accurately reflects the relationships among these commands.

If, however, you change an input cell or output cell as you refine your notebook, it can contain inconsistent data. Input cells that depend on either the contents or the results of the changed cells do not automatically recalculate when you make a change.

When working on a notebook, consider selecting **Evaluate M-book** periodically to ensure that your notebook data is consistent. You can also use calc zones to isolate related commands in a section of the notebook, and

then use **Evaluate Calc Zone** to execute only those input cells contained in the calc zone.

Debugging and MATLAB Notebooks

Do not use debugging functions or the Editor while evaluating cells within a MATLAB Notebook. Instead:

- 1** Complete debugging files from within MATLAB.
- 2** Clear all the breakpoints.
- 3** Access the file using notebook.

If you debug while evaluating a notebook, you can experience problems with MATLAB.

Defining MATLAB Commands as Input Cells for a MATLAB Notebook

In this section...

“Defining Input Cells for a MATLAB Notebook” on page 12-12

“Defining Cell Groups for a MATLAB Notebook” on page 12-13

“Defining Autoinit Input Cells for a MATLAB Notebook” on page 12-14

“Defining Calc Zones for a MATLAB Notebook” on page 12-14

“Converting an Input Cell to Text in a MATLAB Notebook” on page 12-15

For information about evaluating the input cells you define, see “Evaluating MATLAB Commands in a MATLAB Notebook” on page 12-17.

Defining Input Cells for a MATLAB Notebook

To define a MATLAB command in a Word document as an input cell, follow these steps:

- 1 Type the command into the MATLAB Notebook as text. For example,

```
This is a sample MATLAB Notebook.
```

```
a = magic(3)
```

- 2 Position the cursor anywhere in the command, and then 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. This defines the MATLAB command as an input cell:

```
This is a sample MATLAB Notebook.
```

```
[a = magic(3)]
```

Note how the character font of the text in the input cell changes to a bold, dark green color and appears 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 MATLAB Notebook Template” on page 12-23.

Defining Cell Groups for a MATLAB 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 immediately after the group, cell groups are useful when you need several MATLAB commands. For instance, to define a graphic fully.

If you define all the MATLAB commands that produce a graphic as a cell group, and then evaluate that cell group, it 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” on page 12-18 for information about evaluating a cell group. For information about ungrouping a cell group, see “Ungroup Cells” on page 12-37.

Creating a Cell Group

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**.

The selected cells convert into a cell group and cell markers convert to a single pair that surrounds the group:

```
This is a sample cell group.
```

```
[date  
a = magic(3) ]
```

Note the following:

- A cell group cannot contain output cells. If the selection includes output cells, they are deleted.

- A cell group cannot contain text. If the selection includes text, the text appears after the cell group. However, if the text precedes the first input cell in the selection, it remains where it is.
- If you select part or all of an output cell, but not its input cell, the cell group still includes the input cell.

When you create a cell group, it is an input cell, unless its first line is an autoinit cell. In that case, the group is an autoinit cell.

Defining Autoinit Input Cells for a MATLAB Notebook

You can use *autoinit cells* to specify MATLAB commands be evaluated automatically each time a MATLAB Notebook opens. This is a quick and easy way to initialize the workspace. *Autoinit cells* are input cells with the following additional characteristics:

- The autoinit cells evaluate when MATLAB Notebook opens.
- The commands in autoinit cells display in dark blue characters.

Autoinit cells are otherwise identical to input cells.

Creating an Autoinit Cell for a MATLAB Notebook

You can create an autoinit cell in one of the following two ways:

- Enter the MATLAB command as text, then convert the command to an autoinit cell by selecting **Notebook > Define AutoInit Cell**.
- If you already entered the MATLAB command as an input cell, you can convert the input cell to an autoinit cell. Either select the input cell or position the cursor in the cell, then select **Notebook > Define AutoInit Cell**.

See “Evaluating MATLAB Commands in a MATLAB Notebook” on page 12-17 for information about evaluating autoinit cells.

Defining Calc Zones for a MATLAB Notebook

You can partition a MATLAB Notebook into self-contained sections, called *calc zones*. A calc zone is a contiguous block of text, input cells, and output

cells. Section breaks appear before and after the section, defining 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. A notebook can contain any number of calc zones.

Note Using calc zones does not affect the scope of the variables in a notebook. Variables used in one calc zone are accessible to all calc zones.

Creating a Calc Zone

After you create the text and cells that you want to include in the calc zone, define the calc zone by following these steps:

- 1** Select the input cells and text you want to include in the calc zone.
- 2** Select **Notebook > Define Calc Zone**.

Note Select an input cell and its output cell in their entirety to include them in the calc zone.

See “Evaluating a Calc Zone” on page 12-20 for information about evaluating a calc zone.

Converting an Input Cell to Text in a MATLAB Notebook

To convert an input cell (or an autoint cell or a cell group) to text, follow these steps:

- 1** Select the input cell with the mouse or position the cursor in the input cell.
- 2** Select **Notebook > Undefine Cells** or press **Alt+U**.

When the cell converts to text, the cell contents reformat according to the Microsoft Word Normal style. For more information about MATLAB Notebook styles, see “Modifying Styles in the MATLAB Notebook Template” on page 12-23. When you convert an input cell to text, the corresponding output cell also converts to text.

Evaluating MATLAB Commands in a MATLAB Notebook

In this section...

“Evaluating Input Commands” on page 12-17

“Evaluating Cell Groups” on page 12-18

“Evaluating a Range of Input Cells” on page 12-20

“Evaluating a Calc Zone” on page 12-20

“Evaluating an Entire MATLAB Notebook” on page 12-20

“Using a Loop to Evaluate Input Cells Repeatedly” on page 12-21

“Converting Output Cells to Text” on page 12-22

“Deleting Output Cells” on page 12-22

Evaluating Input Commands

After you define a MATLAB command as an input cell, or as an autoinit cell, you can evaluate it in your MATLAB Notebook. Use the following steps to define and evaluate a MATLAB command:

- 1 Type the command into the notebook as text. For example:

```
This is a sample MATLAB Notebook
```

```
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**.

The MATLAB command becomes an input cell. For example:

```
This is a sample MATLAB Notebook
```

```
[a = magic(3)]
```

- 3 Specify the input cell you want to evaluate by selecting it with the mouse or by placing the cursor in it. Then select **Notebook > Evaluate Cell** or press **Ctrl+Enter**.

The input cell is evaluated and the results display in an output cell immediately following the input cell. If there is already an output cell, its contents refresh or update, wherever the output cell is in the notebook. For example:

```
This is a sample MATLAB Notebook.
```

```
[a = magic(3) ]
```

```
[a =  
      8      1      6  
      3      5      7  
      4      9      2 ]
```

The text in the output cell is blue and is enclosed within cell markers. Cell markers are bold, gray brackets. They differ from the brackets that enclose matrices in their size and weight. Error messages appear in red. For information about changing these default formats, see “Modifying Styles in the MATLAB Notebook Template” on page 12-23.

Evaluating Cell Groups

You evaluate a cell group the same way you evaluate an input cell (because a cell group is an input cell), as follows:

- 1 Position the cursor anywhere in the cell or in its output cell.
- 2 Select **Notebook > Evaluate Cell** or press **Ctrl+Enter**.

For information about creating a cell group, see “Defining Cell Groups for a MATLAB Notebook” on page 12-13.

When MATLAB evaluates a cell group, the output for all commands in the group appears in a single output cell. By default, the output cell appears immediately after the cell group the first time the cell group is evaluated. If

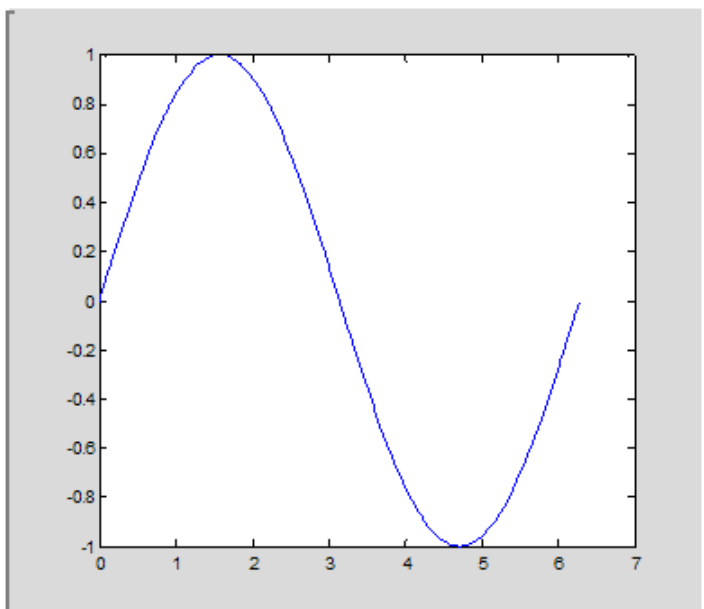
you evaluate a cell group that has an existing output cell, the results appear in that output cell, wherever it is located in the MATLAB Notebook.

Note Text or numeric output always comes first, regardless of the order of the commands in the group.

The following illustration shows a cell group and the figure created when you evaluate the cell group.

This is a sample Notebook with a cell group

```
[t = 0:pi/100:2*pi;  
y = sin(t);  
plot(t,y) ]
```



Evaluating a Range of Input Cells

To evaluate more than one MATLAB command contained in different but contiguous input cells, follow these steps:

- 1 Select the range of cells that includes the input cells you want to evaluate. You can include text that surrounds input cells in your selection.
- 2 Select **Notebook > Evaluate Cell** or press **Ctrl+Enter**.

Each input cell in the selection evaluates, new output cells appear or existing ones are replaced.

Evaluating a Calc Zone

To evaluate a calc zone, follow these steps:

- 1 Position the cursor anywhere in the calc zone.
- 2 Select **Notebook > Evaluate Calc Zone** or press **Alt+Enter**.

For information about creating a calc zone, see “Defining Calc Zones for a MATLAB Notebook” on page 12-14.

By default, the output cell appears immediately after the calc zone the first time the calc zone is evaluated. If you evaluate a calc zone with an existing output cell, the results appear in the output cell wherever it is located in the MATLAB Notebook.

Evaluating an Entire MATLAB Notebook

To evaluate an entire MATLAB Notebook, either select **Notebook > Evaluate M-book** or press **Alt+R**.

Evaluation begins at the top of the notebook, regardless of the cursor position and includes each input cell in the file. As it evaluates the file, notebook inserts new output cells or replaces existing output cells.

Controlling Execution of Multiple Commands

When you evaluate an entire MATLAB Notebook, and an error occurs, evaluation continues. If you want to stop evaluation if an error occurs:

- 1 Select **Notebook > Notebook Options**.

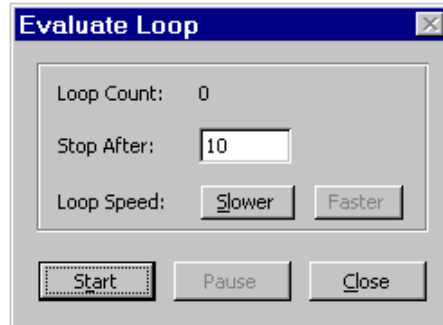
The **Notebook Options** dialog box opens.

- 2 Select the **Stop evaluating on error** check box, and then click **OK**.

Using a Loop to Evaluate Input Cells Repeatedly

To evaluate a sequence of MATLAB commands repeatedly, follow these steps:

- 1 Use the mouse to select the input cells, including any text or output cells located between them.
- 2 Select **Notebook > Evaluate Loop** or press **Alt+L**. The **Evaluate Loop** dialog box displays.



- 3 Enter the number of times you want to evaluate the selected commands in the **Stop After** field, then click **Start**. The button changes to **Stop**. Command evaluation begins, and the number of completed iterations appears 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

You can convert an output cell to text by undefining cells. If the output is numeric or textual, the cell markers disappear and the cell contents convert to text according to the Microsoft Word Normal style. If the output is graphical, the cell markers disappear and the graphic dissociates from its input cell, but contents of the graphic do not change.

Note Undefining an output cell does not affect the associated input cell.

To undefine an output cell, follow these steps:

- 1 Select the output cell you want to undefine.
- 2 Select **Notebook > Undefine Cells** or press **Alt+U**.

Deleting Output Cells

To delete output cells, follow these steps:

- 1 Select an output cell, using the mouse, or place the cursor in the output cell.
- 2 Select **Notebook > Purge Selected Output Cells** or press **Alt+P**.

If you select a range of cells, all the output cells in the selected range disappear, but any associated input cells remain intact.

Printing and Formatting a MATLAB Notebook

In this section...

“Printing a MATLAB Notebook” on page 12-23

“Modifying Styles in the MATLAB Notebook Template” on page 12-23

“Choosing Loose or Compact Format” on page 12-24

“Controlling Numeric Output Format” on page 12-25

“Controlling Graphic Output” on page 12-25

Printing a MATLAB Notebook

You can print all or part of a MATLAB Notebook by doing one of the following, depending on the version of Microsoft Word you are using:

- In Microsoft Word 2002, 2003 — Select **File > Print**.
- In Microsoft Word 2007 — Select **Microsoft Office Button > Print**

Word follows these rules when printing MATLAB Notebook cells and graphics:

- Cell markers do not print.
- Input cells, autoinit cells, and output cells (including error messages) print 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 MATLAB Notebook Template

You can control the appearance of the text in your MATLAB Notebook by modifying the predefined styles stored in the notebook template, `m-book.dot`. These styles control the appearance of text and cells. By default, notebooks use the Word Normal style for all other text.

For example, if you print a notebook 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 notebook on a grayscale printer, these cells appear as shades of gray. To print these cells using black type, modify the color of the Input, Output, AutoInit, and Error styles in the notebook template.

The following table describes the default styles used by MATLAB Notebook. If you modify styles, you can use the information in the table to help you return the styles to their original settings. For general information about using styles in Word documents, see the Word documentation.

Style	Font	Size	Weight	Color
Normal	Times New Roman	10 points	N/A	Black
AutoInit	Courier New	10 points	Bold	Dark blue
Error	Courier New	10 points	Bold	Red
Input	Courier New	10 points	Bold	Dark green
Output	Courier New	10 points	N/A	Blue

When you change a style, Word applies the change to all characters in the notebook that use that style and gives you the option to change the template. Be cautious about changing the template. If you choose to apply the changes to the template, you affect all new notebooks that you create using the template. See the Word documentation for more information.

Choosing Loose or Compact Format

You can specify whether a blank line appears between the input and output cells by selecting the loose or compact format, as follows:

- 1** Select **Notebook > Notebook Options**.
- 2** In the **Notebook Options** dialog box, select either **Loose** or **Compact**.
Loose format adds an empty line. Compact format does not.
- 3** Click **OK**.

Note Changes you make using the **Notebook Options** dialog box take effect for output generated *after* you click **OK**. To affect existing input or output cells, reevaluate the cells.

Controlling Numeric Output Format

To change how numeric output displays, follow these steps:

- 1 Select **Notebook > Notebook Options**.
- 2 In the **Notebook Options** dialog box, select a format from the **Numeric Format** list. These settings correspond to the choices available with the MATLAB format command.
- 3 Click **OK**.

Note Changes you make using the **Notebook Options** dialog box take effect for output generated *after* you click **OK**. To affect existing input or output cells, reevaluate the cells.

Controlling Graphic Output

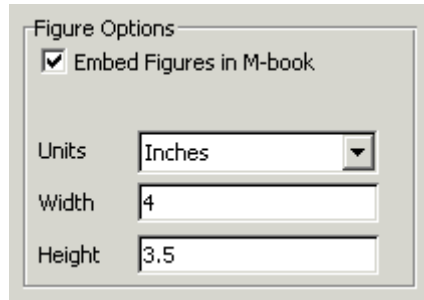
This section describes how to control several aspects of the graphic output produced by MATLAB commands in a MATLAB Notebook, including

- “Embedding Graphic Output in a MATLAB Notebook” on page 12-25
- “Suppressing Graphic Output for Individual Input Cells” on page 12-26
- “Adjusting Graphic Output” on page 12-27

Embedding Graphic Output in a MATLAB Notebook

By default, graphic output is embedded in a MATLAB Notebook. To display graphic output in a separate figure window, follow these steps:

- 1 Select **Notebook > Notebook Options**.
- 2 In the **Notebook Options** dialog box, clear the **Embed Figures in M-book** check box.



3 Click **OK**.

Note Embedded figures do not include Handle Graphics objects generated by the `uicontrol` and `uimenu` functions.

Whether to embed a figure in the MATLAB Notebook is determined by the value of the figure object's `Visible` property. If the value of the property is `off`, the figure embeds in the notebook. If the value of this property is `on`, all graphic output appears in the current figure window.

Suppressing Graphic Output for Individual Input Cells

If an input or autoint cell generates figure output that you want to suppress, follow these steps:

- 1** Place the cursor in the input cell.
- 2** Select **Notebook > Toggle Graph Output for Cell**.

Graphic output from the cell does not appear, and the string `(no graph)` appears after the input cell.

To allow graphic output for a cell, repeat the procedure. The `(no graph)` marker disappears and graphic output from the cell appears.

Note **Toggle Graph Output for Cell** overrides the **Embed Figures in M-book** option, if that option is set.

Adjusting Graphic Output

To set the default size of embedded graphics in a MATLAB Notebook, follow these steps:

- 1** Select **Notebook > Notebook Options**.
- 2** In the **Notebook Options** dialog box, use the **Units**, **Width** and **Height** fields to set the size of graphics generated by the notebook.
- 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, reevaluate the cells.

Change the size of an existing embedded figure by selecting the figure, clicking the left mouse button anywhere in the figure, and then dragging the resize handles of the figure. If you resize an embedded figure using its resize handles and then regenerate the figure, its size reverts to its original size.

To crop graphic output, or add white space around it, follow the instructions for performing these tasks in Microsoft Word. See the Microsoft Word help for details.

Configuring MATLAB notebook

After you install MATLAB Notebook software, but before you begin using it, specify that Word can use the notebook macros, and then configure notebook. (The notebook program installs as part of the MATLAB installation process on Microsoft Windows platforms. For more information, see the MATLAB installation documentation.)

To specify that Word can use the notebook macros:

- In Word 2002, and 2003 do either of the following:
 - Set the macro security level to medium: in Word, select **Tools > Macros > Security**, and in the resulting dialog box, choose **Medium**.
 - After running 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 MATLAB Notebook features, but still maintain a high security level for other macros you use in Word.
- In Word 2007, follow the Word help instructions in the topic entitled “Enable or disable macros in Office documents.”

To configure MATLAB Notebook software:

- 1** Type notebook in the MATLAB Command Window.

MATLAB opens a dialog box that indicates notebook has not been configured and asks if you want to configure it now.

- 2** Click **Yes**.

MATLAB configures MATLAB Notebook software and issues the following messages in the Command Window:

```
Welcome to the utility for setting up the MATLAB Notebook  
for interfacing MATLAB to Microsoft Word
```

```
Setup complete
```

```
Warning: MATLAB is now an automation server
```

When MATLAB configures the software, it:

- a** Accesses the Microsoft Windows system registry to locate Microsoft Word and the Word templates folder. It also identifies the version of Word.
- b** Copies the `m-book.dot` template to the Word templates folder
The MATLAB Notebook software supports Word versions 2002, 2003, and 2007.

If you have previously configured the software, typing `notebook` in the MATLAB Command Window, starts Microsoft Word and creates a MATLAB Notebook. The Command Window displays the message `Warning: MATLAB is now an automation server.`

If you suspect a problem with the current configuration, you can explicitly configure the software by typing:

```
notebook ('-setup')
```

Notebook Feature Reference

In this section...
“Bring MATLAB to Front” on page 12-30
“Define Autoinit Cell” on page 12-31
“Define Calc Zone” on page 12-31
“Define Input Cell” on page 12-32
“Evaluate Calc Zone” on page 12-32
“Evaluate Cell” on page 12-33
“Evaluate Loop” on page 12-34
“Evaluate M-Book” on page 12-34
“Group Cells” on page 12-34
“Hide Cell Markers” on page 12-35
“Notebook Options” on page 12-35
“Purge Selected Output Cells” on page 12-35
“Toggle Graph Output for Cell” on page 12-36
“Undefine Cells” on page 12-36
“Ungroup Cells” on page 12-37

This section provides reference information about each of the MATLAB Notebook features, listed alphabetically. To use these features, select them from the **Notebook** menu in Microsoft Word. (In Word 2007, the **Notebook** menu is on the **Add-Ins** tab.)

Bring MATLAB to Front

Bring MATLAB to Front brings the MATLAB Command Window to the foreground.

Define Autoinit Cell

Define AutoInit Cell creates an autoinit cell by converting the current paragraph, selected text, or input cell. An autoinit cell is an input cell that is automatically evaluated whenever you open a MATLAB Notebook.

Result

If you select this feature while the cursor is in a paragraph of text, the entire paragraph converts to an autoinit cell. If you select this feature while text is selected, the text converts to an autoinit cell. If you select this feature while the cursor is in an input cell, the input cell converts to an autoinit cell.

Format

The autoinit cell format uses 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 a MATLAB Notebook” on page 12-14.

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

A calc zone is defined as a Word document section. Section breaks appear 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” on page 12-20. 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 autoint cell. An input cell contains a MATLAB command.

Result

If you select this feature while the cursor is in a paragraph of text, the entire paragraph converts to an input cell. If you select this feature while text is selected, the text converts to an input cell. If you select this feature while the cursor is in an autoint cell, the autoint cell converts to an input cell.

Format

The text appears enclosed in cell markers and the cell uses 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 a MATLAB Notebook” on page 12-12. For information about evaluating input cells, see “Evaluating MATLAB Commands in a MATLAB Notebook” on page 12-17.

Evaluate Calc Zone

Evaluate Calc Zone sends the input cells in the current calc zone to MATLAB for evaluation. The current calc zone is the Word section that contains the cursor.

Result

As each input cell evaluates, it generates an output cell. When you evaluate an input cell for which there is no output cell, the output cell appears immediately after the input cell that generated it. If you evaluate an input cell for which there is an output cell, the results in the output cell are replaced, wherever it is in the MATLAB Notebook.

See Also

For more information, see “Evaluating a Calc Zone” on page 12-20.

Evaluate Cell

Evaluate Cell sends the current input cell or cell group to MATLAB for evaluation. An input cell contains a MATLAB command. A cell group is a single, multiline input cell that contains more than one MATLAB command. The output or an error message displays in an output cell.

Result

If you evaluate an input cell for which there is no output cell, the output cell appears immediately after the input cell that generated it. If you evaluate an input cell for which there is an output cell, the results in the output cell are replaced, wherever that cell is in the MATLAB Notebook. 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 can 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 in a MATLAB Notebook” on page 12-17. For information about evaluating the entire MATLAB Notebook, see “Evaluating an Entire MATLAB Notebook” on page 12-20.

Evaluate Loop

Evaluate Loop evaluates the selected input cells repeatedly.

For more information, see “Using a Loop to Evaluate Input Cells Repeatedly” on page 12-21.

Evaluate M-Book

Evaluate M-book evaluates the entire MATLAB Notebook, sending all input cells to MATLAB for evaluation. Evaluation begins at the top of the MATLAB Notebook regardless of the cursor position.

Result

As each input cell evaluations, it generates an output cell. When you evaluate an input cell for which there is no output cell, the output cell appears immediately after the input cell that generated it. If you evaluate an input cell for which there is an output cell, the results are replaced in the output cell wherever it is in the MATLAB Notebook.

See Also

For more information, see “Evaluating an Entire MATLAB Notebook” on page 12-20.

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, it appears after the cell group. However, if text precedes the first input cell in the group, the text remains before the group.

If you include output cells in the selection, they disappear. If you select all or part of an output cell before selecting this feature, its input cell appears 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 a MATLAB Notebook” on page 12-13. For information about converting a cell group to individual input cells, see “Ungroup Cells” on page 12-37.

Hide Cell Markers

Hide Cell Markers hides cell markers in the MATLAB Notebook.

When you select this feature, it changes to **Show Cell Markers**.

Note Cell markers do not print 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 a MATLAB Notebook” on page 12-23 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” on page 12-22.

Toggle Graph Output for Cell

Toggle Graph Output for Cell suppresses or allows graphic output from an input cell.

If an input or autoint cell generates figure output that you want to suppress, place the cursor in the input cell and choose this feature. The string (no graph) appears 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 disappears. This feature overrides the **Embed Figures in M-book** option, if that option is set in the **Notebook Options** dialog box.

See Also

See “Embedding Graphic Output in a MATLAB Notebook” on page 12-25 and “Suppressing Graphic Output for Individual Input Cells” on page 12-26 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, that cell becomes undefined. Cell markers disappear and the cell reformats according to the Normal style.

If you undefine an input cell, its output cell automatically undefines. However, if you undefine an output cell, its input cell does not undefine. If you undefine an output cell containing an embedded graphic, the graphic remains in the MATLAB Notebook but is no longer associated with an input cell.

See Also

For information about the Normal style, see “Modifying Styles in the MATLAB Notebook Template” on page 12-23. For information about deleting output cells, see “Purge Selected Output Cells” on page 12-35.

Ungroup Cells

Ungroup Cells converts the current cell group into a sequence of individual input cells or autoint cells. If the cell group is an input cell, the cell group converts to input cells. If the cell group is an autoint cell, the cell group converts to autoint cells. The output cell for the cell group disappears.

A cell group is the current cell group if:

- The cursor is in the cell group.
- The cursor is at the end of a line that contains the closing cell marker for the cell group.
- The cursor is in the output cell for the cell group.
- The cell group is selected.

See Also

For information about creating cell groups, see the description of “Defining Cell Groups for a MATLAB Notebook” on page 12-13.

Source Control Interface

The source control interface provides access to your source control system from the MATLAB desktop. Source control systems, also known as version control, revision control, configuration management, and file management systems, are platform dependent—the topics for the Microsoft Windows platforms appear first, followed by the topics for the UNIX platforms.

- “Source Control Interface on Microsoft Windows” on page 13-2
- “Setting Up the Source Control Interface on Microsoft Windows” on page 13-3
