

MATLAB 7.0 Release Notes

The MATLAB® 7.0 Release Notes describe the changes introduced in the latest version of MATLAB for Release 14. The following topics are discussed in these Release Notes:

Major MATLAB Changes

MATLAB 7.0 introduces many new features and improvements over previous releases. The most important changes to be aware of are

- “Case-Sensitivity in Function and Directory Names” on page 1-43
- “Differences Between Built-Ins and M-Functions Removed” on page 1-43
- “New Features for Nondouble Data Types” on page 1-58
- “MATLAB Stores Character Data As Unicode” on page 1-44

Release Notes

New Features (p. 1-1)

New features and enhancements introduced in this release

Platform Limitations (p. 2-1)

Platform-related limitations to the MATLAB functionality described in these Release Notes and in the MATLAB documentation

Upgrading from an Earlier Release (p. 3-1)

Changes in MATLAB other than new features that have been made since MATLAB 6.5.1

Major Bug Fixes (p. 4-1)

Particularly important bug fixes made since MATLAB 6.5.1.

If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

Known Software and
Documentation Problems
(p. 5-1)

Significant problems that still exist in the MATLAB code or
documentation.

If you are viewing these Release Notes in PDF form, please refer to
the HTML form of the Release Notes, using either the Help browser
or the MathWorks Web site and use the link provided.

Printable Release Notes

PDF version of these Release Notes

New Features

1

| | |
|--|------|
| Desktop Tools and Development Environment Features .. | 1-2 |
| Mathematics Features | 1-23 |
| Programming Features | 1-42 |
| Graphics and 3-D Visualization Features | 1-66 |
| External Interfaces/API Features | 1-74 |
| Creating Graphical User Interfaces (GUIDE) Features .. | 1-85 |

Platform Limitations

2

| | |
|-------------------------------------|-----|
| Graphics Platform Limitations | 2-2 |
|-------------------------------------|-----|

Upgrading from an Earlier Release

3

| | |
|---|------|
| Desktop Tools and Development Environment Upgrade Issues | 3-2 |
| Mathematics Upgrade Issues | 3-5 |
| Programming Upgrade Issues | 3-12 |
| Graphics Upgrade Issues | 3-27 |

| | |
|---|-------------|
| External Interface/API Upgrade Issues | 3-28 |
| Creating Graphical User Interface (GUIDE) Upgrade Issues | 3-30 |

Major Bug Fixes

4 |

Known Software and Documentation Problems

5 |

MATLAB 6.5.1 Release Notes

6 |

| | |
|--|-------------|
| New Features | 6-2 |
| Major Bug Fixes | 6-11 |
| Upgrading from an Earlier Release | 6-22 |
| Known Software and Documentation Problems | 6-23 |

MATLAB 6.5 Release Notes

7 |

| | |
|------------------------------|-------------|
| New Features | 7-2 |
| Major Bug Fixes | 7-43 |

| | |
|--|-------------|
| Platform Limitations | 7-44 |
| Upgrading from an Earlier Release | 7-49 |
| Known Software and Documentation Problems | 7-86 |

MATLAB 6.1 Release Notes

8

| | |
|--|-------------|
| New Features | 8-2 |
| Major Bug Fixes | 8-18 |
| Upgrading from an Earlier Release | 8-23 |
| Known Software and Documentation Problems | 8-29 |

New Features

This section introduces the new features and enhancements added in MATLAB 7.0 since Version 6.5.1 (R13 SP1). If you are using the Help browser, view the New Features in Version 7 video demos that highlight the major new features. Documentation of new MATLAB features is organized into these categories:

- Desktop Tools and Development Environment Features (p. 1-2)
- Mathematics Features (p. 1-23)
- Programming Features (p. 1-42)
- Graphics and 3-D Visualization Features (p. 1-66)
- External Interfaces/API Features (p. 1-74)
- Creating Graphical User Interfaces (GUIDE) Features (p. 1-85)

If you are upgrading from a release earlier than R13 SP1, then you should also see “New Features” on page 6-2 in the MATLAB 6.5.1 Release Notes.

Desktop Tools and Development Environment Features

If you are using the Help browser, view the Desktop Tools and Development Environment new features video demo and the Editor new features video demo to see highlights of the new features.

Documentation of new features and enhancements in MATLAB 7.0 Desktop Tools and Development Environment is organized by these topics:

- “Startup and Shutdown” on page 1-2
- “Desktop” on page 1-2
- “Running Functions—Command Window and History” on page 1-6
- “Help for Using MATLAB” on page 1-8
- “Workspace, Search Path, and File Operations” on page 1-9
- “Editing and Debugging M-Files” on page 1-13
- “Tuning and Managing M-Files” on page 1-18
- “Publishing Results” on page 1-22

Startup and Shutdown

MATLAB is now using Java (JVM) 1.4.2.

Desktop

See the complete “Desktop” documentation online.

Demo of MATLAB Desktop


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

Arranging Documents

The MATLAB desktop now provides you with new options for arranging the following types of documents:

- M-files and other files in the Editor/Debugger
- Arrays in the Array Editor
- Figure windows
- HTML documents in the MATLAB Web browser

You can dock these types of documents in the desktop, undock them from the desktop so each is in its own separate window, or group undocked documents together in their tool. You can now position the documents using these features: tile, left/right split, top/bottom split, floating, or maximized. Use the **Window** menu or toolbar icons to position documents.

Docking Tools and Documents. There are now dock buttons  in the menu bars of undocked tools and documents. Click a dock button to move the tool into the desktop, or to move the document into its tool.

Document Bar. There is now a Document Bar in tools that support documents that you use to go to open documents. It appears when there is more than one maximized document open in a tool. You can hide or move the Document Bar by selecting **Desktop -> Document Bar** menu options.

Saving Layouts. You can save desktop layouts. Select **Desktop -> Save Layout** and provide a name. To restore a saved layout, select **Desktop -> Desktop Layout -> name**.

Launch Pad. The Launch Pad tool was removed. Use the **Start** button instead.

Adding Your Own Toolbox to Start Button. Add your own toolbox to the **Start** button. Select **Start -> Desktop Tools -> View Source Files**. Click **Help** in the resulting dialog box for details.

Finding Files and Content

Search for files and directories, as well as for content within files by selecting **Edit -> Find Files** from any desktop tool. For details, see “Finding Files and Content Within Files” in the online documentation.

MATLAB Shortcuts

You can create and run MATLAB shortcuts, where a shortcut is an easy way to run a group of MATLAB statements. A shortcut is like an M-file script, but unlike an M-file, a shortcut does not have to be on the MATLAB search path or in the current directory when you run it.

Create a shortcut by selecting **Start -> Shortcuts -> New Shortcut** and completing the dialog box. Run the shortcut from the **Start** button.

You can also create a shortcut by dragging selected statements to the shortcut toolbar. This adds the shortcut to the toolbar, from where you can then run it. For details, see “Shortcuts for MATLAB” in the online documentation.

MATLAB Web Browser

MATLAB now displays HTML documents it produces in a new desktop tool, the MATLAB Web browser. You can display HTML documents in this Web browser using the web function.

The web function now opens the MATLAB Web browser by default, instead of opening the MATLAB Help browser. Use the web function’s `-helpbrowser` option to display files in the Help browser.

Menus

- You can now access debugging features from the **Debug** menu of most desktop tools.
- There is no longer a desktop **View** menu, although some tools still have a **View** menu. The **Window** menu in the desktop has changed. Use the new **Desktop** menu to select a layout, and to open and close tools. Use the **Window** menu to access open tools and documents, as well as to position documents. The menus and the menu items in the desktop change, depending on the current tool selected.
- The **Web** menu was removed. Access the items it contained from **Help -> Web Resources**.

Keyboard Shortcuts

There is now a keyboard shortcut you can use to go to each tool and to each open document. For example, use **Ctrl+0** to go to the Command Window, and **Ctrl+Shift+0** to go to the most recently used Editor document. See the **Window** menu for the shortcuts to go to currently open tools and documents.

There have been some changes to the keyboard shortcuts you use with desktop tools. For example, **Ctrl+Tab** now moves you to the next open tool or group of tools tabbed together. In previous releases, **Ctrl+Tab** moved you to the next open document or tool. In MATLAB 7, use **Ctrl+Page Down** to move to the next open document or tool in a tabbed group. For the complete list, see “Keyboard Shortcuts” in the online documentation.

Drag and Drop

You can drag selected text or files between desktop tools. For example, you can

- Select text in the Editor and drag it to the Command Window, which cuts and pastes it into the Command Window. You can use **Ctrl** while dragging to copy selected text instead of just moving it.
- Select a file in the Current Directory browser and drag it to the Editor, which opens the file in the Editor.

You can also drag selected text or files between desktop tools and external tools and applications. For example, you can

- Select a MAT-file from the Microsoft Windows Explorer and drag it to the Command Window, which loads the data into the MATLAB workspace.
- Select text from a page displayed in a Netscape browser and drag it to a file in the Editor, which pastes the text into the file in the Editor.

Arranging Columns in Tools

In desktop tools that contain columns, you can drag a column to a new position. For example, this includes the Current Directory browser, and the Help browser **Index** and **Search** panes. Click a column head to sort by that column. For some tools, you can click again to reverse the sort order.

When a column is too narrow to show all the information in it, position the cursor over a long item in that column, and a tooltip displays showing the complete content of the item.

Font and Color Preferences for Tools

Access font and color preferences for all desktop tools in the **Fonts** and **Colors** preference panels. Select **File -> Preferences -> Fonts** or **File -> Preferences -> Colors**. For more information, click the **Help** button in the preferences dialog box, or see Fonts, Colors, and Other Preferences in the online documentation.

Running Functions—Command Window and History

See the complete “Running Functions” documentation online.

Command Window Features

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

Additions. These are new features in the Command Window.

- Tab completion now has a graphical interface. For example, type `cos` and press the **Tab** key. A list of functions that begin with `cos` appears. Double-click the function you want and MATLAB completes the name in the Command Window. Alternatively, when the list of names appears, you can type the next unique letter in the name, and the first name in the list that matches it is selected. Continue typing unique letters to select the name you want, and press **Enter**. Press **Escape** to clear the list without selecting a name.
- There is a new preference that allows you to use arrow keys to navigate in the Command Window instead of recalling history.
- The incremental search interface has been updated. It now indicates the search direction. It is also case-sensitive when you enter uppercase letters in the search field.
- Use the new `commandwindow` function to open the Command Window when it is closed (for example, from an M-file), or to select the Command Window when it is already open.
- On Macintosh platforms, you can now use **Command+**. (**Command** key and period key) to stop execution of a running program.

Changes. These features operate differently in this release.

- When you include an ellipsis in a statement so that you can continue the statement on the next line, any text you type after the `...` on the same line is considered to be a MATLAB comment and now is syntax highlighted as a comment. In previous releases, the syntax highlighting did not indicate the text after the `...` as a comment.

- Evaluate selection (available from context menus for various tools) no longer appends the selection to the statement at the prompt, but instead runs the selection. Make a selection and press **Enter** or **Return** to append the selection to the statement at the prompt and execute it.
- The default colors for syntax highlighting have been modified. Unterminated strings are now maroon, while terminated strings are now purple. This is the opposite of previous versions. Maroon is considered to be more of an “alerting” color, resembling the default of red for errors, which is the reason for the change. If you prefer the colors used in previous versions, change them using preferences—see Syntax Highlighting Colors in the online documentation.

In addition, arguments in statements entered using command syntax rather than function syntax are highlighted as strings, emphasizing that variables in command syntax are passed as literal strings rather than as their values.

- Stopping execution using **Ctrl+C** (^C) has changed. Windows and UNIX platforms now respond similarly to **Ctrl+C**, and in general, stop execution without the need for pause or drawnow statements in your M-files. For M-files that run for a long time, or that call built-ins or MEX-files that take a long time, **Ctrl+C** does not always effectively stop execution. In that event, include a drawnow command in your M-file, for example, within a large loop. **Ctrl+C** might be less responsive if you started MATLAB with the `-nodesktop` option.

Command History Features

- If you are using the Help browser, watch the new Command history video demo for an overview of the major functionality.
- Entries in the Command History tool now appear with syntax highlighting.
- Entries in the Command History now appear in a tree view so you can minimize the length of the visible history. The top level nodes of the tree are the dates/times for each session, and beneath that is the history for that session. Click the - to the left of a date/time to hide the history entries for that session. Click the + to the left of a date/time entry to show history entries for that session.
- Use the new `commandhistory` function to open the Command History when it is closed, and to select it when it is open.

- The default for saving the history has changed. Now, by default, MATLAB saves the history file after five statements have been added to the history. You can modify the frequency using Command History preferences.

Help for Using MATLAB

See the complete “Help” documentation online.

General Help and Documentation

- If you are using the Help browser, watch the new Help and Documentation video demo for an overview of the major functionality.
- Documentation is automatically installed for all the products you install. Documentation is no longer accessible from CD-ROMs. To access the documentation for products not installed on your system, use The MathWorks Web site, <http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml>. Because of this change, the docroot function is no longer needed and will not be supported.

Help Browser

- The Index tab now has an alphabetical quick index, so you can choose a letter to see entries starting with that letter. You can still type any index term in the text box to go directly to that term. Index entries are now shown as links. Entries that are merely headings do not go to a specific page and do not appear as links.
- In the Search tab, you no longer select the type of search. Results are ordered so reference pages appear first, followed by headings that include the search terms. After performing a search, click the link at the bottom of the Search pane to look for the same term in the technical support database on The MathWorks Web site.
- As is true for all desktop tools, you can drag columns in the Index and Search panes to reorder them, or click a column head to sort by that column.
- Add pages in the Help browser to favorites (also known as bookmarking pages) by selecting **Favorites -> Add to Favorites**. The **Favorites Editor** dialog box opens. Accept the default entries or modify the **Label** and click **Save**. Access favorites from the **Favorites** menu or from the **Start** menu **Shortcuts** item.

- Click the binoculars icon on the Display pane toolbar to search within the page.
- The Help browser is now used only for MathWorks documentation installed with your products. You can no longer enter a URL in the **Title** field of the display pane. Instead run the web function to enter a URL in the **Location** field. Links from the documentation to Web pages display the Web pages in the MATLAB Web browser, not in the Help browser.

Help functions

- The new docsearch function allows you to execute a full text search of the Help browser documentation from the Command Window.
- The help function now allows you to get help for methods and classes. For details, see specific instructions in the release notes about using help and doc for each product, or type help help.


Workspace, Search Path, and File Operations

See the complete “Workspace, Search Path, and File Operations” documentation online.


- “MATLAB Workspace and Workspace Browser” on page 1-9
- “Array Editor” on page 1-10
- “Search Path” on page 1-11
- “File Operations” on page 1-11

MATLAB Workspace and Workspace Browser

- If you are using the Help browser, watch the new Workspace Browser video demo for an overview of the major functionality.
- The Workspace browser now includes a **Value** column where you can see the content of the variable, or a description of the content. Click the value in the **Value** column to edit the content.
- Click a variable name (in the **Name** column) to rename the variable. To create a copy of a variable, right-click and select **Duplicate** from the context menu.

- Click the plot icon  in the Workspace browser toolbar to plot the selected variable. Choose from other applicable plots by clicking the arrow next to the plot button. The function used to create the plot appears in the Command Window so you can use it again later.
- Click the print button in the Workspace browser toolbar to print a view of the current workspace.
- MAT-files are now compressed by default. For details on compressing MAT-files, see “Compressed Data Support in MAT-Files” on page 1-59.
- Use the new function `genvarname` to construct a valid MATLAB variable name from a given candidate, where the candidate can be a string or a cell array of strings. For details, type `help genvarname`.
- The new function `datatipinfo(x)` displays information about the variable, `x`.

Array Editor

- If you are using the Help browser, watch the new Array Editor video demo for an overview of the major functionality.
- You can now view and edit the content of cell arrays and structures in the Array Editor. For example, double-click a structure in the Workspace browser to open it in the Array Editor. In the Array Editor, double-click an element of the structure to open it as its own Array Editor document. You can then view and edit the contents.
- You can select contiguous elements in an array, and then click the plot button  on the Array Editor toolbar to plot only the selected elements. Click the arrow next to the plot button in the toolbar to select from other applicable plots.
- You can print an array from the Array Editor. Select **File -> Print** to create a print of the current variable.
- You can open arrays having up to 2^{19} (524288) elements, which is eight times more than the previous limit, 2^{16} (65536).
- You can save a variable to a MAT-file from the Array Editor. Select **File -> Save** and complete the resulting **Save** dialog box.