- “Checking Files Into and Out of Source Control from the MATLAB Desktop on Microsoft Windows” on page 13-11
- “Additional Source Control Actions on Microsoft Windows” on page 13-14
- “Performing Source Control Actions from the Editor, Simulink, or Stateflow File Menu on Microsoft Windows” on page 13-23
- “Troubleshooting Source Control Problems on Microsoft Windows” on page 13-24
- “Source Control Interface on UNIX Platforms” on page 13-26
- “Specifying the Source Control System on UNIX Platforms” on page 13-27
- “Checking Files Into the Source Control System on UNIX Platforms” on page 13-30
- “Checking Files Out of the Source Control System on UNIX” on page 13-33
- “Undoing the Checkout on UNIX Platforms” on page 13-36

Source Control Interface on Microsoft Windows

If you use source control systems to manage your files, you can interface with the systems to perform source control actions from within the MATLAB, Simulink, and Stateflow® products. Use menu items in the MATLAB, Simulink, or Stateflow products, or run functions in the MATLAB Command Window to interface with your source control systems.

The source control interface on Windows works with any source control system that conforms to the Microsoft Common Source Control standard, Version 1.1. If your source control system does not conform to the standard, use a Microsoft Source Code Control API wrapper product for your source control system so that you can interface with it from the MATLAB, Simulink, and Stateflow products.

This documentation uses the Microsoft® Visual SourceSafe® software as an example. Your source control system might use different terminology and not support the same options or might use them in a different way. Regardless, you should be able to perform similar actions with your source control system based on this documentation.

Perform most source control interface actions from the Current Folder browser. You can also perform many of these actions for a single file from the MATLAB Editor, a Simulink model window, or a Stateflow chart window—for more information, see “Performing Source Control Actions from the Editor, Simulink, or Stateflow File Menu on Microsoft Windows” on page 13-23. Another way to access many of the source control actions is with the `verctrl` function.

Setting Up the Source Control Interface on Microsoft Windows

In this section...

“Create Projects in Source Control System” on page 13-3

“Specify Source Control System with MATLAB Software” on page 13-5

“Register Source Control Project with MATLAB Software” on page 13-7

“Add Files to Source Control” on page 13-10

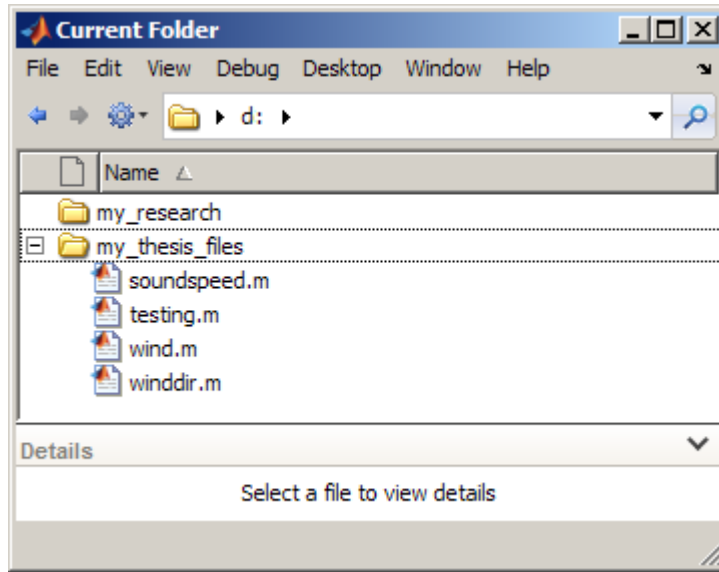
Create Projects in Source Control System

In your source control system, create the projects that your folders and files will be associated with.

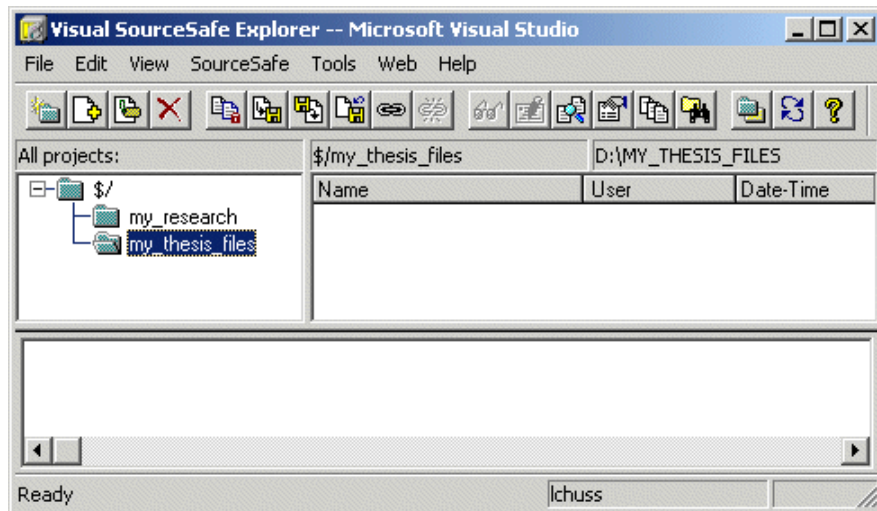
All files in a folder must belong to the same source control project. Be sure the working folder for the project in the source control system specifies the correct path to the folder on disk.

Example of Creating Source Control Project

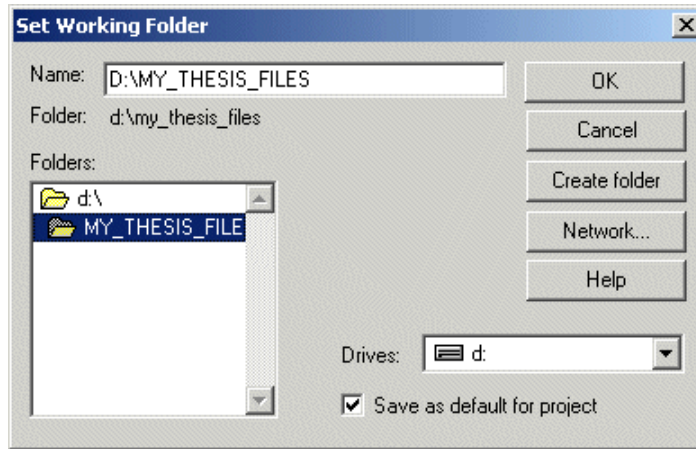
This example uses the project `my_thesis_files` in Microsoft Visual SourceSafe. This illustration of the Current Folder browser shows the path to the folder on disk, `C:\my_thesis_files`.



The following illustration shows the example project in the source control system.



To set the working folder in Microsoft Visual SourceSafe for this example, select `my_thesis_files`, right-click, select **Set Working Folder** from the context menu, and specify `D:\my_thesis_files` in the resulting dialog box.

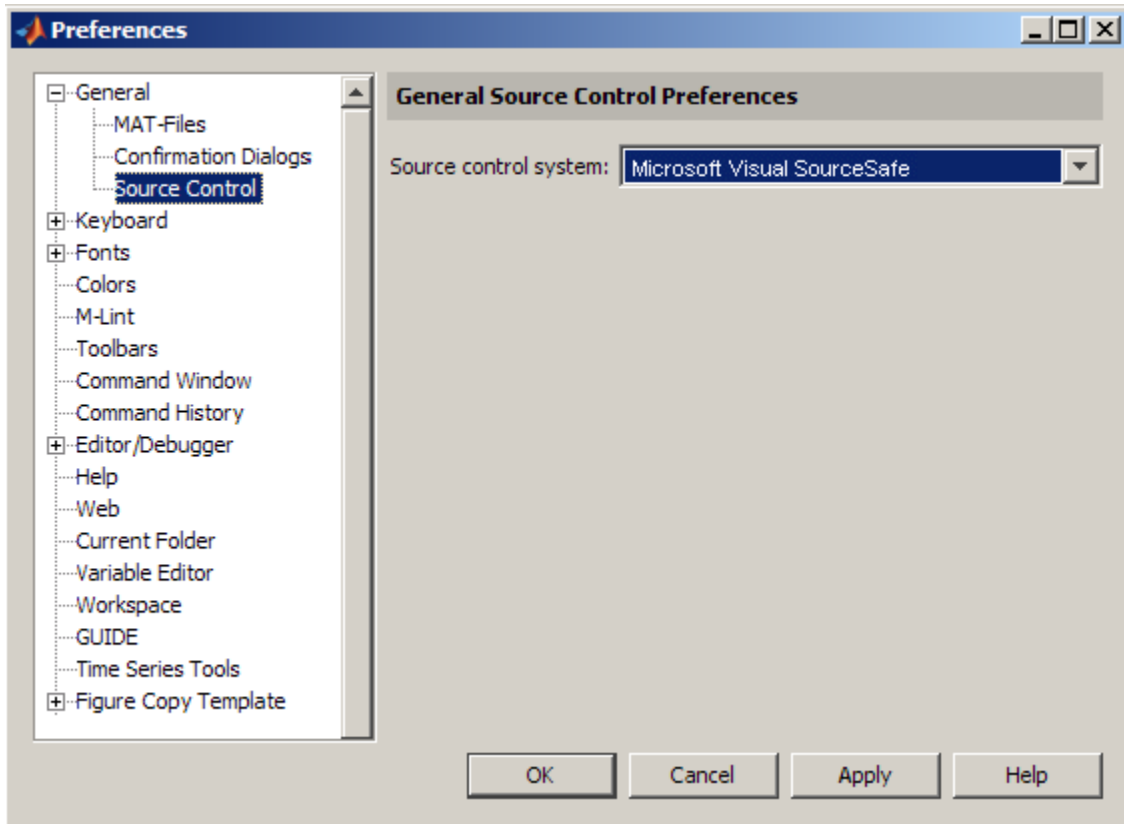


Specify Source Control System with MATLAB Software

In MATLAB, specify the source control system you want to access. Select **File > Preferences > General > Source Control**.

The currently selected system is shown in the Preferences dialog box. The list includes all installed source control systems that support the Microsoft Common Source Control standard.

Select the source control system you want to interface with and click **OK**.



MATLAB remembers preferences between sessions, so you only need to perform this action again when you want to access a different source control system.

Source Control with 64-Bit Versions of MATLAB

If you run a 64-bit version of MATLAB and want MATLAB to interface with your source control system, your source control system must be 64-bit compliant. If you have a 32-bit source control system, or if you have a 64-bit source control system running in 32-bit compatibility mode, MATLAB cannot use it. In that event, MATLAB displays a warning about the problem in the Source Control preference pane.

Function Alternative for Specifying Source Control System

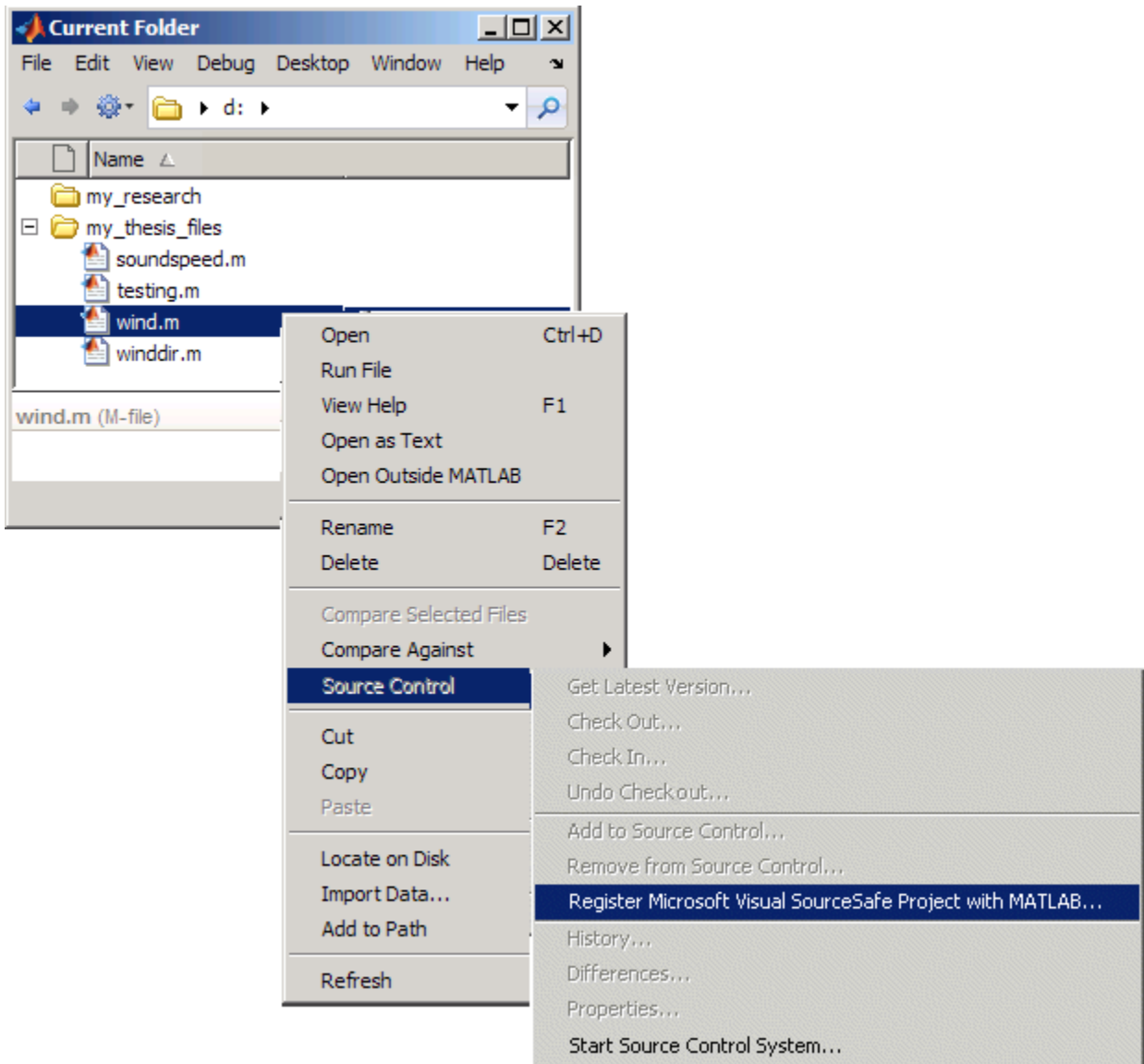
A function alternative to select a source control system is not available, but you can list all available source control systems using `verctrl` with the `all_systems` argument. Use `cmopts` to display the name of the currently selected source control system.

Register Source Control Project with MATLAB Software

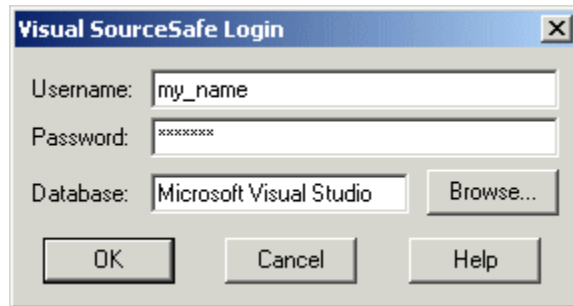
Register a source control system project with a folder in MATLAB, that is, associate a source control system project with a folder and all files in that folder. Do this only one time for any file in the folder, which registers all files in that folder:

- 1 In the MATLAB Current Folder browser, select a file that is in the folder you want to associate with a project in your source control system. For example, select `D:\my_thesis_files\wind.m`. This will associate all files in the `my_thesis_files` folder.
- 2 Right-click, and from the context menu, select **Source Control > Register Name_of_Source_Control_System Project with MATLAB**. The **Name_of_Source_Control_System** is the source control system you selected using preferences as described in “Specify Source Control System with MATLAB Software” on page 13-5.

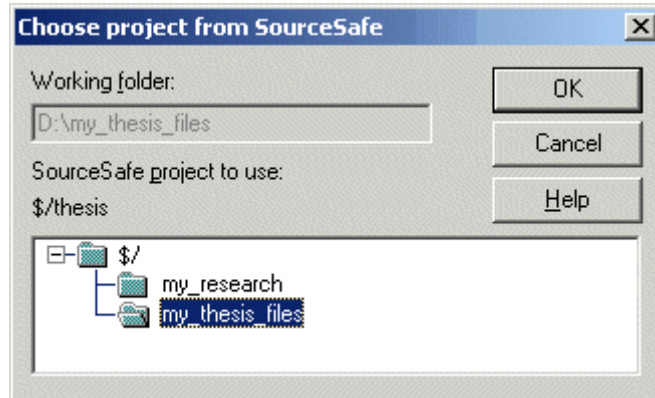
The following example shows Microsoft Visual SourceSafe.



- 3 In the resulting **Name_of_Source_Control_System Login** dialog box, provide the user name and password you use to access your source control system, and click **OK**.



- 4 In the resulting **Choose project from Name_of_Source_Control_System** dialog box, select the source control system project to associate with the folder and click **OK**. This example shows my_thesis_files.

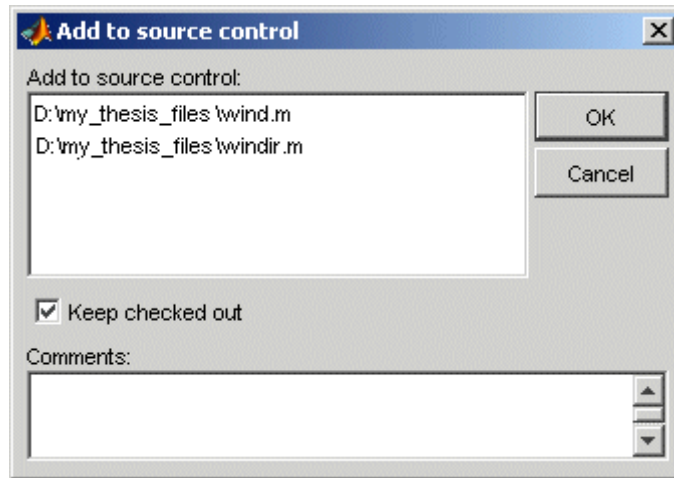


The selected file, its folder, and all files in the folder, are associated with the source control system project you selected. For the example, MATLAB associates all files in D:\my_thesis_files with the source control project my_thesis_files.

Add Files to Source Control

Add files to the source control system. Do this only once for each file:

- 1 In the Current Folder browser, select files you want to add to the source control system.
- 2 Right-click, and from the context menu, select **Source Control > Add to Source Control**.
- 3 The resulting **Add to source control** dialog box lists files you selected to add. You can add text in the **Comments** field. If you expect to use the files soon, select the **Keep checked out** check box (which is selected by default). Click **OK**.



If you try to add an unsaved file, the file is automatically saved upon adding.

Function Alternative

The function alternative is `verctrl` with the `add` argument.

Checking Files Into and Out of Source Control from the MATLAB Desktop on Microsoft Windows

In this section...

“Check Files Into Source Control” on page 13-11

“Check Files Out of Source Control” on page 13-12

“Undoing the Checkout” on page 13-13

Before checking files into and out of your source control system from the MATLAB desktop, be sure to set up your system for use with MATLAB as described in “Setting Up the Source Control Interface on Microsoft Windows” on page 13-3.

Check Files Into Source Control

After creating or modifying files using MATLAB software or related products, check the files into the source control system by performing these steps:

- 1** In the Current Folder browser, select the files to check in. A file can be open or closed when you check it in, but it must be saved, that is, it cannot contain unsaved changes.
- 2** Right-click, and from the context menu, select **Source Control > Check In**.
- 3** In the resulting **Check in file(s)** dialog box, you can add text in the **Comments** field. If you want to continue working on the files, select the check box **Keep checked out**. Click **OK**.

If a file contains unsaved changes when you try to check it in, you will be prompted to save the changes to complete the checkin. If you did not keep the file checked out and you keep the file open, note that it is a read-only version.

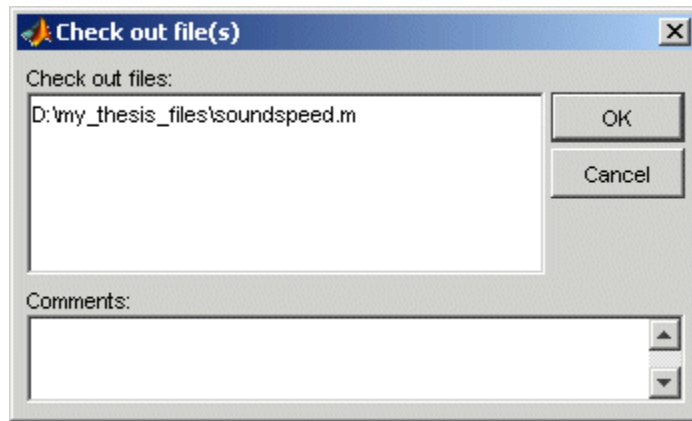
Function Alternative

The function alternative is `verctrl` with the `checkin` argument.

Check Files Out of Source Control

From MATLAB, to check out the files you want to modify, perform these steps:

- 1 In the Current Folder browser, select the files to check out.
- 2 Right-click, and from the context menu, select **Source Control > Check Out**.