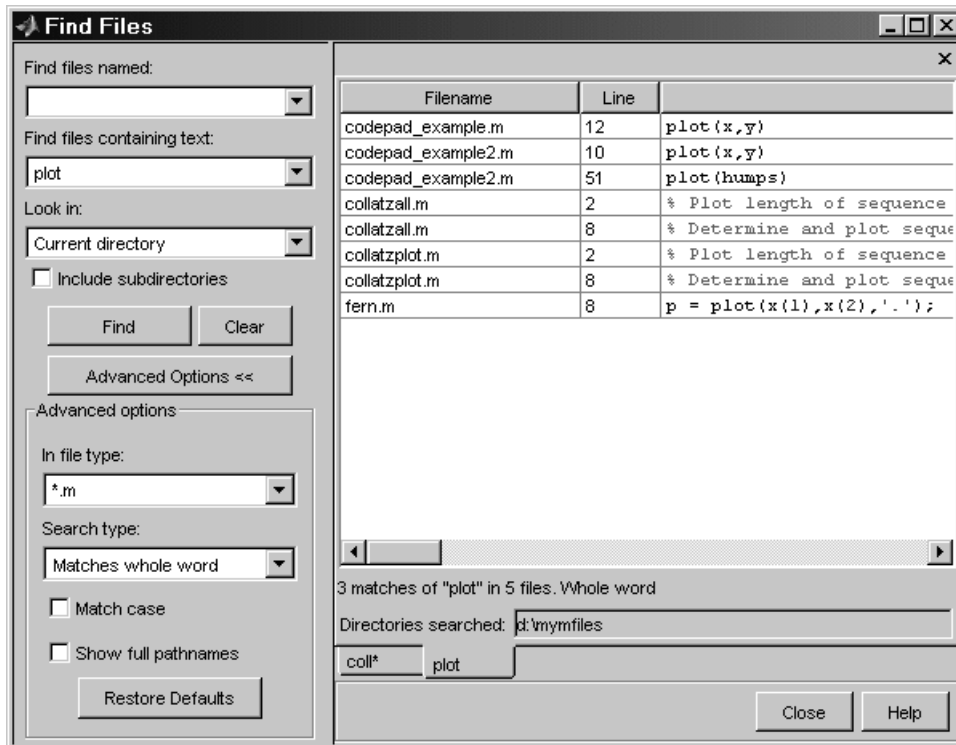
Search Path

- MATLAB now considers built-in files to be the same as other M-files on the search path. For more information, see “File Operations, Workspace, and Path” on page 3-3 in Upgrading from an Earlier Release.
- There is a new function, `savepath`, that saves the current search path to a file, `pathdef.m`, so that you can use the same search path in future sessions. Note that this function replaces `path2rc`.
- There is a new function, `restoredefaultpath`, that helps redefine the search path file, `pathdef.m`, to include only files installed with MathWorks products. Use this function to recover from problems with the path. If that fails, run

```
restoredefaultpath; matlabrc.
```
- The `genpath` function now includes empty directories in the generated path string.
- The `which` function now displays the pathname for built-in functions, as well as for overloaded functions when only the overloaded functions are available.

File Operations

Finding Files and Content Within Files. From any desktop tool, select **Edit -> Find Files**. Complete the resulting dialog box to find specified files or files containing specified text in the directories you choose. Double-click a file in the results listing to open it. For details, see “Finding Files and Content Within Files” in the online documentation.



Preventing Accidental File Deletion. Use the new recycle function or the **General** preference for the delete function to send files you remove using the delete function to the Recycle Bin on Windows, to the Trash Can on Macintosh, or to a /tmp/MATLAB_Files_timestamp directory on UNIX systems. You can then recover any accidentally deleted files from these locations.

Current Directory Browser Enhancements.

- If you are using the Help browser, watch the new Current Directory Browser video demo for an overview of the major functionality.
- You can access source control system features from the Current Directory browser. Right-click a file or directory, and from the context menu, select **Source Control** and then select the source control function you want to use.

- To open a file using an external application, select **Open Outside MATLAB** from the context menu. For example, if you select `myfile.doc`, **Open Outside MATLAB** opens `myfile.doc` in Microsoft Word, assuming you have the `.doc` file association configured to launch Word.
- Using the Current Directory browser, you can now copy and paste directories, including the entries contents.
- As is true for all desktop tools, you can drag columns in the Current Directory browser to reorder them, or click a column head to sort by that column. For an item that does not fit in its column, you can hover over it to see the full name of the item.
- The current directory field appears in the Current Directory browser only when the Current Directory browser is undocked from the MATLAB desktop. When the Current Directory browser is docked in the MATLAB desktop, use the current directory text field in the desktop toolbar.

Visual Directory and Directory Reports in the Current Directory Browser. There are new tools accessible from the Current Directory browser for tuning and managing M-files. For details, see “Visual Directory Tool in the Current Directory Browser” on page 1-18 and “Directory Reports in the Current Directory Browser” on page 1-19.

Editing and Debugging M-Files

If you are using the Help browser, view the Editor new features video demo to see highlights of the major new features.

See the complete “M-File Editing and Debugging” documentation online.

- “Opening, Arranging, and Closing Documents” on page 1-14
- “Visual Changes” on page 1-14
- “Entering Statements” on page 1-15
- “Finding and Replacing Text” on page 1-16
- “Printing M-Files” on page 1-16
- “Breakpoints and Debugging” on page 1-16
- “Rapid Code Iteration Using Cells” on page 1-17
- “Preferences for the Editor/Debugger” on page 1-18

Opening, Arranging, and Closing Documents

- You can drag a file onto Editor to open it. For example, drag a text file from Windows Explorer onto the Editor.
- There is now an Editor/Debugger preference you can set to automatically remove autosave files when you close the source file. Select **Preferences -> Editor/Debugger -> Autosave**, and under **Close options**, select the **Automatically delete autosave files** check box.
- To move from an Editor document to the Command Window, press **Ctrl+0**. To move back to the Editor document, press **Ctrl+Shift+0**.
- When you close the last open document in the Editor, the Editor remains open.
- When a file is open in the Editor and you open that same file outside of MATLAB and make changes to it, the Editor automatically updates the file to include the changes you made outside the Editor. This only applies if you did not make any changes to the file in the Editor. If you want to be prompted before the Editor updates the file, clear the Editor/Debugger preference for automatically reloading files.
- In the previous version, you used a preference to automatically open files when debugging. Now, instead of using a preference, you select **Open M-Files When Debugging** from the **Debug** menu in any desktop tool.
With this item selected, when you run an M-file containing breakpoints, the file opens in the Editor/Debugger when MATLAB encounters a breakpoint.

Visual Changes

- The Editor now supports syntax highlighting for other languages, specifically C/C++, Java, and HTML. Use Editor language preferences to change the colors for the syntax highlighting.
- In edit mode, datatips are now off by default. Select the preference to display them in edit mode. Datatips display until you move the cursor. Datatips are always on in debug mode.
- There is now a faint line at column 75, which serves as a useful reminder of where text would be cut off when printing the document. Remove the line or change the column at which the line appears using Editor/Debugger Display preferences.

- The feature **Text -> Balance Delimiters** has been removed.
- The default colors for syntax highlighting M-files have been modified. Unterminated strings are now maroon, while terminated strings are now purple. This is the opposite of previous versions. Maroon is considered to be more of an “alerting” color, resembling the default of red for errors, which is the reason for the change. If you prefer the colors used in previous versions, change them using preferences—see “Syntax Highlighting Colors” in the online documentation.

In addition, arguments in statements entered using command syntax rather than function syntax are highlighted as strings, emphasizing that variables in command syntax are passed as literal strings rather than as their values.

Entering Statements

- You can create a block comment in an M-file using any text editor, that is, you can comment out contiguous lines of code. Type `%{` on the line before the first line of the comment and `%}` following the last line of the comment. The lines in between are considered to be comments. Do not include any code on the lines with the block comment symbols. You can also nest block comments. See “Commenting Using Any Text Editor” for details.
- To change the case of selected text, select the text and then press:
 - **Alt+U, U** to change all text to upper case
 - Press **Alt+U, L** to change all text to lower case
 - Press **Alt+U, R** to change the case of each letter
- MATLAB now supports nested functions and the Editor provides preferences regarding how to indent them.
- When you press the **Insert** key, text entry is done in overwrite mode and the cursor assumes a block shape. Press the **Insert** key again to return to insert mode.

Finding and Replacing Text

- You can find directories, files, and content within multiple files. Select **Edit -> Find Files**. For details, see “Finding Files and Content Within Files” in the online documentation.
- The incremental search interface has been updated. It now indicates the search direction. It is also case-sensitive when you enter uppercase letters in the search field.

Printing M-Files

Page setup options differ slightly from previous versions.

Breakpoints and Debugging

- You can specify conditional breakpoints in an M-file. MATLAB only stops at the line with the breakpoint if the condition is met. Conditional breakpoints have a yellow breakpoint icon, which you can copy and paste to other lines.
- You can disable standard and conditional breakpoints. MATLAB ignores a disabled breakpoint until you enable it again. A disabled breakpoint icon has an X through it.
- Set error breakpoints for all files by selecting **Debug -> Stop If Errors/Warnings**, and then completing the resulting dialog box. You can specify a message identifier for an error or warning breakpoint so that MATLAB stops only if it encounters the specified error or warning message.
- Enhancements to debugging functions include `dbstop if caught error`, `dbclear if caught error`, and `dbclear if all error`. The `dbstop if all error` option has been grandfathered and will not be supported in future versions. To specify a message identifier, use `dbstop if error ID`, `dbstop if caught error ID`, `dbstop if warning ID`, and the corresponding `dbclear` options. The `dbstatus` function has been updated to reflect the changes to `dbstop` and `dbclear`.

- The `dbstop` function has been updated to support nested and anonymous functions. See the `dbstop` reference page for details.
You cannot use the Editor/Debugger GUI to set breakpoints in anonymous functions, but must use the `dbstop` function instead. Note that when you save a file in the Editor/Debugger that contains breakpoints in anonymous functions, those breakpoints are cleared. They are also cleared when you run an unsaved file from the Editor/Debugger GUI, because running first saves the file.
- The `dbstack` function has been updated to supported nested functions. See the `dbstack` function reference page and the “Editing and Debugging” upgrade issues for more information.
- The `dbstatus` function has been updated to support conditional breakpoints. See the `dbstatus` function reference page and the “Editing and Debugging” upgrade issues for more information.
- You can access useful tools for M-files from the Editor/Debugger. From the **Tools** menu, select **Check Code with M-Lint**, **Show Dependency Report**, or **Open Profiler**. For details about these tools, see “Tuning and Managing M-Files” on page 1-18.
- MATLAB now uses a new notation for reporting the path of functions, subfunctions, and nested functions. As an example, `A/B>C/D` means directory A, file B, (sub)function C within the file B, and nested function D within C.

Rapid Code Iteration Using Cells

If you are using the Help browser, watch the new Rapid Code Iteration Using Cells video demo for an overview of the major functionality.

In the Editor, cell features allow you to easily make changes to values in a section of an M-file to readily see the impact of the changes. First, you define cells in a file, then evaluate a cell or cells, iterate values in the cell, and then reevaluate the cell(s). Cells also allow you to publish M-file code and results to popular formats, such as HTML and Microsoft Word. For details, see “Rapid Code Iteration Using Cells” in the online documentation.

Preferences for the Editor/Debugger

- There is now a preference that allows you to add a new line to end of a file upon saving.
- The feature that instructs M-files to open automatically when debugging is no longer in preferences but is now accessible from the **Debug** menu in all desktop tools.

Tuning and Managing M-Files



If you are using the Help browser, watch the new Directory Reports video demo for an overview of the major functionality.

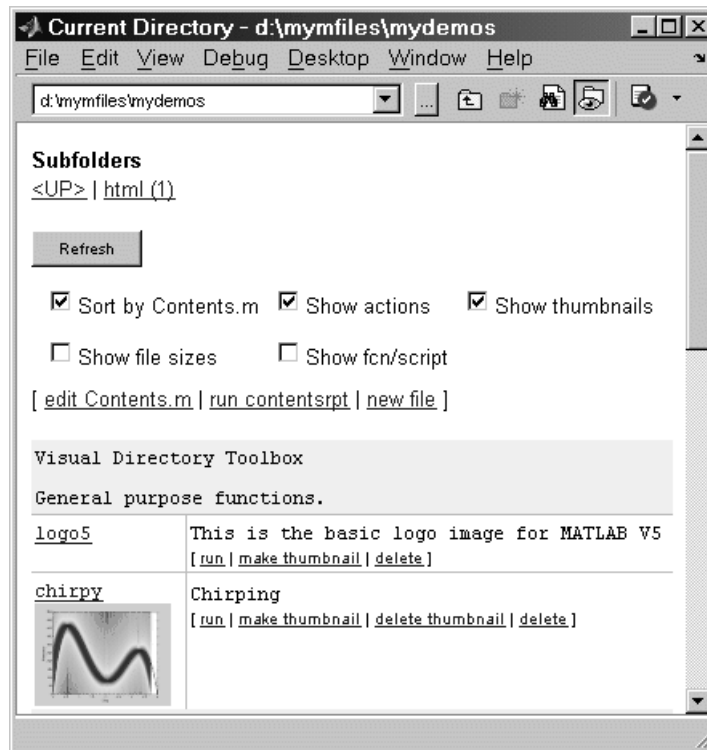
See the complete “Tuning and Managing M-Files” documentation online. Use these tools to fine tune and manage your M-files, and to prepare them for distribution to other users.

- “Visual Directory Tool in the Current Directory Browser” on page 1-18
- “Directory Reports in the Current Directory Browser” on page 1-19
- “Profiler for Measuring Performance” on page 1-21

Visual Directory Tool in the Current Directory Browser

The Visual Directory view of the Current Directory provides useful information about the M-files in a directory. It can help you polish M-files before providing them to others to use.

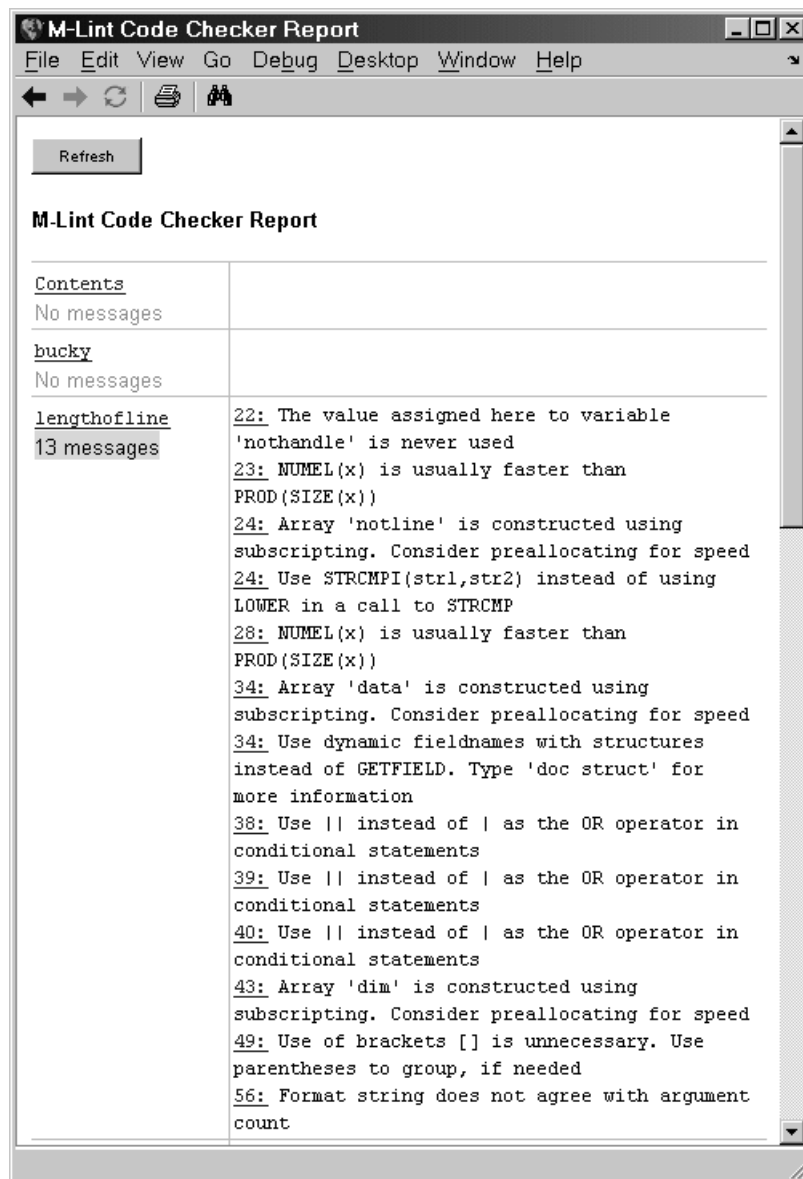
Click the Show Visual Directory button  on the Current Directory browser toolbar. The view changes—see the following figure for an example. To return to the Classic view of the Current Directory browser, click the button  again. For more information, see “Visual Directory in Current Directory Browser” in the online documentation.



Directory Reports in the Current Directory Browser

In the Current Directory browser, select **View -> Directory Reports** and select the type of report to run. The report appears as an HTML document in the MATLAB Web browser. A summary of the reports follows. For more information, see “Directory Reports in Current Directory Browser” in the online documentation.

M-Lint Code Check Report. The M-Lint report displays potential errors and problems, as well as opportunities for improvement in your code. For example, one common message is that a variable is defined but never used. You can also produce an M-Lint report for specified files using the `mlint` function, or run the M-Lint report from the Editor/Debugger or Profiler.