- 3 The resulting **Check out file(s)** dialog box lists files you selected to check out. Enter comment text in the **Comments** field, which appears if your source control system supports comments on checkout. Click **OK**.



After checking out a file, make changes to it in MATLAB or another product, and save the file. For example, edit a file in the Editor.

If you try to change a file without first having checked it out, the file is read-only, as seen in the title bar, and you will not be able to save any changes. This protects you from accidentally overwriting the source control version of the file.

If you end the MATLAB session, the file remains checked out. You can check in the file from within MATLAB during a later session, or folder from your source control system.

Function Alternative

The function alternative is `verctrl` with the `checkout` argument.

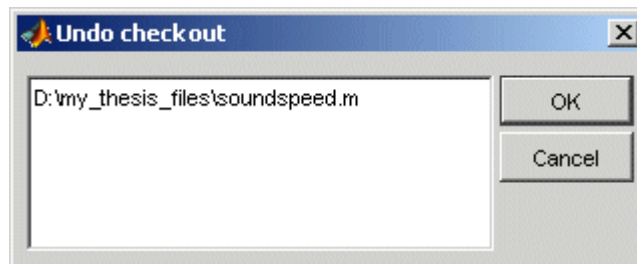
Undoing the Checkout

You can undo the checkout for files. The files remain checked in, and do not have any of the changes you made since you last checked them out. To save any changes you have made since checking out a particular file select **File > Save As**, and supply a different file name before you undo the checkout.

To undo a checkout, follow these steps:

- 1 In the MATLAB Current Folder browser, select the files for which you want to undo the checkout.
- 2 Right-click, and from the context menu, select **Source Control > Undo Checkout**.

The MATLAB **Undo checkout** dialog box opens, listing the files you selected.



- 3 Click **OK**.

Function Alternative

The function alternative is `verctrl` with the `undocheckout` argument.

Additional Source Control Actions on Microsoft Windows

In this section...
“Getting the Latest Version of Files for Viewing or Compiling” on page 13-14
“Removing Files from the Source Control System” on page 13-15
“Showing File History” on page 13-16
“Comparing the Working Copy of a File to the Latest Version in Source Control” on page 13-18
“Viewing Source Control Properties of a File” on page 13-20
“Starting the Source Control System” on page 13-21

Getting the Latest Version of Files for Viewing or Compiling

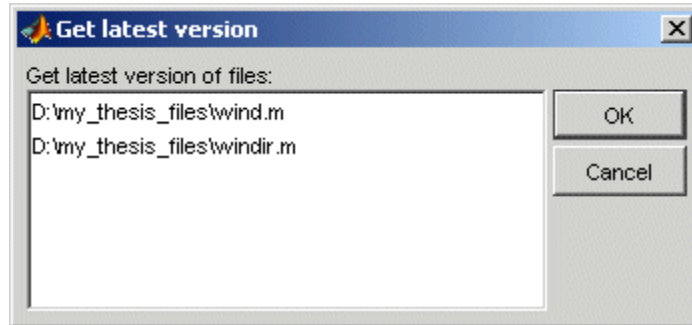
You can get the latest version of a file from the source control system for viewing or running. Getting a file differs from checking it out. When you get a file, it is write protected so you cannot edit it, but when you check out a file, you can edit it.

To get the latest version, follow these steps:

- 1 In the MATLABCurrent Folder browser, select the folders or files that you want to get. If you select files, you cannot select folders too.

- 2 Right-click, and from the context menu, select **Source Control > Get Latest Version**.

The MATLAB Get latest version dialog box opens, listing the files or folders you selected.



- 3 Click **OK**.

You can now open the file to view it, run the file, or check out the file for editing.

Function Alternative

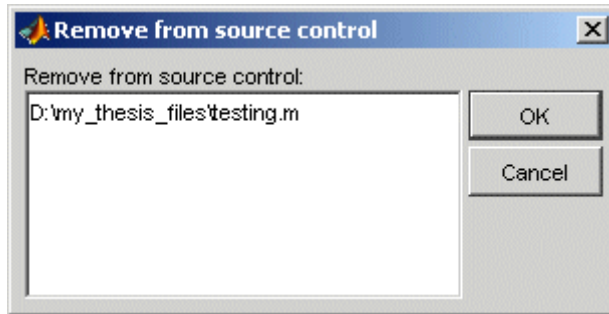
The function alternative is `verctrl` with the `get` argument.

Removing Files from the Source Control System

To remove files from the source control system, follow these steps:

- 1 In the MATLAB Current Folder browser, select the files you want to remove.
- 2 Right-click, and from the context menu, select **Source Control > Remove from Source Control**.

The MATLAB **Remove from source control** dialog box opens, listing the files you selected.



- 3 Click **OK**.

Function Alternative

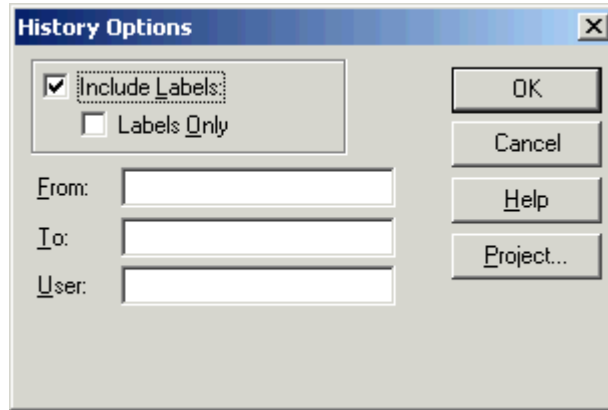
The function alternative is `verctrl` with the `remove` argument.

Showing File History

To show the history of a file in the source control system, follow these steps:

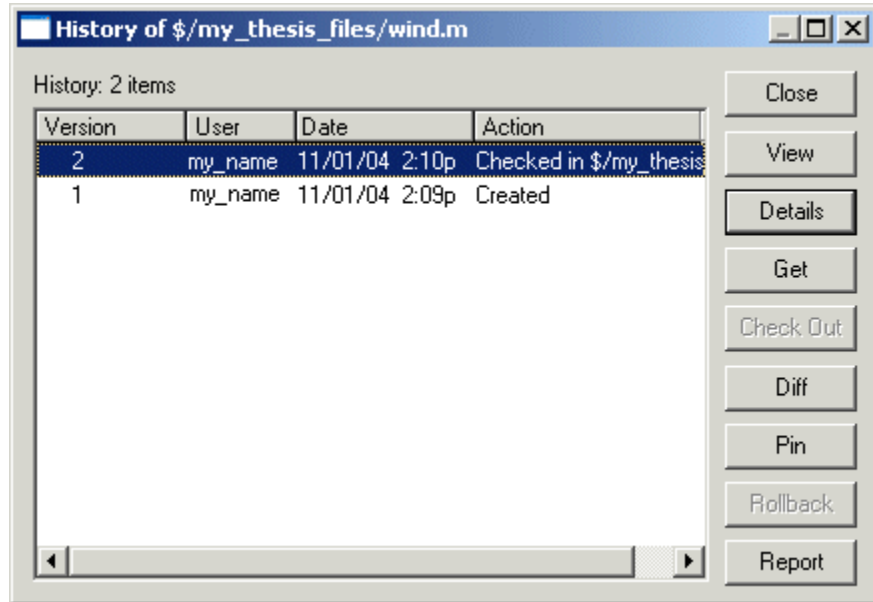
- 1 In the MATLAB Current Folder browser, select the file for which you want to view the history.
- 2 Right-click, and from the context menu, select **Source Control > History**.

A dialog box, which is specific to your source control system, opens. For Microsoft Visual SourceSafe, the **History Options** dialog box opens, as shown in the following example illustration.



- 3 Complete the dialog box to specify the range of history you want for the selected file and click **OK**. For example, enter my_name for **User**.

The history presented depends on your source control system. For Microsoft Visual SourceSafe, the **History** dialog box opens for that file, showing the file's history in the source control system.



Function Alternative

The function alternative is `verctrl` with the `history` argument.

Comparing the Working Copy of a File to the Latest Version in Source Control

You can compare the current working copy of a file with the latest checked-in version of the file in the source control system. This highlights the differences between the two files, showing the changes you made since you checked out the file.

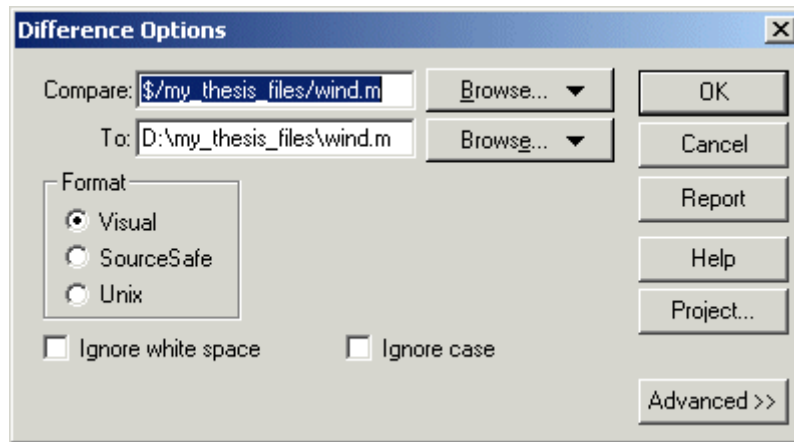
To view the differences, follow these steps:

- 1 In the MATLAB Current Folder browser, select the file for which you want to view differences. This is a file that has been checked out and edited.

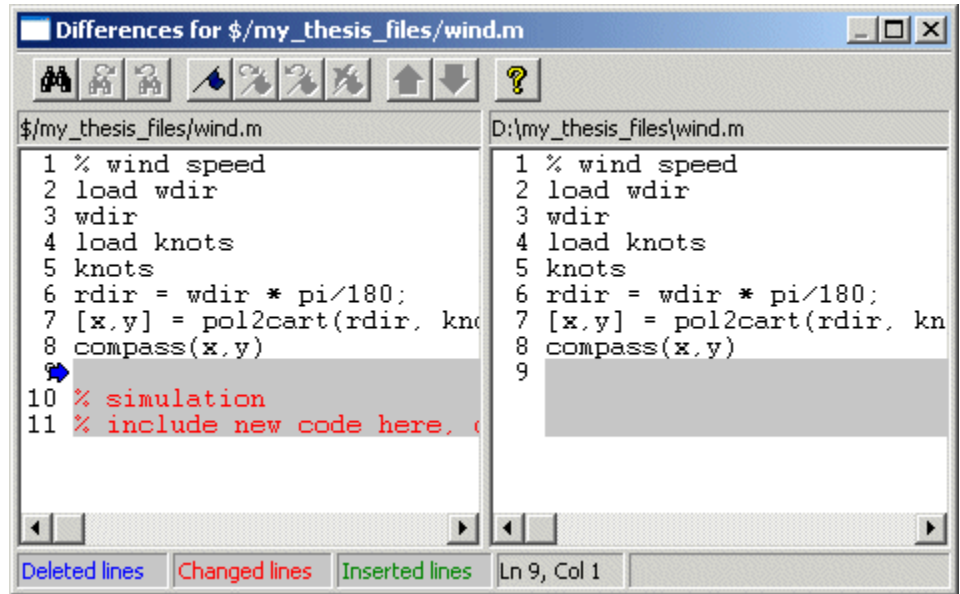
- 2 Right-click, and from the context menu, select **Source Control > Differences**.

A dialog box, which is specific to your source control system, opens. For Microsoft Visual SourceSafe, the **Difference Options** dialog box opens.

- 3 Review the default entries in the dialog box, make any needed changes, and click **OK**. The following example is for Microsoft Visual SourceSafe.



The method of presenting differences depends on your source control system. For Microsoft Visual SourceSafe, the **Differences for** dialog box opens. This highlights the differences between the working copy of the file and the latest checked-in version of the file.



Function Alternative

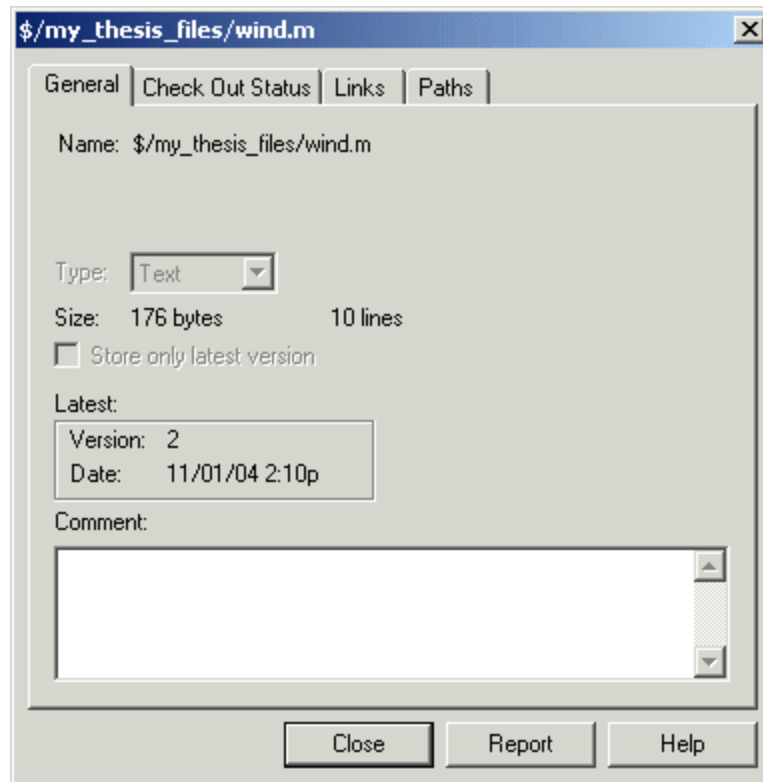
The function alternative is `verctrl` with the `showdiff` or `isdiff` argument.

Viewing Source Control Properties of a File

To view the source control properties of a file, follow these steps:

- 1 In the MATLAB Current Folder browser, select the file for which you want to view properties.
- 2 Right-click, and from the context menu, select **Source Control > Properties**.

A dialog box, which is specific to your source control system, opens. The following example shows the Microsoft Visual SourceSafe properties dialog box.



Function Alternative

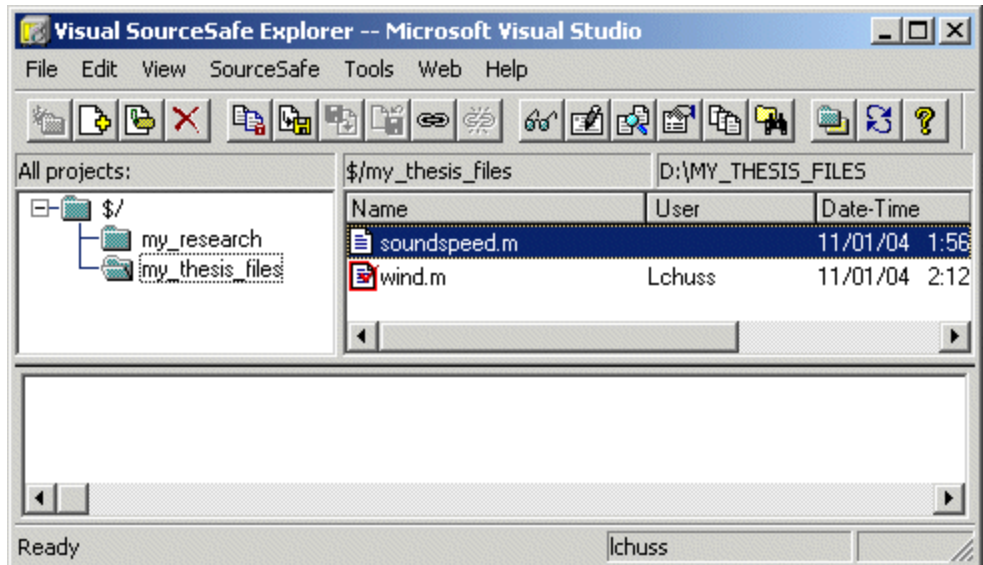
The function alternative is `verctrl` with the `properties` argument.

Starting the Source Control System

All the MATLAB source control actions automatically start the source control system to perform the action, if the source control system is not already open. If you want to start the source control system from MATLAB without performing a specific action source control action,

- 1 Right-click any folder or file in the MATLAB Current Folder browser
- 2 From the context menu, select **Source Control > Start Source Control System**.

The interface to your source control system opens, showing the source control project associated with the current folder in MATLAB. The following example shows the Microsoft Visual SourceSafe Explorer interface.



Function Alternative

The function alternative is `verctrl` with the `runsc` argument.

Performing Source Control Actions from the Editor, Simulink, or Stateflow File Menu on Microsoft Windows

You can create or open a file in the Editor, the Simulink or Stateflow products and perform most source control actions from their **File > Source Control** menus, rather than from the Current Folder browser. Following are some differences in the source control interface process when you use the Editor, Simulink, or Stateflow:

- You can perform actions on only one file at time.
- Some of the dialog boxes have a different icon in the title bar. For example, the **Check out file(s)** dialog box uses the MATLAB Editor icon instead of the MATLAB icon.
- You cannot add a new (Untitled) file, but must instead first save the file.
- You cannot register projects from the Simulink or Stateflow products. Instead, register a project using the Current Folder browser, as described in “Register Source Control Project with MATLAB Software” on page 13-7.

Troubleshooting Source Control Problems on Microsoft Windows

In this section...

“Source Control Error: Provider Not Present or Not Installed Properly” on page 13-24

“Restriction Against @ Character” on page 13-25

“Add to Source Control Is the Only Action Available” on page 13-25

“More Solutions for Source Control Problems” on page 13-25

Source Control Error: Provider Not Present or Not Installed Properly

In some cases, MATLAB software recognizes your source control system but you cannot use source control features for MATLAB. Specifically, when you select **File > Preferences > General > Source Control**, or run `cmopts`, MATLAB lists your source control system, but you cannot perform any source control actions. Only the **File > Source Control > Start Source Control System** menu item is available, and when you select it, MATLAB displays this error:

```
Source control provider is not present or not installed properly.
```

Often, this error occurs because a registry key that MATLAB requires from the source control application is not present. Make sure this registry key is present:

```
HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider\  
InstalledSCCProviders
```

The registry key refers to another registry key that is similar to

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SourceSafe\ScsServerPath
```

This registry key has a path to a DLL-file in the file system. Make sure the DLL-file exists in that location. If you are not familiar with registry keys, ask your system administrator for help.

If this does not solve the problem and you use Microsoft Visual SourceSafe, try running a client setup for your source control application. When SourceSafe is installed on a server for a group to use, each machine client can run a setup but is not required to do so. However, some applications that interface with SourceSafe, including MATLAB, require you to run the client setup. Run the client setup, which should resolve the problem.

If the problem persists, access source control outside of MATLAB.

Restriction Against @ Character

Some source control systems, such as Perforce® and Synergy™, reserve the @ character. Perforce, for example, uses it as a revision specifier. Therefore, you might experience problems if you use these source control systems with MATLAB files and folders that include the @ character in the folder or file name.

You might be able to work around this restriction by quoting nonstandard characters in file names, such as with an escape sequence, which some source control systems allow. Consult your source control system documentation or technical support resources for a workaround.

Add to Source Control Is the Only Action Available

To use source control features for a file in the Simulink or Stateflow products, the file's source control project must first be registered with MATLAB. When a file's source control project is *not* registered with MATLAB, all **File > Source Control** menu items are disabled except **Add to Source Control**. You can select **Add to Source**, which registers the project with MATLAB, or you can register the project using the Current Folder browser, as described in "Register Source Control Project with MATLAB Software" on page 13-7. You can then perform source control actions for all files in that project (folder).

More Solutions for Source Control Problems

The latest solutions for problems interfacing MATLAB with a source control system appear on the MathWorks Web page for support at <http://www.mathworks.com/support/>. Search Solutions and Technical Notes for "source control."

Source Control Interface on UNIX Platforms

If you use a source control system to manage your files, you can check MATLAB program files and Simulink models, and Stateflow charts into and out of the source control system from within the MATLAB, Simulink, and Stateflow products.

The source control interface supports four popular source control systems, as well as a custom option:

- ClearCase® software from IBM® Rational®
- Concurrent Version System (CVS)
- ChangeMan® and PVCS® software from Serena®
- Revision Control System (RCS)
- Custom option — Allows you to build your own interface if you use a different source control system. For details, see the reference page for `customverctrl`.

Perform source control interface actions for a single file using menu items in the MATLAB Editor, a Simulink model window, or a Stateflow chart window. To perform source control actions on multiple files, use the Current Folder browser. Alternatively, run source control functions in the Command Window, which provide some options not supported with the menu items.

Specifying the Source Control System on UNIX Platforms

In this section...

“MATLAB Desktop Alternative” on page 13-27

“Function Alternative” on page 13-28

“Setting a View and Checking Out a Folder with ClearCase Software on UNIX Platforms” on page 13-29

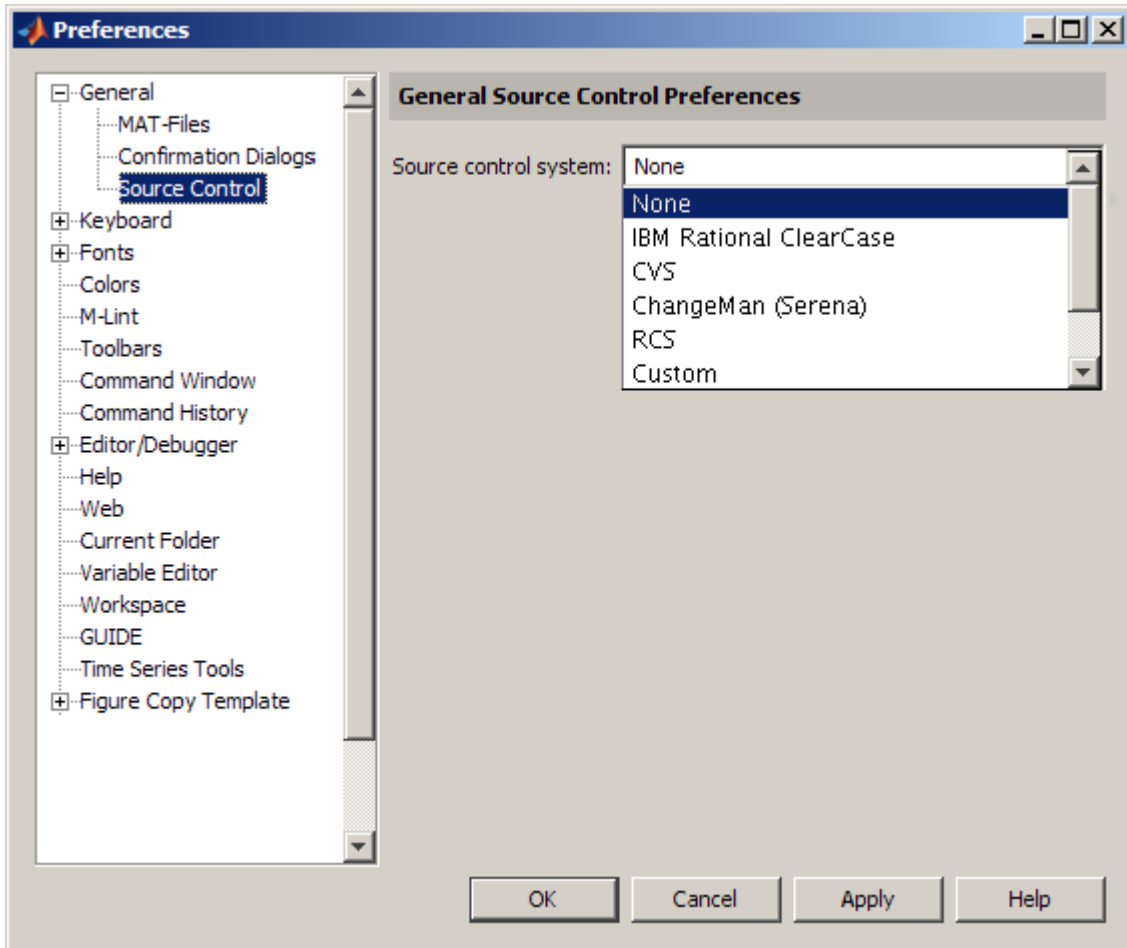
MATLAB Desktop Alternative

To specify the source control system you want to access, select

File > Preferences > General > Source Control.

The currently selected system is shown in the Preferences dialog box. The default selection is None.

Select the source control system with which you want to interface and click **OK.**



MATLAB remembers preferences between sessions, so you only need to perform this action when you want to access a different source control system.

Function Alternative

A function alternative to select a source control system is not available, but you can list the currently selected source control system by running `cmopts`.

Setting a View and Checking Out a Folder with ClearCase Software on UNIX Platforms

If you use ClearCase software on a UNIX platform, perform the following from ClearCase:

- 1** Set a view.
- 2** Check out the folder that contains files you want to save, check in, or check out.

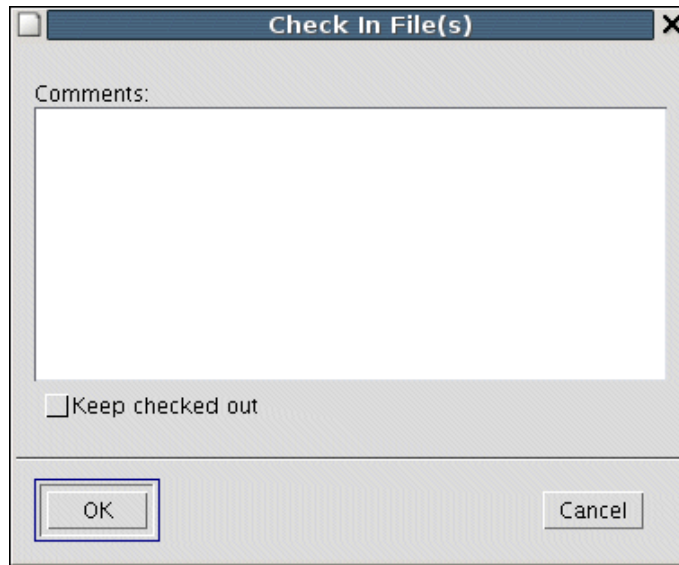
You can now use the MATLAB, Simulink, or Stateflow source control interfaces to ClearCase software.

Checking Files Into the Source Control System on UNIX Platforms

In this section...
“Checking In One or More Files Using the Current Folder Browser” on page 13-30
“Checking In One File Using the Editor, or the Simulink or Stateflow Products” on page 13-31
“Function Alternative” on page 13-32

Checking In One or More Files Using the Current Folder Browser

- 1** From the Current Folder browser, select the file or files to check in. A file can be open or closed when you check it in, but it must be saved, that is, it cannot contain unsaved changes.
- 2** Right-click, and from the context menu, select **Source Control > Check In**.
- 3** In the resulting **Check in file(s)** dialog box, you can add text in the **Comments** field. If you want to continue working on the files, select the check box **Keep checked out**. Click **OK**.



The files are checked into the source control system. If any file contains unsaved changes when you try to check it in, you will be prompted to and must then save the changes to complete the checkin.

An error appears in the Command Window if a file is already checked in.

If you did not keep a file checked out and you keep that file open, note that it is a read-only version.

Checking In One File Using the Editor, or the Simulink or Stateflow Products

- 1 From the Editor, or the Simulink or Stateflow products, with the file open and saved, select **File > Source Control > Check In**.
- 2 In the resulting **Check in file(s)** dialog box, you can add text in the **Comments** field. If you want to continue working on the files, select the check box **Keep checked out**. Click **OK**.

Function Alternative

Use `checkin` to check files into the source control system. The files can be open or closed when you use `checkin`. The `checkin` function takes this form:

```
checkin({'file1', 'file2', ...}, 'comments', 'comment_text', ...  
       'option', 'value')
```

For `file`, use the complete path and include the file extension. You must supply the `comments` argument and a comments string with `checkin`.

Use the `option` argument to

- Check in a file and keep it checked out — set the `lock` option value to `on`.
- Check in a file even though it has not changed since the previous check in — set the `force` option value to `on`.

The `comments` argument and the `lock` and `force` options apply to all files checked in.

Example Using `checkin` Function

To check in the file `clock.m` with the comment `Adjustment for leap year`, type

```
checkin('\myserver\myfiles\clock.m', 'comments', ...  
       'Adjustment for leap year')
```

For other examples, see the reference page for `checkin`.

Checking Files Out of the Source Control System on UNIX

In this section...

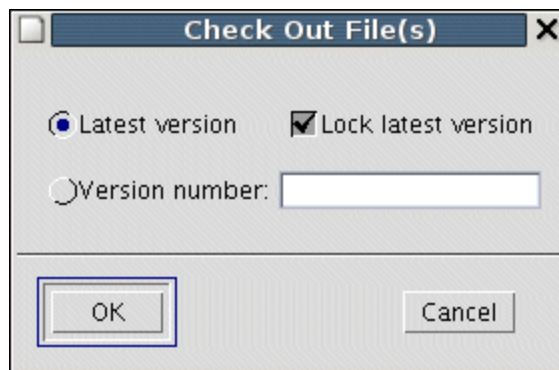
“Checking Out One or More Files Using the Current Folder Browser” on page 13-33

“Checking Out a Single File Using the Editor, or the Simulink or Stateflow Products” on page 13-34

“Function Alternative” on page 13-34

Checking Out One or More Files Using the Current Folder Browser

- 1 In the Current Folder browser, select the file or files to check out.
- 2 Right-click, and from the context menu, select **Source Control > Check Out**. The **Check out file(s)** dialog box opens.



- 3 Complete the dialog box:
 - a To check out the versions that were most recently checked in, select the **Latest version** option.
 - b To check out a specific version of the files, select the **Version number** option and type the version number in the field.

- c To prevent others from checking out the files while you have them checked out, select **Lock latest version**. To check out read-only versions of the file, clear **Lock latest version**.

4 Click **OK**.

An error appears in the Command Window if a file is already checked out.

After checking out files, make changes to them using MATLAB software or another software product, and save the files. For example, edit a file in the Editor.

If you try to change a file without first having checked it out, the file is read-only, as seen in the title bar, and you will not be able to save any changes. This protects you from accidentally overwriting the source control version of the file.

If you end the MATLAB session, the file or files remain checked out. You can check in files from within MATLAB during a later session, or directly from your source control system.

Checking Out a Single File Using the Editor, or the Simulink or Stateflow Products

- 1** Open the MATLAB program file, Simulink model, or Stateflow chart you want to check out. The title bar indicates the file is read-only.
- 2** Select **File > Source Control > Check Out**. The **Check out file(s)** dialog box opens.
- 3** Complete the dialog box as described in step of “Checking Out One or More Files Using the Current Folder Browser” on page 13-33, and click **OK**.

Function Alternative

Use `checkout` to check out a file from the source control system. You can check out multiple files at once and specify checkout options. The `checkout` function takes this form:

```
checkout({'file1','file2', ...},'option','value')
```

For `filen`, use the complete path and include the file extension.

Use the option argument to

- Check out a read-only version of the file — set the `lock` option value to `off`.
- Check out the file even if you already have it checked out — set the `force` option value to `on`.
- Check out a specific version of the file — use the `revision` option, and assign the version number to the `value` argument.

The options apply to all files being checked out. The files can be open or closed when you use `checkout`.

Example Using checkout Function—Check Out a Specific Version of a File

To check out the 1.1 version of the file `clock.m`, type

```
checkout('\myserver\myfiles\clock.m', 'revision', '1.1')
```

For other examples, see the reference page for `checkout`.

Undoing the Checkout on UNIX Platforms

In this section...

“Impact of Undoing a File Checkout” on page 13-36

“Undoing the Checkout for One or More Files Using the Current Folder Browser” on page 13-36

“Undoing the Checkout for a Single File Using the Editor, or the Simulink or Stateflow Products” on page 13-36

“Function Alternative” on page 13-37

Impact of Undoing a File Checkout

When you undo the checkout for a file, the file remains checked in, and does not have any of the changes you made since you checked it out. To save any changes you have made since checking out a file, select **File > Save As**, and supply a different file name before you undo the checkout. Undo the checkout using the Current Folder browser for one or more files. For only one file, you can also use the Editor, or the Simulink or Stateflow products.

Undoing the Checkout for One or More Files Using the Current Folder Browser

- 1 In the MATLAB Current Folder browser, select the file or files for which you want to undo the checkout.
- 2 Right-click, and from the context menu, select **Source Control > Undo Checkout**. MATLAB undoes the checkout.

An error appears in the Command Window if the file is not checked out.

Undoing the Checkout for a Single File Using the Editor, or the Simulink or Stateflow Products

- 1 Open the MATLAB program file, Simulink model, or Stateflow chart for which you want to undo the checkout.

- 2 Select **File > Source Control > Undo Checkout**. MATLAB undoes the checkout.

Function Alternative

The `undocheckout` function takes this form:

```
undocheckout({'file1','file2', ...})
```

Use the complete path for `file` and include the file extension. For example, to undo the checkout for the files `clock.m` and `calendar.m`, type

```
undocheckout({'\myserver\myfiles\clock.m',...  
'\myserver\myfiles\calendar.m'})
```


Internationalization

- “How the MATLAB Process Uses Locale Settings” on page 14-2
- “Setting the Locale” on page 14-4
- “Troubleshooting I18n Messages and Settings” on page 14-9

How the MATLAB Process Uses Locale Settings

A *locale* is part of the user environment definition. It defines language, territory, and *codeset*, which is a coded character set. The MATLAB process uses the user-specified locale name on all platforms. MATLAB also reads the user-specified *UI language name*, and uses it to select localized resources in the specified language. By using this feature, you can select localized resources in US-English. The user-specified UI language setting also controls language and country settings of the Sun™ Java Virtual Machine (JVM) software.

Consider the following when choosing your locale settings. To see what your current settings are, use the instructions in “Setting Locale on Windows Platforms” on page 14-4, “Setting Locale on Linux Platforms” on page 14-6, or “Setting Locale on Macintosh Platforms” on page 14-7.