TODO/FIXME Report. The TODO/FIXME report shows M-files that contain text strings you included as notes to yourself, such as TODO.

Help Report. The Help report presents a summary view of the help component of your M-files. Use this information to help you identify files of interest or to help you identify files that lack help information.

Contents Report. The Contents report displays information about the integrity of the Contents.m file for the directory. A Contents.m file is a file you create that provides a brief description for relevant M-files in the directory. When you type help followed by the directory name, such as help mydemos, MATLAB displays the information in the Contents.m file. Use the Contents report to help you clean up and maintain your Contents.m file. If there is no Contents.m file, use the Contents report to create one.

Dependency Report. The Dependency report shows all M-files called by each M-file, or in other words, shows all children of each M-file. Use this report to determine all files you need to provide to someone who wants to run an M-file.

File Comparison Report. The File Comparison report identifies the differences between two files in the current directory. For example, you can easily compare an autosaved version of a file to the latest version of the file.

Coverage Report. Run the Coverage report after you run the Profile report to identify what percentage of the file was executed when it was profiled.

Profiler for Measuring Performance

- Access the Profiler from the **Desktop** menu or the Editor/Debugger **Tools** menu.
- In the Profiler summary report, click a column name to sort the report by that column.

- In the Profiler detail report, specify options to show busy lines (lines where the most time was spent) and to show the file listing (the M-file code). Other options allow you to run the M-Lint Code Check report, which provides messages for improving the file, and the Coverage report, which indicates how much of the file was exercised during profiling. For more information about these reports, see “Directory Reports in Current Directory Browser” in the online documentation. After selecting an option in the detail report, click **Refresh** to update the report. The performance acceleration information in the detail report has been removed.
- The profile report previously supported in MATLAB is no longer available. This was the report you generated by running `profile report` or `profreport`. There is a new function, `profsave` that replaces `profreport`. The `profsave` function saves a static version of the HTML profile report.

Publishing Results

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

If you are using the Help browser, watch the new Publishing M Code from the Editor video demo for an overview of the major functionality.

You can publish M-files to HTML, XML, LaTeX, Word, and PowerPoint documents. The published documents can include code, formatted comments, and results, such as graphs in Figure windows. Use cells and cell publishing features in the Editor/Debugger. For details, see “Publishing to HTML, XML, LaTeX, PowerPoint, and Word Using Cells” in the online documentation.

Notebook. If you currently use Notebook, consider using cell publishing from the Editor instead, which provides more features and flexibility for most applications.

Notebook has been improved with regards to speed and stability, with a few minor changes in operation. The improvements were available via a Web-downloadable update to MATLAB version 6.5, and are now part of MATLAB version 7. For details about the differences, see Solution 36072 on the MathWorks Web site.

Mathematics Features

MATLAB 7.0 adds the following mathematics features and enhancements:

- “Nondouble Arithmetic” on page 1-25
- “New Class and Data Inputs for eps” on page 1-25
- “New Class Inputs for realmax and realmin” on page 1-26
- “New Functions intmax and intmin” on page 1-27
- “New Warnings for Integer Arithmetic” on page 1-27
- “New Class Inputs for ones, zeros, and eye” on page 1-29
- “New Class and Size Inputs for Inf and NaN” on page 1-29
- “New Class Inputs for sum” on page 1-30
- “New Functions for Numerical Data Types” on page 1-30
- “complex Now Accepts Inputs of Different Data Types” on page 1-31
- “Bit Functions Now Work on Unsigned Integers” on page 1-31
- “New Function linsolve for Solving Systems of Linear Equations” on page 1-32
- “New Function accumarray for Constructing Arrays with Accumulation” on page 1-32
- “Enhancements to Discrete Fourier Transform Functions” on page 1-33
- “Enhancements to lscov” on page 1-33
- “Enhanced Functions for Computational Geometry” on page 1-34
- “New and Enhanced Functions for Ordinary Differential Equations (ODEs)” on page 1-34
- “New Output Function for Optimization Functions” on page 1-35
- “New Support for Interpolation Functions” on page 1-35
- “Enhanced sort Capabilities and Performance” on page 1-37
- “New Input Argument for Incomplete Gamma Function” on page 1-37
- “New Function quadv Integrates Complex, Array-Valued Functions” on page 1-37
- “New Form for Generalized Hessian” on page 1-38

- “New Output for `polyeig`” on page 1-38
- “New Trigonometric Functions For Angles in Degrees” on page 1-38
- “Overriding the Default BLAS Library on Intel/Windows Systems” on page 1-40

New Nondouble Mathematics Features

MATLAB Version 7.0 now supports many arithmetic operations and mathematical functions on the following nondouble MATLAB data types:

- `single`
- `int8` and `uint8`
- `int16` and `uint16`
- `int32` and `uint32`

Most of the built-in MATLAB functions that perform mathematical operations now support inputs of type `single`. In addition, the arithmetic operators and the functions `sum`, `diff`, `colon`, and some elementary functions now support integer data types.

This section covers the following topics:

- “Nondouble Arithmetic” on page 1-25
- “New Class and Data Inputs for `eps`” on page 1-25
- “New Class Inputs for `realmax` and `realmin`” on page 1-26
- “New Class Inputs for `ones`, `zeros`, and `eye`” on page 1-29
- “New Functions `intmax` and `intmin`” on page 1-27
- “New Warnings for Integer Arithmetic” on page 1-27

Note In Version 7.0, MATLAB only supports mathematical operations on nondouble data types for built-in functions; it does *not* support these operations for M-file functions unless otherwise stated in the M-file help.

Nondouble Arithmetic

This section describes how MATLAB performs arithmetic on nondouble data types.

Single Arithmetic. You can now combine numbers of type `single` with numbers of type `double` or `single`. MATLAB performs arithmetic as if both inputs had type `single` and returns a result of type `single`. For more information, see “Single-Precision Mathematics” in the online MATLAB documentation.

Integer Arithmetic. You can now combine numbers of an integer data type with numbers of the same integer data type or type `scalar double`. MATLAB performs arithmetic as if both inputs had type `double` and then converts the result to the same integer data type.

MATLAB computes operations on arrays of integer data type using *saturating* integer arithmetic. Saturating means that if the result is greater than the upper bound of the integer data type, MATLAB returns the upper bound. Similarly, if the result is less than the lower bound of the data type, MATLAB returns the lower bound. For more information, see “Integer Mathematics” in the online MATLAB documentation.

New Class and Data Inputs for `eps`

You can now call the function `eps` with the syntax

```
eps(x)
```

If `x` has type `double`, `eps(x)` returns the distance from `x` to the next largest double-precision floating point number. This is a measure of the accuracy of `x` as a double-precision number. `eps(1)` returns the same value as `eps` with no input argument.

You can now replace expressions of the form

```
if Y < eps * abs(X)
```

with

```
if Y < eps(X)
```

If `x` has type `single`, `eps(x)` returns the distance from `x` to the next largest single-precision floating point number. This is a measure of the accuracy of `x` as a single-precision number.

The command

```
eps('single')
```

```
ans =
```

```
1.1921e-007
```

returns the same value as `eps(single(1))`. The value of `eps('single')` is the same as `single(2^-23)`. The command `eps('double')` returns the same result as `eps`.

See “`eps` in Double and Single-Precision Arithmetic” in the online MATLAB documentation for more information.

New Class Inputs for `realmax` and `realmin`

You can now call the function `realmax` with the syntax

```
realmax('single')
```

```
ans =
```

```
3.4028e+038
```

which returns the largest single-precision number. Similarly,

```
realmin('single')
```

returns the smallest single-precision number.

The commands `realmax('double')` and `realmin('double')` return the same results as `realmax` and `realmin`, respectively. See “Largest and Smallest Numbers” in the online MATLAB documentation for more information.

New Functions `intmax` and `intmin`

Two new functions, `intmax` and `intmin`, return the largest and smallest numbers, respectively, for integer data types. For example,

```
intmax('int8')  
  
ans =  
  
127
```

returns the largest number of type `int8`. See “Largest and Smallest Values for Integer Data Types” in the online MATLAB documentation for more information.

New Warnings for Integer Arithmetic

This section describes four new warning messages for integer arithmetic in Version 7.0. While these warnings are turned off by default, you can turn them on as a diagnostic tool or to warn of behavior in integer arithmetic that might not be expected.

To turn all four warning messages on at once, enter

```
intwarning on
```

Integer Conversion of Noninteger Values. MATLAB can now return a warning when it rounds up a number in converting to an integer data type. For example,

```
int8(2.7)  
Warning: Conversion rounded non-integer floating point value to  
nearest int8 value.  
  
ans =  
  
3
```

Integer Conversion of NaN. When MATLAB converts NaN (Not-a-Number) to an integer data type, the result is 0. MATLAB can now return a warning when this occurs. For example,

```
int16(NaN)
Warning: NaN converted to int16(0).

ans =

     0
```

Integer Conversion Overflow. MATLAB can now return a warning when you convert a number to an integer data type and the number is outside the range of the data type. For example,

```
int8(300)
Warning: Out of range value converted to intmin('int8') or
intmax('int8').

ans =

    127
```

Integer Arithmetic Overflow. MATLAB can now return a warning when the result of an operation on integer data types is either NaN or outside the range of that data type. For example,

```
int8(100) + int8(100)
Warning: Out of range value or NaN computed in integer arithmetic.

ans =

    127
```

To turn all of these warnings off at once, enter

```
intwarning off
```

New Class Inputs for ones, zeros, and eye

You can now call ones or zeros with an input argument specifying the data type of the output. For example,

```
ones(m, n, p, ..., 'single')
```

or

```
ones([m, n, p, ...], 'single')
```

returns an m-by-n-by-p-by ... array of type single containing all ones. zeros uses the same syntax.

You can now call eye with this input argument for two-dimensional arrays. For example,

```
eye(m, 'single')
```

returns an m-by-m identity matrix of type single. The command

```
eye(m, n, 'int8')
```

returns an m-by-n array of type int8.

New Class and Size Inputs for Inf and NaN

The functions Inf and NaN now accept inputs that enable you to create Infs or NaNs of specified sizes and floating-point data types. As examples,

- Inf('single') or NaN('single') create the single-precision representations of Inf and NaN, respectively.
- Inf(m,n,p, ...) or NaN(m,n,p, ...) create m-by-n-by-p-by-... arrays of Infs or NaNs, respectively.

See the reference pages for Inf and NaN for more information.

New Class Inputs for sum

The following new input arguments for `sum` control how the summation is performed on numeric inputs:

- `s = sum(x, 'native')` and `s = sum(x, dim, 'native')` accumulate in the native type of its input and the output `s` has the same data type as `x`. This is the default for `single` and `double`.
- `s = sum(x, 'double')` and `s = sum(x, dim, 'double')` accumulate in double-precision. This is the default for integer data types.

In Version 7.0, `sum` applied to a vector of type `single` performs `single` accumulation and returns a result of type `single`. In other words, `sum(x)` is the same as `sum(x, 'native')` if `x` has type `single`. This is a change in the behavior of `sum` from previous releases. To make `sum` accumulate in `double`, as in previous releases, use the input argument `'double'`.

New Functions for Numerical Data Types

MATLAB 7.0 contains three new functions for detecting and converting data types:

- `cast` enables you to cast a variable to a different data type or class
- `isfloat` enables you to detect floating-point arrays. `isfloat(A)` returns 1 if `A` has type `double` or `single` and 0 otherwise. `isfloat(A)` is the same as `isa(A, 'float')`.
- `isinteger` enables you to detect integer arrays. `isinteger(A)` returns 1 if `A` has integer data type and 0 otherwise. `isinteger(A)` is the same as `isa(A, 'integer')`

complex Now Accepts Inputs of Different Data Types

The function `complex` now accepts inputs of different data types when you use the syntax

```
complex(a,b)
```

according to the following rules:

- If either of `a` or `b` has type `single`, `c` has type `single`.
- If either of `a` or `b` has an integer data type, the other must have the same integer data type or type `scalar double`, and `c` has the same integer data type.

Bit Functions Now Work on Unsigned Integers

The following functions now work on unsigned integer inputs:

- `bitand`
- `bitcmp`
- `bitget`
- `bitor`
- `bitset`
- `bitshift`
- `bitxor`

Instead of using `flints` (integer values stored in floating point) to do your bit manipulations, consider using unsigned integers, as a more natural representation of bit strings. Instead of using `bitmax`, use the `intmax` function with the appropriate class name. For example, use `intmax('uint32')` if you are working with unsigned 32 bit integers.

In addition, the function `bitcmp` now accepts the following new syntax for inputs of type `uint8`, `uint16`, and `uint32`:

```
bitcmp(A)
```

`bitcmp` now uses the data type of `A` to determine how to take the bitwise complement.

New Function `linsolve` for Solving Systems of Linear Equations

The new `linsolve` function enables you to solve systems of linear equations of the form $Ax = b$ more quickly when the matrix of coefficients A has a special form, such as upper triangular. When you specify one of these special types of systems, `linsolve` is faster than `mldivide` or `\` (backslash) because it does not check whether the matrix actually has the form you specify.

Note If the matrix A does not have the form you specify in `opts`, `linsolve` returns incorrect results because it does not perform error checking. If you are unsure of the form of A , use `mldivide`, or `\` instead.

New Function `accumarray` for Constructing Arrays with Accumulation

The new `accumarray` function enables you to construct an array with accumulation. The following example uses `accumarray` to construct a 5-by-5 matrix A from a vector `val`. The function `accumarray` adds the entries of `val` to A at the indices specified by the matrix `ind`, which has the same number of rows as `val`. If an index in `ind` is repeated, the entries of `val` accumulate at the corresponding entry of A .

```
ind = [1 2 5 5;1 2 5 5]';
val = [10.1 10.2 10.3 10.4]';
A = accumarray(ind, val)
```

A =

```
10.1000         0         0         0         0
         0 10.2000         0         0         0
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0 20.7000
```

To get the (5,5) entry of A , `accumarray` adds the entries of `val` corresponding to repeated pair of indices (5,5).

```
A(5, 5) = 10.3 + 10.4
```

In general, if `ind` has `ndim` columns, `A` will be an `N`-dimensional array with `ndim` dimensions, whose size is `max(ind)`.

Enhancements to Discrete Fourier Transform Functions

The new function `fftw` enables you to optimize the speed of the discrete Fourier transform (DFT) functions `fft`, `ifft`, `fft2`, `ifft2`, `fftn`, and `ifftn`. You can use `fftw` to set options for a tuning algorithm that experimentally determines the fastest algorithm for computing a discrete Fourier transform of a particular size and dimension at run time.

The functions `ifft`, `ifft2`, and `ifftn` now accept the input argument `'symmetric'`, which causes these functions to treat the array `X` as conjugate symmetric. This option is useful when `X` is not exactly conjugate symmetric, merely because of round-off error.

Enhancements to `lscov`

The command

```
lscov(A,b,V)
```

now accepts either a weight vector or a covariance matrix for `V`. If you enter `lscov(A,b)` without a third argument, `lscov` uses the identity matrix for `V`.

The command `lscov(A, b, V, alg)` now enables you to specify the algorithm used to compute the result when `V` is a matrix. You can specify `alg` to be one of the following:

- `'chol'` uses the Cholesky decomposition of `V`
- `'orth'` uses the orthogonal decomposition of `V`

The command

```
[x stdx mse] = lscov(...)
```

now returns `mse`, the mean squared estimate (MSE).

The command

```
[x stdx mse S] = lscov(...)
```

now returns `S`, the estimated covariance matrix of `x`.

In addition, `lskov` can now accept a design matrix A that is rank deficient and a covariance matrix, V , that is positive semidefinite.

Enhanced Functions for Computational Geometry

The following functions, which perform geometric computations on a set of points in N -dimensional space, now provide many new options:

- `convull` — Compute convex hulls
- `convhulln` — Compute N -dimensional convex hulls
- `delaunay` — Construct Delaunay triangulation
- `delaunay3` — Construct 3-dimensional Delaunay tessellations
- `delaunayn` — Construct N -dimensional Delaunay tessellations
- `griddata` — Data gridding and surface fitting
- `griddata3` — Data gridding and surface fitting for 3-dimensional data
- `griddatan` — Data gridding and hypersurface fitting (dimensions ≥ 2)
- `voronoi` — Construct Voronoi diagrams
- `voronoin` — Construct N -dimensional Voronoi diagrams

These functions now accept an input cell array `options` that gives you greater control over how they perform calculations. These functions use the software `Qhull`, created at the National Science and Technology Research Center for Computation and Visualization of Geometric Structures (the Geometry Center). For more information on the available options, see <http://www.qhull.org/>.

New and Enhanced Functions for Ordinary Differential Equations (ODEs)

MATLAB 7.0 provides two new functions for solving implicit ODEs and extending solutions to ODEs, along with several enhancements to existing ODE-related functions:

- `ode15i`, which is new in Version 7.0, provides the capability to solve fully implicit ODE and DAE problems of the form $f(t, y, y') = 0$ with consistent initial conditions, i.e., $f(t_0, y_0, y'_0) = 0$. `ode15i` provides an interface that is similar to that of the other MATLAB ODE solvers and is as easy to use. A supporting function `decic` helps you calculate consistent initial conditions.