- **Default Locale Setting** — If the user-specified locale is not supported, MATLAB uses the default locale `en_US.US-ASCII`.
- **UI Language Setting** — The UI language setting should be set to either the same language as the user-specified locale or to **US-English**. Otherwise, non-7-bit ASCII characters might not display properly.
- **Supported Character Set** — MATLAB supports the character set specified by the user locale setting. However, MATLAB might not properly handle character codes greater than 2 bytes.
- **Script Compatibility** — Non-7-bit ASCII characters in MATLAB scripts created with one locale setting might not be compatible with a different locale setting.

For example, if you create a script with the `ja_JP.UTF-8` locale setting, the script might not be compatible when executed on a platform with the `ja_JP.eucJP` locale setting.

- **Numeric Format Uses C Locale** — MATLAB reads the user locale for all categories except for the `LC_NUMERIC` category. This category controls numeric data formatting and parsing. MATLAB always sets `LC_NUMERIC` to the C locale. For more information, see “Numbers Display Period for Decimal Point” on page 14-10.
- **Platform-Specific Localized Formats** — MATLAB usually uses platform-neutral localized formats and rules. You can, however, use

the operating system short date format to display files, as described in “Customizing the Column Display” on page 7-19.

Windows Platform-Specific Behavior

The user locale and system locale must be the same value on the Microsoft Windows platform. If these values are not the same, you might see garbled text or incorrect characters. For information on controlling these settings, see “Setting Locale on Windows Platforms” on page 14-4.

Macintosh Platform-Specific Behavior

On the Apple Macintosh OS X platform, MATLAB reads the user locale setting and the user UI language setting. For information on controlling these settings, see “Setting Locale on Macintosh Platforms” on page 14-7. MATLAB ignores the LANG environment variable and the Terminal application locale setting.

MATLAB automatically chooses a codeset for each combination of language and territory on the Mac OS X platform. If you customize the locale setting on OS X, MATLAB ignores the customized portion.

Setting the Locale

In this section...
“Setting Locale on Windows Platforms” on page 14-4
“Setting Locale on Linux Platforms” on page 14-6
“Setting Locale on Macintosh Platforms” on page 14-7

Setting Locale on Windows Platforms

MATLAB software uses the *system locale* and *user locale* on Windows platforms:

- “Setting User Locale on Windows 7 Platforms” on page 14-4
- “Setting System Locale on Windows 7 Platforms” on page 14-4
- “Setting User Locale on Windows Vista Platforms” on page 14-5
- “Setting System Locale on Windows Vista Platforms” on page 14-5
- “Setting User Locale on Windows XP Platforms” on page 14-6
- “Setting System Locale on Windows XP Platforms” on page 14-6

Setting User Locale on Windows 7 Platforms

- 1 Select **Start** -> **Control Panel** -> **Clock, Language, and Region** -> **Regional and Language**.
- 2 Open **Formats** tab.
- 3 Select a target locale from the **Format:** drop-down list.

Setting System Locale on Windows 7 Platforms

- 1 Select **Start** -> **Control Panel** -> **Clock, Language, and Region** -> **Regional and Language**.
- 2 Open **Administrative** tab.

- 3** Look in the **Language for non-Unicode programs** section.
- 4** Click **Change system locale...** button.
- 5** Select a target locale from the **Current system locale:** drop-down list.
- 6** Reboot the system.

Note When you change the system locale, you must reboot your system; otherwise, you might see unexpected locale-setting behaviors.

Setting User Locale on Windows Vista Platforms

- 1** Select **Start -> Control Panel -> Regional and Language Options**.
- 2** Open **Formats** tab.
- 3** Select an item from the drop-down list.

Setting System Locale on Windows Vista Platforms

- 1** Select **Start -> Control Panel -> Regional and Language Options**.
- 2** Open **Administrative** tab.
- 3** Click **Change system locale...** button.
- 4** Select an item from the drop-down list.
- 5** Reboot the system.

Note When you change the system locale, you must reboot your system; otherwise, you might see unexpected locale-setting behaviors.

Setting User Locale on Windows XP Platforms

- 1 Select **Start -> Control Panel -> Regional and Language Options**.
- 2 Open **Regional Options** tab.
- 3 Select an item from the drop-down list.

Setting System Locale on Windows XP Platforms

- 1 Select **Start -> Control Panel -> Regional and Language Options**.
- 2 Open **Advanced** tab.
- 3 Select an item from the drop-down list.
- 4 Reboot the system.

Note When you change the system locale, you must reboot your system; otherwise, you might see unexpected locale-setting behaviors.

Setting Locale on Linux Platforms

Linux platforms manage locale settings with six *locale categories*. These are the same categories used by C standard library functions.

The following locale categories are available:

- LC_CTYPE controls character data manipulations.
- LC_COLLATE controls character collation/sorting operations.
- LC_TIME controls date/time data formatting or parsing.
- LC_NUMERIC controls numeric data formatting or parsing.
- LC_MONETARY controls monetary data formatting or parsing.
- LC_MESSAGES controls the user UI language.

Setting User Locale and User UI Language

Use the LANG environment variable to specify a single locale for all locale categories. The locale specified with this variable might be partially or entirely over-written by other environment variables.

Use the environment variables LC_CTYPE, LC_COLLATE, LC_TIME, LC_NUMERIC, and LC_MONETARY to specify a locale for a particular category.

Use the LC_ALL environment variable to over-write all locales specified with other environment variables. If a single locale has to be set to all locale categories, use LANG instead of LC_ALL.

Configuring Fonts to Display Asian Characters

On some Linux systems, to properly display Asian characters in the MATLAB Desktop, you must configure the font with the Java Runtime Environment (JRE™). If you previously configured fonts for your system, you must also make the configuration changes for the JRE distributed with MATLAB.

To configure, make a symbolic link between your font and the MATLAB font fallback directory. For example, to use the Kochi font, at the Linux system prompt type:

```
ln -s /usr/share/fonts/truetype/kochi
    matlabroot/sys/java/jre/glnxa64/jre/lib/fonts/fallback
```

where *matlabroot* is the folder where you installed MATLAB.

Alternatively, edit the fontconfig.properties file. See your Java documentation for information about this file.

Setting Locale on Macintosh Platforms

The Macintosh OS X platform manages the user locale setting and the user UI language setting.

Setting User Locale

- 1 Select System Preferences ->International

2 Open **Formats** tab

3 Select an item from the **Region** pop-up menu

Setting UI Language

1 Select **System Preferences ->International**

2 Open **Language** tab

3 Drag an item to the top of the **Languages** list

Troubleshooting I18n Messages and Settings

The term *I18n* is an abbreviation for internationalization, where 18 stands for the number of letters between the i and the n.

In this section...

“Asian Characters Incorrectly Displayed on Linux Systems” on page 14-9

“Characters Incorrectly Displayed on Windows Systems” on page 14-10

“datenum Might Not Return Correct Value” on page 14-10

“Numbers Display Period for Decimal Point” on page 14-10

“MATLAB Displays Messages in English” on page 14-11

“File or Folder Names Incorrectly Displayed” on page 14-11

Asian Characters Incorrectly Displayed on Linux Systems

On some Linux systems, to properly display Asian characters in the MATLAB Desktop, you must configure the font with the Java Runtime Environment (JRE). If you previously configured fonts for your system, you must also make the configuration changes for the JRE distributed with MATLAB.

To configure, make a symbolic link between your font and the MATLAB font fallback directory. For example, to use the Kochi font, at the Linux system prompt type:

```
ln -s /usr/share/fonts/truetype/kochi
    matlabroot/sys/java/jre/glnxa64/jre/lib/fonts/fallback
```

where *matlabroot* is the folder where you installed MATLAB.

Alternatively, edit the `fontconfig.properties` file. See your Java documentation for information about this file.

Characters Incorrectly Displayed on Windows Systems

The user locale and system locale must be the same value on the Microsoft Windows platform. If these values are not the same, you might see garbled text or incorrect characters. For information on controlling these settings, see “Setting Locale on Windows Platforms” on page 14-4.

datenum Might Not Return Correct Value

To ensure the correct calculation of functions using date values associated with files and folders, replace `datenum` function calls with the use of the `dir` function `datenum` field.

For example, look at the modification date of your MATLAB `license.txt` file:

```
cd(matlabroot)
f=dir('license.txt')
```

MATLAB displays information similar to:

```
f =
      name: 'license.txt'
      date: '10-May-2007 17:48:22'
     bytes: 5124
     isdir: 0
    datenum: 7.3317e+005
```

If your code uses the `date` field of the `dir` command, similar to:

```
n=datenum(f.date);
```

replace it with the `datenum` field:

```
n=f.datenum;
```

Numbers Display Period for Decimal Point

MATLAB uses a period for a decimal point, regardless of the format specified by the user locale. For example, the value of `pi` can be displayed as `3,1415` or `3.1415`, depending on the format used by a locale. MATLAB always displays `3.1415`.

The MATLAB language reserves the use of commas to the cases described in the “Comma — ,” topic of the Programming Fundamentals Symbol Reference.

MATLAB Displays Messages in English

MATLAB displays messages in English, regardless of the UI language setting, except when running in a Japanese Microsoft Windows environment.

File or Folder Names Incorrectly Displayed

On Windows and Linux platforms, characters used in file or folder names must be in the supported character set. See **Supported Character Set** in “How the MATLAB Process Uses Locale Settings” on page 14-2.

On Macintosh platforms, for files and folders used by MATLAB, characters in the file or folder name must be in the 7-bit ASCII character set.

Symbols and Numerics

- , after functions 3-47
- ; after functions 3-47
- % comment
 - creating 9-42
- % comment symbol 9-41
- ! function 3-8
 - argument length restrictions 3-9
- %% 9-178
- {% block comment symbol 9-43
- >> prompt in Command Window 3-3
- ... in statements 3-20

A

- absolute path name 7-8
 - copying 7-9
- accelerators
 - Command Window 2-66
- accelerators, keyboard 2-66
- Access Bridge 2-162
- accessibility 2-159
 - documentation 2-160
 - installation 2-162
 - troubleshooting 2-165
- account
 - MathWorks products 2-119
- activate license 2-121
- antialiasing
 - desktop fonts 2-149
- AppleScript
 - running from MATLAB 3-9
- archive files
 - adding files to 7-40
 - creating 7-38 to 7-39
 - extracting files from 7-39
- arrays
 - editing 6-24
 - workspace 6-2
- assistive technology 2-159

- asv 9-85
- autoinit cells
 - converting input cells to 12-31
 - converting to input cells 12-32
 - defining 12-14
- AutoInit style
 - definition of 12-24
- automatic completion of statement
 - Command Window 3-24
 - Editor 9-46
- automatic fix
 - M-Lint 9-112
- autosave files 9-85

B

- back and forward navigation 9-75
- backup files
 - MATLAB Editor autosave and 9-85
- bang (!) function 3-8
- base workspace 6-9
- batch mode for starting MATLAB 1-17
- beep
 - preferences 2-139
- binary files
 - comparing 7-59
- blank line 9-16
- blank spaces in MATLAB commands 3-17
- block comments 9-43
 - extending 9-43
- blue breakpoint icon 9-168
- bold text
 - in published MATLAB code 11-43
 - within cell 11-43
- bookmarks
 - in files in Editor 9-71
 - in Help browser 4-12
- books 4-38
- Boolean searching in Help browser 4-22
- breaking long lines 3-20

- breaking out of a running program 3-8
- breakpoints
 - anonymous functions 9-168
 - blue icon 9-168
 - clearing (removing) 9-160
 - clearing, automatically 9-161
 - conditional 9-166
 - disabling and enabling 9-159
 - multiple per line 9-168
 - running file 9-148
 - setting 9-144
 - types 9-144
- Bring MATLAB to Front 12-30
- browser
 - for Web 2-101
- bugs, reporting to The MathWorks 4-38
- built-in editor 9-4

- C**
- C/C++
 - editing files in Editor 9-8
- caching
 - files 9-84
 - search path 7-6
- calc zones
 - defining 12-14
 - ensuring workspace consistency in MATLAB Notebooks 12-10
 - evaluating 12-20
 - output from 12-20
- callbacks
 - in shortcuts 2-57
- calling from MATLAB 3-8
- capitalization in MATLAB 3-17
- case
 - changing lower to upper in Editor 9-39
 - changing upper to lower in Editor 9-39
- case sensitivity in MATLAB 3-17
- cell arrays
 - editing 6-27
- cell breaks 9-178 11-19
- cell dividers. *See* cell breaks
- cell groups
 - converting to input cells 12-37
 - creating 12-13
 - definition of 12-13
 - evaluating 12-18
 - output from 12-18
- cell highlighting
 - troubleshooting 9-182
- cell markers
 - defined 12-12
 - hiding 12-35
 - printing 12-23
- cell mode 9-175
- cell scripts 9-175
- cell titles 9-180
- cells
 - files and 9-175
- cells in files 9-175
 - removing 9-184
- changing
 - search path 7-75
- character set
 - preference for MAT-files 2-131
- checkin
 - on UNIX platforms 13-32
- checking in files
 - on UNIX platforms 13-30
- checking out files
 - on UNIX platforms 13-33
 - on Windows platforms 13-12
 - undoing on UNIX platforms 13-36
 - undoing on Windows platforms 13-13
- checkout
 - on UNIX platforms 13-34
- class help
 - adjusting wrapping for 9-45
- clc 3-50

- clear 6-9
- ClearCase source control system
 - configuring on UNIX platforms 13-29
- clearing
 - Command Window 3-50
 - variables 6-9
- clicking on multiple items 2-113
- clipboard 2-114
- closing
 - desktop tools 2-8
 - MATLAB 1-23
- code
 - automatically analyzing for warnings and errors 9-107 to 9-108
 - checking 10-22
 - debugging
 - options 9-4
 - code analysis 9-107
 - Code analysis
 - Editor access 9-107
 - Code Analyzer preferences
 - setting 9-124
 - Code Analyzer Report 10-2 10-22
 - checking MATLAB code 10-22
 - code cells 9-10
 - evaluating 9-194
 - files and 9-175
 - publishing and 11-2
 - code cells in files
 - beep 9-195
 - defining 9-178
 - evaluating 9-195
 - evaluating code in 9-196
 - modifying values in 9-197
 - nested 9-185
 - toolbar 9-177
 - code examples 9-2
 - code folding
 - behavior 9-62
 - preferences 9-62
 - viewing code in Tooltip 9-61
 - code folding in files 9-57
 - code folding preferences in files 9-34
 - code iteration 9-175
 - code resources 9-2
 - code samples
 - sample code 9-2
 - collapsing
 - code in files 9-34 9-57
 - Collatz problem 9-142
 - colors
 - general preferences 2-152
 - Help browser 4-32
 - in files 9-53
 - indicators for syntax 3-23
 - preferences in MATLAB 2-150
 - printing MATLAB Notebook 12-23
 - column numbers 9-55
 - columns
 - customizing in Current Folder browser 7-19
 - command flags 1-14
 - Command History
 - about 3-66
 - changing date format in 7-20
 - deleting entries in window 3-76
 - file 3-67
 - find entry by letter 3-70
 - preferences 3-78
 - printing window contents 3-76
 - running functions from window 3-69
 - command history file 3-68
 - command line
 - defined 3-3
 - editing 3-18
 - command name completion
 - Command Window 3-24
 - Editor 9-46
 - command switches 1-14
 - Command Window
 - bringing to front in Notebook 12-30