The existing functions `odeset` and `odeget` enable you to set integration properties that affect the problem solution. `deval` evaluates the numerical solution obtained with `ode15i`.

- `odextend`, which is new in Version 7.0, enables you to extend the solution to an ODE created by an ODE solver.
- `bvp4c` can now solve multipoint boundary value problems. To see an example of how to solve a three-point boundary value problem, enter `threebvp`. To see the code for the example, enter `edit threebvp`. Enter `help bvp4c` to learn more about `bvp4c`.

- `deval` can now evaluate the derivative of the solution to an ODE as well as the solution itself. The command

```
[psxint, spxint] = deval(sol,xint)
```

returns `spxint`, the value of the derivative to `sol`.

New Output Function for Optimization Functions

In MATLAB 7.0, you can create an output function for several optimization functions in MATLAB. The optimization function calls the output function at each iteration of its algorithm. You can use the output function to obtain information about the data at each iteration or to stop the algorithm based on the current values of the data. You can use the output function with the following optimization functions:

- `fminbnd`
- `fminsearch`
- `fzero`

See “Calling an Output Function Iteratively” for an example of how to use the output function.

New Support for Interpolation Functions

The following interpolation functions now have enhanced features:

- `interp1` — The command `YI = interp1(X,Y,XI)` now accepts a multidimensional array `Y` and returns an array of the correct dimensions. If `Y` is an array of size `[n,m1,m2,...,mk]`, `interp1` performs interpolation for

each m_1 -by- m_2 -by-...- m_k value in Y . If XI is an array of size $[d_1, d_2, \dots, d_j]$, YI has size $[d_1, d_2, \dots, d_j, m_1, m_2, \dots, m_k]$.

The command `pp = interp1(X,Y,'method','pp')` uses the specified method to generate the piecewise polynomial form (ppform) of Y . See the reference page for `interp1` for information about the available methods.

- `interp2`, `interp3`, and `interp n` — You can now pass in a scalar argument, `ExtrapVal`, which these functions return for any values of XI and YI that lie outside the range of values spanned by X and Y defining the grid. For example,

```
ZI = interp2(X,Y,Z,XI,YI,'method',ExtrapVal)
```

returns the value of `ExtrapVal` for any values of XI or YI that are outside the range of values spanned by X and Y .

- `ppval` now accepts multidimensional arrays returned by the `spline` function using the syntax

```
YY = ppval(spline(X,Y), XX)
```

Each entry of YY is obtained by evaluating `spline(X,Y)` at the corresponding value of XX .

- `spline` — The command `YY = spline(X,Y,XX)` now accepts a multidimensional array Y and returns an array of the correct dimensions. Note that `YY = spline(X,Y,XX)` is the same as `YY = ppval(spline(X,Y), XX)`.

If `spline(X, Y)` is scalar-valued, then YY is of the same size as XX . If `spline(X, Y)` is $[D_1, \dots, D_r]$ -valued, and XX has size $[N_1, \dots, N_s]$, then YY has size $[D_1, \dots, D_r, N_1, \dots, N_s]$, where $YY(:, \dots, :, J_1, \dots, J_s)$ is the value of `spline(X, Y)` at $XX(J_1, \dots, J_s)$. There are two exceptions to this rule:

- N_1 is ignored if XX is a row vector, that is, if N_1 is 1 and s is 2.
- `spline` ignores any trailing singleton dimensions of XX .

Enhanced sort Capabilities and Performance

Improved Performance

sort performance has been improved for numeric arrays of randomly ordered data. Although there is some performance improvement for all such numeric arrays, you should see the greatest improvement for integer arrays and multidimensional arrays.

Sort Direction

A new argument, `mode`, lets you specify whether sort returns the sorted array in ascending or descending order.

New Input Argument for Incomplete Gamma Function

The incomplete gamma function, `gammainc`, now accepts the input argument `tail`, using the syntax

```
Y = gammainc(X,A,tail)
```

`tail` specifies the tail of the incomplete gamma function when `X` is non-negative. The choices are for `tail` are 'lower' (the default) and 'upper'. The upper incomplete gamma function is defined as

$$1 - \text{gammainc}(x,a)$$

New Function `quadv` Integrates Complex, Array-Valued Functions

The new function `quadv` integrates complex, array-valued functions.

New Form for Generalized Hessian

The function `hess` has a new syntax of the form

$$[AA, BB, Q, Z] = \text{hess}(A, B)$$

where A and B are square matrices, and returns an upper Hessenberg matrix AA , an upper triangular matrix BB , and unitary matrices Q and Z such that

$$Q^*A^*Z = AA$$

and

$$Q^*B^*Z = BB$$

New Output for `polyeig`

The function `polyeig` can now return a vector of condition numbers for the eigenvalues, when you call it with the syntax

$$[X, E, S] = \text{polyeig}(A_0, A_1, \dots, A_p)$$

At least one of A_0 and A_p must be nonsingular. Large condition numbers imply that the problem is close to one with multiple eigenvalues.

New Trigonometric Functions For Angles in Degrees

The following new functions compute trigonometric functions of arguments in degrees.

| Function | Purpose |
|-------------------|---|
| <code>sind</code> | Compute the sine of an argument in degrees |
| <code>cosd</code> | Compute the cosine of an argument in degrees |
| <code>tand</code> | Compute the tangent of an argument in degrees |
| <code>cotd</code> | Compute the cotangent of an argument in degrees |
| <code>secd</code> | Compute the secant of an argument in degrees |
| <code>cscd</code> | Compute the cosecant of an argument in degrees |

The following new functions compute the inverse trigonometric functions and return the answer in degrees:

| Function | Purpose |
|-----------------|---|
| asind | Compute the inverse sine of an argument and return answer in degrees |
| acosd | Compute the inverse cosine of an argument and return answer in degrees |
| atand | Compute the inverse tangent of an argument and return answer in degrees |
| acotd | Compute the inverse cotangent of an argument and return answer in degrees |
| asecd | Compute the inverse secant of an argument and return answer in degrees |
| acscd | Compute the inverse cosecant of an argument and return answer in degrees |

New Functions for Computing Logarithms, Exponentials, and nth Roots

The following new functions compute logarithms, exponentials, and nth roots of real numbers.

| Function | Purpose |
|-----------------|--|
| expm1 | Compute $\exp(x) - 1$ accurately for small values of x |
| log1p | Compute $\log(1+x)$ accurately for small values of x |
| nthroot | Compute the real nth root of a real number |

Overriding the Default BLAS Library on Intel/Windows Systems

Note Intel has used aggressive optimization to compile MKL. This optimization causes NaNs to be treated as zeros in some situations. Calculations that do not involve NaNs are done correctly. In some calculations that do involve NaNs, the NaNs will not propagate.

MATLAB uses the Basic Linear Algebra Subroutines (BLAS) libraries to speed up matrix multiplication and LAPACK-based functions like `eig`, `svd`, and `\mldivide`). At start-up, MATLAB selects the BLAS library to use.

For R14, MATLAB still uses the ATLAS BLAS libraries, however, on Windows systems running on Intel processors, you can switch the BLAS library that MATLAB uses to the Math Kernel Library (MKL) BLAS, provided by Intel.

If you want to take advantage of the potential performance enhancements provided by the Intel BLAS, you can set the value of the environment variable `BLAS_VERSION` to the name of the MKL library, `mk1.dll`. MATLAB uses the BLAS specified by this environment variable, if it exists.

To set the `BLAS_VERSION` environment variable, follow this procedure:

- 1 Click the **Start** button, go to the **Settings** menu, and select **Control Panel**.
- 2 On the **Control Panel** menu, select **System**.
- 3 In the **System Properties** dialog box, click the **Advanced** tab.
- 4 On the **Advanced** panel, click the **Environment Variables** button.
- 5 In the **Environment Variables** dialog box, click the **New** button in the User variables section.
- 6 In the **New User Variable** dialog box, enter the name of the variable as `BLAS_VERSION` and set the value of the variable to the name of the MKL library: `mk1.dll`.

Multithreading Disabled in Intel Math Kernel Library (MKL) BLAS

The Intel Math Kernel Library (MKL) is multithreaded in several areas. By default, this threading capability is disabled. To enable threading in the MKL library, set the value of the `OMP_NUM_THREADS` environment variable. Intel recommends setting the value of the `OMP_NUM_THREADS` variable to the number of processors you want to use in your application.

To set the value of this environment variable, follow the instructions outlined in “Overriding the Default BLAS Library on Intel/Windows Systems” on page 1-40.

Before enabling multithreading, read the Intel Math Kernel Library 6.1 for Windows Technical User Notes that explains certain limitations of this capability.