- clearing 3-50
 - editing in 3-18
 - getting started message bar 3-63
 - paging of output in 3-47
 - preferences 3-60
 - printing contents of 3-50
 - prompt 3-2
 - scroll buffer 3-64
 - width 3-62
- commands
- executing a group of 2-57
 - on multiple lines 3-20
 - to operating system 3-8
- comments
- adding/removing in C/C++ files 9-42
 - adding/removing in Java files 9-42
 - adding/removing with any text editor 9-42
 - adding/removing with Editor 9-41
 - adjusting wrapping for class help 9-45
 - block 9-43
 - color indicators 2-152
 - creating in MATLAB code files 9-40
 - marking up within code cell 11-19
 - multiline statements 9-44
 - using ... (ellipsis) 9-44
 - within a line 9-44
 - wrapping in files 9-45
- comp.soft-sys.matlab 4-40
- comparing
- directories 7-50
 - files 7-50
- comparing working copy to source control version
- on Windows platforms 13-18
- Comparison Tool
- features of 7-63
- completing statements automatically
- Command Window 3-24
 - Editor 9-46
- compression
- MAT-files and Fig-Files 2-131
- conditional breakpoints 9-166
- configuration management
- See* source control system interface 13-1
- configuration, desktop 2-5
- configurations
- reassociating 9-99
 - renaming 9-99
 - See also* publish configurations 9-88
 - See also* run configurations 9-88
- configuring Notebook 12-28
- confirmation dialog boxes
- preferences 2-132
- console mode 3-62
- Contents Report 10-11
- context menus 2-108
- continuation
- long lines 3-20
- continuing long statements 3-20
- conversion
- Word document to MATLAB Notebook 12-7
- copying
- files and folders 7-45
- Coverage Report 10-20
- <CR> 9-16
- crash 1-24
- creating
- files and folders
 - using functions 7-41
- cropping graphics
- in MATLAB Notebooks 12-27
- cssm 4-40
- current folder
- at startup for MATLAB 1-8
 - changing 7-4
 - in MATLAB 7-2
 - viewing 7-4
- Current Folder browser 7-12
- asterisks and 7-19
 - changing date format in 7-20
 - columns 7-19

- details panel 7-21
 - preferences 7-14
 - refresh display 7-15
 - running scripts from 7-48
 - viewing image thumbnails in 7-21

- D**
- data consistency
 - calc zones in MATLAB Notebooks 12-10
 - evaluating MATLAB Notebooks 12-10
 - in MATLAB Notebook 12-10
- data tips
 - example 9-153
- date format
 - changing in Command History window 7-20
 - changing in Current Folder browser 7-20
- dbclear 9-160
- dbstop
 - example 9-148
- deactivate license 2-121
- Debugger 9-1
- debugging
 - code
 - options 9-4
 - ending 9-158
 - example 9-142
 - features 9-141
 - files 9-104
 - Notebook 12-11
 - prompt 9-149
 - stepping 9-150
 - techniques 9-104
 - with unsaved changes 9-165
- decimal places in output 3-48
- defaults
 - preferences for MATLAB 2-124
 - setting in startup file for MATLAB 1-15
- Define Autoinit Cell 12-31
- Define Calc Zone 12-31
- Define Input Cell 12-32
- deleting
 - files and folders
 - using Current Folder browser 7-42
 - using functions 7-44
 - variables 6-9
- delimiter
 - matching in Editor 2-139
 - preferences for matching 2-139
- demos
 - using 4-25
- Dependency Report 10-15
- description for file
 - viewing in Current Folder browser 7-20
- desktop
 - active window 2-6
 - color preferences 2-150
 - configuration 2-5
 - description 2-2
 - docking 2-14
 - focus 2-6
 - font preferences for 2-141
 - grouping tools 2-14
 - layout
 - saving 2-37
 - maximizing tools 2-17
 - minimizing tools 2-17
 - predefined layouts 2-38
 - saving layout 2-37
 - tools
 - closing 2-8
 - opening 2-4
 - windows
 - closing 2-8
 - docking 2-9
 - grouping together 2-14
 - moving 2-10
 - opening 2-4
 - sizing 2-9
 - sizing using keyboard 2-9 2-13

- undocking 2-13 2-35
- development environment for MATLAB 2-2
- diagnostics
 - startup
 - Macintosh 1-6
- diary 3-51
- difference reporting for files 7-50
- directories
 - comparing 7-50 7-60
- directory. *See* folder
- disabling
 - breakpoints 9-159
- displaying
 - output 3-47
- displaying source control properties of a file 13-20
- dividers for code cells. *See* cell breaks
- do not show again
 - preferences 2-132
- docking tools in desktop 2-14
- docking windows in desktop 2-9
- document bar
 - document name 2-23
 - width 2-23
- document titles
 - in published MATLAB code 11-19
- documentation
 - accessibility 2-160
 - all products 4-39
 - most current version 4-39
 - printed 4-35
 - Web site 4-39
- documents
 - arranging in Editor 9-11
- dots (...) 3-20
- dragging in the desktop 2-114
- dynamic hyperlinks
 - inserting in published code 11-49
 - inserting to run MATLAB code 11-49

E

- echo execution 3-47
- edit
 - creating new files in the Editor 9-8
- editing
 - files
 - outside of MATLAB 9-4
 - in Command Window 3-18
 - MATLAB files 9-1
- editor
 - built-in 9-4
 - external to MATLAB 9-15
 - setting as default 9-15
- Editor 9-1
 - arranging documents 9-11
 - changing casing in 9-39
 - closing 9-11
 - closing files 9-86
 - description 9-6
 - example 9-142
 - go to
 - bookmark 9-71
 - function 9-71
 - line number 9-71
 - highlighting current line in 9-55
 - horizontal lines 9-180
 - indenting 9-6
 - modifying values 9-194
 - navigating 9-71
 - navigating back and forward 9-75
 - opening files in 9-9
 - preferences 9-13
 - publishing files 11-64
 - redoing an activity 9-40
 - rule displayed 9-56
 - running files 9-87
 - running with unsaved changes 9-165
 - status bar
 - function 9-57
 - undoing an activity 9-40

- using Command Window features in 9-38
 - Editor/Debugger
 - files in File menu 9-15
 - publishing images preferences 11-74
 - publishing preferences 11-74
 - EDU>> prompt in Command Window 3-3
 - ellipses (...) in statements 3-20
 - Embed Figures in MATLAB Notebook 12-25
 - embedding graphics
 - in MATLAB Notebook 12-25
 - encoding
 - preference when saving 2-131
 - end of file 9-16
 - ending MATLAB 1-23
 - environment settings at startup 1-15
 - environment variables 3-9
 - error breakpoints
 - stop for errors 9-170
 - error logs 1-24
 - error message identifiers 9-172
 - error messages
 - in Command Window 3-7
 - error style
 - definition 12-24
 - errors
 - color indicators 2-152
 - finding in files 9-104
 - run-time 9-104
 - source control 13-24
 - syntax 9-104
 - errors and warnings
 - analyzing code for 9-107
 - Evaluate Calc Zone 12-32
 - Evaluate Cell 12-33
 - Evaluate Loop 12-34
 - Evaluate Loop dialog box 12-21
 - Evaluate M-Book 12-34
 - evaluating
 - MATLAB Notebooks, ensuring data consistency 12-10
 - selection in Command History window 3-69
 - selection in Command Window 3-11
 - evaluating sections of a file 9-195
 - exact phrase
 - Help browser search 4-20
 - exe 3-8
 - executables
 - running from MATLAB 3-8
 - executing
 - group of statements 2-57
 - execution
 - displaying functions during 3-47
 - stopping 3-8
 - exiting MATLAB 1-23
 - expanding
 - code in files 9-34 9-57
- F**
- f* button 9-71
 - F Inc Search field 9-82
 - fatal error 1-24
 - favorites in Help browser 4-12
 - FIG files
 - opening in GUIDE 7-45 to 7-46
 - Fig-files
 - compatibility 2-131
 - save options 2-131
 - file
 - editing
 - options 9-4
 - file management system
 - See* source control system interface 13-1
 - files
 - appearance of 9-53
 - backing up 9-83 9-85
 - cleanup before publishing 11-54
 - closing 9-86
 - code cells and 9-175
 - colors in 9-53

- comparing 7-50
 - creating 9-4
 - from Command History window 3-70
 - creating from Command History 9-2
 - creating from Command Window 9-2
 - creating new 9-7
 - debugging 9-104
 - determining cyclomatic complexity of 9-105
 - determining McCabe complexity of 9-105
 - editing files 9-6
 - finding by name 7-30
 - formatting for publishing 11-10
 - formatting MATLAB comments in 9-45
 - log 1-16
 - managing 7-2
 - marking up code for publishing 11-60
 - marking up for publishing
 - section titles 11-22
 - table of contents 11-21
 - naming
 - avoiding conflicts 7-70
 - opening 9-9
 - opening as text files 7-45 to 7-46
 - opening outside MATLAB 7-45 to 7-46
 - operations in MATLAB 7-2
 - pausing 9-106
 - performance of 10-27
 - printing 9-85
 - profiling 10-27
 - publishing 11-64
 - before and after formatting 11-4
 - bold text 11-43
 - graphics 11-30
 - HTML markup tags 11-34
 - hyperlinks 11-46
 - inline LaTeX math symbols 11-39
 - italic text 11-43
 - LaTeX markup 11-36
 - LaTeX math symbols as blocks 11-40
 - lists 11-27
 - monospaced text 11-43
 - preformatted text 11-25
 - trademark symbols 11-45
 - publishing process 11-3
 - recommendations on saving 9-84
 - run configurations for 9-88
 - running
 - from Command Window 3-7
 - running sections of 9-175
 - saving 9-83
 - saving automatically in Editor 9-85
 - search path 7-66
 - searching contents of 7-30
 - snapshot of output in published MATLAB code 11-42
 - summary of markup for publishing 11-57
 - syntax highlighting in 9-53
- Files
- opening file or variable from 9-76
- filter
- Current Folder browser 7-27
- Find Files dialog box 7-30
- finding
- files and folders
 - by name 7-27
 - files by name and content 7-30
 - text in Command History window 3-75
 - text in Command Window 3-52
- finish.m file running when quitting 1-24
- firewall
- settings to work through 2-104
- fix me reports 10-4
- flags
- for startup 1-14
- focus 2-6
- folder
- operations in MATLAB 7-2
- folders
- comparing 7-60
 - creating 7-36

- MATLAB
 - caching 9-84
- font
 - adding new family for MATLAB 2-150
 - antialiasing in desktop 2-149
 - Help browser 4-30
 - preferences in MATLAB 2-141
 - size, additional values 2-142
 - smoothing in desktop 2-149
- format 3-48
 - controlling numeric format in MATLAB Notebook 12-25
 - in MATLAB Notebook 12-25
 - preferences 3-62
- FTP
 - transferring files via link 3-12
- full path name 7-8
 - copying 7-9
- function hints
 - disabling 3-37
- function name
 - automatic completion
 - Command Window 3-24
 - Editor 9-46
- function workspace 6-9
- functions
 - color indicators 2-152
 - determining usage of 9-107
 - displaying during execution 3-47
 - executing a group of 2-57
 - long (on multiple lines) 3-20
 - multiple in one line 3-20
 - naming
 - avoiding conflicts 7-70

G

- get latest version of file on Windows
 - platforms 13-14
- getting files 13-33

- graphical debugger 9-1
- graphics
 - controlling output in MATLAB Notebook 12-26
 - embedding in MATLAB Notebook 12-25
 - in MATLAB Notebooks 12-25
 - in published MATLAB code 11-30
 - within cell 11-27
- gray background color in desktop 2-152
- gray breakpoint icons 9-147
- gray horizontal lines in Editor 9-180
- green indicator in Editor 9-107
- Group Cells 12-34
- grouping
 - tools in desktop 2-14

H

- HDF
 - preference when saving 2-131
- headings
 - within code cell 11-19
- help
 - creating for program files 5-9
 - for selected function 3-38
- Help browser
 - color preferences 4-32
 - font preferences 4-30
 - printing from 4-35
 - viewing page location 4-12
- Help Report 10-8
- hidden files
 - viewing 7-16
- Hide Cell Markers 12-35
- history
 - automatic log file 1-16
 - source control on Windows platforms 13-16
- history file 3-67
- history of statements 3-66
- history.m file 3-67

- home 3-50
- horizontal lines in Editor 9-180
- hot keys 2-66
 - desktop 2-66
 - Variable Editor 6-34
- HTML
 - editing files in Editor 9-8
- HTML markup tags
 - in published MATLAB code 11-34
- HTML viewer in MATLAB 2-101
- hyperlinks
 - Command Window 3-12
 - in published MATLAB code 11-46
 - inserting in published code 11-49
 - running functions by 3-13
- I**
- image files
 - previewing in Current Folder Browser 7-21
- images
 - resizing in published MATLAB code 11-86
- import
 - files for use with MATLAB 7-72
- in files 9-10
- include
 - files with MATLAB 7-72
- incremental searching
 - in Editor 9-82
- indented text
 - within cell 11-27
- indenting
 - functions and nested functions 9-55
 - in Command Window 3-23
 - in Editor 9-54
 - preference in Editor
 - C/C++ 9-30
 - HTML 9-34
 - Java 9-32
 - preference in Editor/Debugger 9-21
- info.xml
 - Start button 2-96
- info.xml validation errors 5-61
- initiation (init) file for MATLAB 1-15
- inline LaTeX math symbols
 - in published MATLAB code 11-39
- inline links
 - within cell 11-46
- input
 - to MATLAB in Command Window 3-2
- input cells
 - controlling evaluation 12-21
 - controlling graphic output 12-26
 - converting autoinit cell to 12-32
 - converting text to 12-32
 - converting to autoinit cell 12-31
 - converting to cell groups 12-37
 - converting to text 12-15
 - defining in MATLAB Notebooks 12-12
 - evaluating 12-17
 - evaluating cell groups 12-18
 - evaluating in loop 12-21
 - maintaining consistency 12-10
 - timing out during evaluation 12-33
 - use of Word Normal style 12-16
- Input style
 - definition of 12-24
- Insert key
 - Editor 9-39
- insert mode
 - Editor 9-39
- Internet
 - proxy server settings 2-104
- interrupting a running program 3-8
- invalid breakpoints 9-147
- italic text
 - in published MATLAB code 11-43
 - within cell 11-43
- iterative programming 9-175

J

Java
 editing files in Editor 9-8
 Java Heap
 preferences 2-136
 Java VM
 starting without 1-17
 JAWS 2-161

K

K>>
 prompt in Command Window 3-3
 K>> prompt in Command Window
 debugging mode 9-149
 keyboard statement 9-106
 keyboard 9-106
 keyboard shortcuts
 Command Window 2-66
 Variable Editor 6-34
 keywords
 color indicators 2-152
 matching in Editor 2-139

L

LaTeX markup
 in MATLAB code 11-36
 LaTeX math symbols
 in published MATLAB code 11-40
 layout for desktop
 saving 2-37
 license information 4-37
 license management 2-121
 licenses 2-119
 line
 horizontal
 in Editor 9-180
 vertical
 in Editor 9-56

line breaks
 adding for long statements 3-20
 line continuation 3-20
 line numbers 9-55
 going to 9-71
 line termination 9-16
 line wrapping 3-62
 links
 Command Window 3-12
 in published MATLAB code 11-46
 lists
 in published MATLAB code 11-27
 within cell 11-27
 load 6-7
 locking files on checkout 13-33
 log
 automatic 1-16
 file 1-16
 session 3-51
 statements 3-66
 logfile startup option 1-16
 login
 remote on Macintosh 1-6
 long lines 3-20
 looping
 to evaluate input cells 12-21
 lowercase usage in MATLAB 3-17

M

M-Lint 9-107
 M-Lint Code Check Report. *See* Code Analyzer Report
 M-lint messages
 analyzing code for 9-107
 M-Lint messages
 suppressing 9-117
 M-Lint preferences. *See* Code Analyzer preferences
 Macintosh

- startup
 - remote login 1-6
 - MAT-files
 - comparing 7-57
 - compatibility 2-131
 - compression options 2-131
 - creating 6-5
 - defined 6-5
 - loading 6-7
 - preferences 2-131
 - updating using Current Folder Browser 7-37
 - viewing variables without loading 7-21
 - matched delimiters
 - preferences 2-139
 - matching parentheses
 - in Editor 2-139
 - MATLAB
 - commands, executing in a Word
 - document 12-17
 - quitting 1-23
 - confirmation 1-23
 - search path 7-72
 - MATLAB code file comments
 - purpose of 9-40
 - MATLAB code files
 - file association (Windows) 1-2
 - running
 - at startup 1-17
 - MATLAB files
 - editing 9-1
 - matlab folder 1-9
 - MATLAB functions
 - running by hyperlink 3-13
 - MATLAB installations
 - search path with 7-80
 - MATLAB Notebooks
 - creating 12-2
 - data consistency 12-10
 - data integrity 12-10
 - entering text and commands 12-9
 - evaluating all input cells 12-20
 - modifying style template 12-23
 - opening 12-6
 - printing 12-23
 - sizing graphic output 12-27
 - styles 12-23
 - matlab:
 - running functions with 3-13
 - matlab.mat 6-7
 - matlabrc.m, startup file 1-15
 - matrices
 - editing 6-24
 - maximizing
 - tools in desktop 2-17
 - measuring performance of your code 10-27
 - membership Web page 2-119
 - message identifiers 9-172
 - messages
 - suppressing 9-117
 - suppressing indicators 9-117
- Microsoft Word
- converting document to MATLAB Notebook 12-7
- minimize
- Windows startup option 1-16
- minimizing
- tools in desktop 2-17
- model files
- description in Current Folder browser 7-20
- monospaced text
- in published MATLAB code 11-43
 - within cell 11-43
- more 3-47
- mouse, right-clicking 2-108
- moving
- files and folders 7-45
- multidimensional arrays
- editing 6-27
- multiple item selection 2-113
- multiple lines for statements 3-20

- multiprocessing 3-8
- multithreading
 - turning off 1-17

N

- name clashes 7-70
- naming
 - functions and variables
 - avoiding conflicts 7-70
- navigating
 - files 9-71
- nested
 - code cells in files 9-185
- nested comments 9-43
- nested functions
 - indenting 9-55
- newsgroup for MATLAB 4-40
- newsletters 4-40
- nojvm startup option 1-17
- Normal style (Microsoft Word)
 - default style in MATLAB Notebook 12-23
 - defaults 12-24
 - used in undefined input cells 12-16
- notebook
 - function 12-2
 - overview 12-2
 - platforms supported 12-1
- Notebook
 - configuring 12-28
 - debugging 12-11
 - options 12-35
- Notebook menu
 - Word menu bar 12-2
- numbering lines 9-55
- numeric format
 - controlling in MATLAB Notebook 12-25
 - output 3-48
 - preferences 3-62

O

- objects
 - editing 6-27
- openvar
 - using 6-26
- operating system commands 3-8
- operators
 - searching for 4-23
- optimizing code performance 10-27
- options
 - shutdown 1-24
 - startup 1-14
- orange underline in file 9-112
- output
 - display
 - format 3-48
 - hidden 3-47
 - hiding 3-47
 - in Command Window 3-2
 - paging 3-47
 - spaces per tab 3-64
 - spacing of 3-62
 - suppressing 3-47
- output cells
 - converting to text 12-22
 - purging 12-22
- Output style
 - definition 12-24
- overwrite mode
 - Editor 9-39

P

- paging in the Command Window 3-47
- parentheses
 - matching 2-139
- parentheses matching
 - preferences 2-139
- partial
 - path name 7-10

- partial word
 - Help browser search 4-21
- passcodes 2-119
- path. *See* search path
- PATH environment variable 3-9
- path name 7-8
 - absolute 7-8
 - length 7-9
 - partial 7-10
 - relative 7-8
 - See also* absolute path name; full path name; relative path name
- pathdef.m
 - location 7-73
- pausing execution of file 9-144
- PDF
 - reader, preference for Help browser 4-33
- PDF documentation 4-35
- performance
 - improving for you code 10-27
- periods (...) 3-20
- Perl variables
 - passing
 - at startup 1-17
- Plot Selector tool
 - using the 6-10
- plotting
 - from the Workspace browser 6-10
- pop-up menus 2-108
- precision
 - output display 3-48
- preferences
 - code folding 9-62
 - Current Folder browser 7-14
 - Editor 9-13
 - MATLAB, general 2-129
 - publishing 11-74
 - publishing images 11-74
- preformatted text
 - in published MATLAB code 11-25
- printed documentation 4-35
- printing
 - Command History window contents 3-76
 - Command Window contents 3-50
- printing a MATLAB Notebook
 - cell markers 12-23
 - color 12-23
 - defaults 12-23
- problems, reporting to The MathWorks 4-38
- product filter in Help browser
 - preference 4-28
- profile 10-46
 - example 10-47
- profiling 10-27
- program control blocks
 - code folding and 9-34
- program elements
 - going to 7-23
- program files
 - help
 - viewing in Current Folder browser 7-20
 - naming
 - avoiding conflicts 7-70
 - viewing help for 7-21
- programs
 - running from MATLAB 3-8
 - stopping while running 3-8
- prompt
 - in Command Window 3-2
 - when debugging 9-149
- properties
 - source control on Windows platforms 13-20
 - tab completion
 - Command Window 3-32
 - Editor 9-51
- proxy server settings 2-104
- publish configuration
 - creating multiple 11-93
 - running 11-92
- publish configurations

- creating 11-66
- finding 9-96
- for files in Editor 11-65
- porting 11-104
- publish settings 11-70
- removing 9-98
- publish settings
 - in publish configurations 11-70
 - template 11-90
- publish_configurations.m file 11-104
- published MATLAB code
 - resizing images in 11-86
- publishing
 - MATLAB code and results 11-2
 - using code cells and 11-2
- publishing images preferences 11-74
- publishing preferences 11-74
- Purge Output Cells 12-35
- purging output cells 12-22

Q

- quitting
 - saving workspace 1-24
- quitting MATLAB 1-23
 - confirmation 1-23

R

- R Inc Search field 9-82
- rapid code iteration 9-175
 - scenarios 9-176
- rapid development 9-175
- recall previous lines 3-21
- recovering deleted files 7-42
- recycle 7-42
- red breakpoint icons 9-147
- red underline in file 9-112
- redo
 - in desktop 2-114

- in Editor 9-40
- refresh
 - Current Folder browser 7-15
- registered trademarks
 - within cell 11-45
- relative path 7-8
- release
 - latest 2-123
- remote login
 - Macintosh 1-6
- removing files from source control system 13-15
- renaming
 - files and folders
 - using Current Folder browser 7-42
 - using functions 7-42
- Report
 - Code Analyzer 10-22
 - folder 10-2
- reports
 - accessing 10-2
 - Contents 10-11
 - Dependency 10-15
 - Help 10-8
 - To do 10-4
 - TODO/FIXME 10-4
 - using 10-2
- Reports
 - Coverage 10-20
 - Fix me 10-4
- requirements
 - MATLAB 1-1
- resizing windows in the desktop 2-9
- restoring
 - tools in desktop 2-17
- results in MATLAB, displaying 6-26
- revision control
 - See* source control system interface 13-1
- right-hand text limit 9-56
- rule
 - in Editor 9-56

- rules (lines) in Editor 9-180
- run configurations
 - creating 9-88
 - creating multiple 9-93
 - exporting 9-96
 - finding 9-96
 - for files in Editor 9-88
 - importing 9-96
 - porting 9-96
 - removing 9-98
 - using 9-88
- run_configurations.m file 9-96
- run-time errors 9-104

S

- save
 - function 6-7
- saving
 - MAT-files
 - preferences 2-131
 - workspace upon quitting 1-24
- screen reader 2-161
- script for startup 1-15
- scroll buffer for Command Window 3-64
- scrolling in Command Window 3-47
- search path
 - adding folders to 7-75
 - behavior when changing folders 7-83
 - changing 7-75
 - default 7-66
 - description 7-66
 - problems and recovering 7-81
 - saving 7-79
 - using 7-72
 - using with different MATLAB installations 7-80
 - viewing 7-74
- searching
 - for files by name and content 7-30
- Help browser
 - Boolean 4-22
 - exact phrase (" ") 4-20
 - wildcard (*) or partial word 4-21
- in Current Folder browser
 - by name 7-27
 - typeahead 7-27
- special characters 4-23
- text
 - Command History window 3-75
 - Command Window 3-52
 - incrementally 9-82
- section breaks
 - in calc zones 12-31
- section titles
 - in published MATLAB code 11-22
- segmentation violation 1-24
- segv 1-24
- selecting multiple items 2-113
- semicolon (;)
 - after functions 3-47
 - between functions 3-20
- separator in functions 3-20
- session
 - automatic log file 1-16
- session log
 - Command History 3-66
 - diary 3-51
- setting breakpoints 9-144
- shadowed functions 7-70
- shell escape 3-8
- shortcut
 - for MATLAB in Windows 1-1
 - keys in MATLAB 2-66
- shortcut keys
 - Variable Editor 6-34
- shortcuts
 - categories 2-62
 - creating
 - from Command History window 3-70

- defined 2-57
 - deleting 2-62
 - displaying hidden labels on the toolbar 2-64
 - editing 2-62
 - file 2-59
 - labels, hiding 2-64
 - moving 2-62
 - organizing 2-62
 - toolbar 2-60
- Shortcuts
- Deleting from toolbar 2-64
- shortcuts.xml 2-59
- Show Cell Markers 12-35
- show file history on Windows platforms 13-16
- shutdown
- MATLAB 1-23
 - options 1-24
- Simulink model
- opening from a file 9-76
- Simulink models
- viewing complete descriptions of 7-21
- singleCompThread startup option 1-17
- sizing windows in the desktop 2-9
- smart recall 3-21
- source control on UNIX platforms
- getting files 13-33
 - locking files 13-33
- source control system interface 13-1
- UNIX platforms 13-26
 - preferences 13-27
 - selecting system 13-27
 - supported systems 13-26
 - Windows platforms
 - adding files 13-10
 - preferences 13-5
 - selecting system 13-5
 - supported systems 13-2
- source control system interface on UNIX platforms
- checking in files 13-30
 - checking out files 13-33
 - configuring ClearCase source control system 13-29
 - undoing file check-out 13-36
- source control system interface on Windows platforms
- checking out files 13-12
 - comparing working copy to source control version 13-18
 - displaying file properties 13-20
 - get latest version of file 13-14
 - removing files 13-15
 - showing file history 13-16
 - starting source control system 13-21
 - troubleshooting 13-24
 - undoing file check-out 13-13
- spaces in MATLAB commands 3-17
- spacing
- output in Command Window 3-62
 - tabs in Command Window 3-64
- special characters
- searching for 4-23
- splash screen
- startup option 1-17
- split screen display
- Editor 9-67
- stack
- in Editor 9-149
 - viewing 6-9
- Start button 2-94
- customizing 2-96
- starting MATLAB
- DOS 1-1
 - Linux 1-5
 - Windows 1-1
- startup
- diagnostics
 - Macintosh 1-6
 - files for MATLAB 1-15
 - files open 9-16

- folder for MATLAB 1-8
- Macintosh, remote login 1-6
- options for MATLAB 1-14
- script 1-15
- startup.m
 - location 1-16
 - startup file 1-15
- statement
 - definition 3-6
- statements
 - defined 3-5
 - executing a group of 2-57
 - long (on multiple lines) 3-20
- stepping through files 9-150
- stopping execution 3-8
- stops
 - in files 9-144
- stops (...) 3-20
- strings
 - across multiple lines 3-20
 - color indicators 2-152
 - saving as Unicode 2-131
- structures
 - editing 6-27
 - tab completion 3-31 9-50
- style preferences for text 2-141
- styles in MATLAB Notebook
 - modifying 12-23
- subfunctions
 - displaying in Editor status bar 9-57
 - going to in file 9-71
- support
 - technical 4-38
- suppressing output 3-47
- switches
 - for startup 1-14
- symbols
 - searching for 4-23
- syntax
 - color indicators 2-152

- color preferences in MATLAB 2-150
- coloring and indenting 3-23
- errors 9-104
- highlighting 9-53
- system browser
 - UNIX 2-106
- system environment variables 3-9
- system path for UNIX 3-9
- system requirements
 - MATLAB 1-1
- system Web browser 2-101

T

- tab
 - indenting in Editor 9-54
 - preference for indenting in Editor
 - C/C++ 9-30
 - HTML 9-34
 - Java 9-32
 - preference for indenting in Editor/Debugger 9-21
 - spacing in Command Window 3-64
- tab completion
 - Command Window 3-24
 - Editor 9-46
- tabbing desktop windows together 2-14
- Technical Support
 - contacting 4-38
 - Web page 2-119
- templates
 - for publish settings 11-90
 - MATLAB Notebook 12-23
- temporary folder
 - for deleted files 7-42
- terminating a running program 3-8
- text
 - converting to input cells 12-32
 - finding and replacing 9-80
 - finding in current file 9-78

- preferences in MATLAB 2-141
- styles in MATLAB Notebook 12-23
- text editor, setting as default 9-15
- text editors for files 9-4
- text files
 - comparing 7-52
 - editing in Editor 9-8
- threads
 - turning off multithreading 1-17
- time
 - measured for your code 10-27
- time-out message
 - while evaluating multiple input cells in a MATLAB Notebook 12-33
- titles
 - in published MATLAB code 11-19 11-21 to 11-22
- TLC
 - editing files in Editor 9-8
- tmp/MATLAB_Files folder 7-44
- to do reports 10-4
- TODO/FIXME Report 10-4
- Toggle Graph Output for Cell 12-36
- token matching
 - preferences 2-139
- tool tips 2-110
- toolbars
 - customizing 2-156
 - desktop 2-110
 - Editor cell mode 9-177
 - shortcuts 2-60
- toolbox path cache
 - preferences 1-20
- toolboxes
 - custom, adding to Start button 2-96
- tools in desktop
 - description 2-2
- Tooltips
 - for data 9-153
 - viewing folded code in 9-61

- trademark symbols
 - in published MATLAB code 11-45
 - within cells 11-45
- trial versions 2-119
- troubleshooting
 - cell highlighting 9-182
 - source control problems 13-24
- type ahead feature 3-21

U

- UNC (Universal Naming Convention) path 10-3
- uncomment 9-41
- Undefine Cells 12-36
- undo
 - in desktop 2-114
 - in Editor 9-40
- undocking windows from desktop 2-13 2-35
- undoing file check-out
 - on UNIX platforms 13-36
 - on Windows platforms 13-13
- Ungroup Cells 12-37
- Unicode
 - preference when saving 2-131
- UNIX
 - system path 3-9
- updates
 - to newer versions 2-123
- updates to products 2-119
- uppercase usage in MATLAB 3-17
- utilities
 - running from MATLAB 3-8

V

- validating
 - MATLAB code 10-22
- values
 - examining 9-152
- Variable Editor 6-24

- cut, copy, paste, clear 6-36
 - decimal separator 6-47
 - keyboard shortcuts 6-34
 - preferences 6-46
 - size limitations 6-26
 - undo and redo 6-40
 - variables
 - deleting or clearing 6-9
 - determining usage of 9-107
 - displaying values of 6-26
 - editing values 6-24
 - finding in current file 9-78
 - naming
 - avoiding conflicts 7-70
 - opening from a file 9-76
 - saving 6-5
 - viewing 7-21
 - viewing during execution 9-152
 - viewing values in Editor 9-153
 - workspace 6-2
 - Verilog
 - editing files in Editor 9-8
 - version 2-123
 - information for MathWorks products 4-37
 - latest available 2-123
 - version control
 - See* source control system interface 13-1
 - vertical line
 - in Editor 9-56
 - VHDL
 - editing files in Editor 9-8
 - viewing desktop tools 2-5
 - Visible figure property
 - embedding graphics in MATLAB Notebook 12-26
- W**
- warning breakpoints 9-170
 - warning message identifiers 9-172
 - Web
 - accessing from MATLAB 2-119
 - preferences 2-104
 - proxy server settings 2-104
 - site for MathWorks 2-119
 - Web Browser
 - font 2-107
 - in MATLAB 2-101
 - Web site
 - documentation 4-39
 - who 6-4
 - whos 6-4
 - wildcard (*)
 - Help browser search 4-21
 - window
 - active 2-6
 - focus 2-6
 - windows in desktop
 - about 2-2
 - arrangement 2-5
 - closing 2-8
 - docking 2-9
 - moving 2-10
 - opening 2-5
 - sizing 2-9
 - undocking 2-13 2-35
 - Word documents
 - converting to MATLAB Notebook 12-7
 - workspace
 - base 6-9
 - clearing 6-9
 - defined 6-2
 - functions 6-9
 - initializing in MATLAB Notebook 12-14
 - loading 6-7
 - MATLAB Notebook contamination 12-10
 - opening 6-7
 - protecting integrity 12-10
 - saving 6-5
 - tool 6-2

- viewing 6-4
- viewing during execution 9-152
- Workspace browser
 - description 6-2
 - plotting variables from 6-10
 - preferences 6-9
- wrapping
 - lines in Command Window 3-62
 - long statements 3-20

X

XML

- editing files in Editor 9-8
- XML: file validation 5-61

Y

- yellow highlighting in file 9-112
 - current cell 9-180
 - data tip 9-153

Z

- zip files
 - adding files to 7-40
 - creating 7-38 to 7-39
 - extracting files from 7-39
 - viewing contents of 7-38