Programming Features

MATLAB 7.0 adds the following programming features and enhancements. For a list of new functions, see “Summary of New Functions” on page 1-49

Changes You Should Note

- “Case-Sensitivity in Function and Directory Names” on page 1-43
- “Differences Between Built-Ins and M-Functions Removed” on page 1-43
- “MATLAB Stores Character Data As Unicode” on page 1-44

Other Programming Features

- “New Calling Syntax for Function Handles” on page 1-45
- “Arrays of Function Handles” on page 1-46
- “Anonymous Functions” on page 1-46
- “Nested Functions” on page 1-48
- “Summary of New Functions” on page 1-49
- “New Features in Regular Expression Support” on page 1-50
- “Functions that Use Regular Expressions” on page 1-50
- “Changes to Error Message Format” on page 1-51
- “Cell Array Support for String Functions” on page 1-54
- “Freestyle Date String Format” on page 1-54
- “Additional Class Output From `mat2str`” on page 1-54
- “`datestr` Returns Date In Localized Format” on page 1-55
- “Form and Locale for weekday” on page 1-55
- “String Properties” on page 1-55
- “Bit Functions on Unsigned Integers” on page 1-56
- “`nargin` and `nargout` Now Work on Built-Ins” on page 1-56
- “`nargchk` Has a New Format for Error Messages” on page 1-56
- “Using `strtok` on Cell Arrays of Strings” on page 1-57
- “Protecting Files from Unwanted Deletion” on page 1-57
- “`inmem` Returns Path Information” on page 1-57

- “Accessing Cell and Structure Arrays Without deal” on page 1-58
- “Calling Private Functions From Scripts” on page 1-58
- “New Features for Nondouble Data Types” on page 1-58
- “Unicode-Based Character Classification” on page 1-58
- “Compressed Data Support in MAT-Files” on page 1-59
- “Comprehensive Function for Reading Text Files” on page 1-59
- “Saving Structures with the save Function” on page 1-60
- “New Data Import/Export Features” on page 1-60
- “MATLAB Performance Acceleration” on page 1-64
- ““Using MATLAB” Documentation Is Now Three Books” on page 1-65

Case-Sensitivity in Function and Directory Names

Prior to this release, filenames for MATLAB functions and Simulink[®] models, (M, P, MEX, DLL, and MDL files) and also directory names were interpreted somewhat differently by MATLAB with regards to case sensitivity, depending upon which platform you were running on. Specifically, earlier versions of MATLAB handled these names with case sensitivity on UNIX, but without case sensitivity on Windows.

This release addresses the issue of case sensitivity in an effort to make MATLAB consistent across all supported platforms. By removing these differences, we hope to make it easier for MATLAB users to write platform independent code.

This change is more fully discussed in “Case-Sensitivity in Function and Directory Names” on page 3-15, under “Programming Upgrade Issues.”

Differences Between Built-Ins and M-Functions Removed

MATLAB implements many of its core functions as built-ins. In previous releases of MATLAB, there have been several significant differences between the way MATLAB handles built-in and M-file functions. As of this release, MATLAB handles both types of functions the same. This change affects function dispatching and the output of the functions and which functions.

This change is more fully discussed in “Differences Between Built-Ins and M-Functions Removed” on page 3-19, under “Programming Upgrade Issues.”

MATLAB Stores Character Data As Unicode

Prior releases of MATLAB represented character data in memory using a system default character encoding scheme that was padded out to 16-bits. This was the case both in memory and in MAT-files. If this data needed to be accessible to multiple users, each user's system had to use the same character encoding scheme. For those users whose default encoding scheme differed, the exchange of character-oriented information was not possible.

In Release 14, this limitation is removed by adopting the Unicode character data encoding scheme in `mxArrays` and their storage in MAT-files. For more information regarding Unicode, consult the Unicode Consortium web site at <http://www.unicode.org>.

Changes to `save` and `matOpen`

MATLAB writes character data to MAT-files using Unicode character encoding by default. You can override this setting and use the default character set for your system instead by doing one of the following:

- From the MATLAB command line or a MATLAB function, save your data to the MAT-file using the command `save -v6`
- From a C mex file, open the MAT-file you will write the data to using the command `matOpen -wL`

See the individual reference pages for these functions for more information.

Caution If you have saved data to a MAT-file using MATLAB Release 14 Beta 2, please read “MAT-Files Generated By Release 14 Beta2 Must Be Reformatted” on page 3-13.

Character Rendering on Linux

Character data rendering has been improved for Linux operating systems that are configured with a UTF-8 default character set.

For More Information

For more information on saving character data using Unicode encoding, see “Writing Character Data” in the External Interfaces documentation. For information on the internal formatting of MAT-files, see the “MAT-File Format” document in the MATLAB documentation available in PDF format

Caution MAT-files saved in MATLAB version 7.0 without using the new `-v6` flag are not readable in previous versions of MATLAB. See “Making Release 14 MAT-files Readable in Earlier Versions” on page 3-13.

New Calling Syntax for Function Handles

You can now call functions by means of their related function handles using standard calling syntax rather than having to use `feval`. When calling a function using its handle, specify the function handle name followed by any input arguments enclosed in parentheses.

For the `parabola` function shown here, construct a function handle `h` and call the `parabola` function by means of the handle:

```
function y = parabola(a, b, c, x)
y = a*x.^2 + b*x + c;

parabHandle = @parabola;

parabHandle(1.3, .2, 30, 25)
```

When calling functions that take no input arguments, you must use empty parentheses after the function handle:

```
parabHandle()
```

For purposes of backward compatibility, the use of `feval` to evaluate function handles is still supported in this release. See “Function Handles and Backward Compatibility” on page 3-20.

Arrays of Function Handles

Previous releases of MATLAB supported arrays of function handles. You created such an array using the `[]` operator, and indexed into the array with the `()` operator:

```
x = [@sin @cos @tan];  
plot(feval(x(2), -pi:.01:pi));
```

In Release 14, MATLAB supports arrays of functions handles using cell arrays. You create and index into a function handle array using the `{}` operator:

```
x = {@sin @cos @tan};  
plot(x{2}(-pi:.01:pi));
```

For purposes of backward compatibility, standard arrays of function handles are still supported in this release. See “Function Handles and Backward Compatibility” on page 3-20.

Anonymous Functions

Anonymous functions give you a quick means of creating simple functions without having to create M-files each time. You can construct an anonymous function either at the MATLAB command line or from within another function or script.

Refer to “Anonymous Functions” in the MATLAB Programming documentation for more complete coverage of this topic. For more information on anonymous functions, open the M-file `anondemo.m` in the MATLAB Editor by typing

```
edit anondemo
```

Syntax

The syntax for creating an anonymous function from an expression is

```
fhandle = @(arglist) expr
```

where `arglist` is a comma-separated list of input variables, and `expr` is any valid MATLAB expression. The constructor returns a function handle, `fhandle`, that is mapped to this new function. Creating a function handle for an anonymous function gives you a means of invoking the function. It is also useful when you want to pass your anonymous function in a call to some other function.

Note Function handles not only provide access to anonymous functions. You can create a function handle to any MATLAB function. The constructor uses a different syntax: `fhandle = @functionname` (e.g., `fhandle = @sin`). To find out more about function handles, see “Function Handles” on page -27.

You can use the function handle for an anonymous function in the same way as any other MATLAB function handle.

A Simple Example

To create a simple function `sqr` to calculate the square of a number, use

```
sqr = @(x) x.^2;
```

To execute the function, type the name of the function handle, followed by any input arguments enclosed in parentheses:

```
a = sqr(5)
a =
    25
```

Since `sqr` is a function handle, you can pass it to other functions. The code shown here passes the function handle for anonymous function `sqr` to the MATLAB `quad` function to compute its integral from zero to one:

```
quad(sqr, 0, 1)
ans =
    0.3333
```

Arrays of Anonymous Functions

To store multiple anonymous functions in an array, use a cell array. See “Arrays of Anonymous Functions” in the MATLAB Programming documentation.

Examples

You can find more examples of how to use anonymous functions in MATLAB under “Examples of Anonymous Functions.”

Nested Functions

You can now define one or more functions within another function in MATLAB. These inner functions are said to be *nested* within the function that contains them. You can also nest functions within other nested functions.

Refer to “Nested Functions” in the MATLAB Programming documentation for more complete coverage of this topic. For more information on nested functions, open the M-file `nesteddemo.m` in the MATLAB Editor by typing

```
edit nesteddemo
```

Writing a Nested Function

To write a nested function, simply define one function within the body of another function in an M-file. Like any M-file function, a nested function contains any or all of the usual function components. In addition, you must always terminate a nested function with an end statement:

```
function x = A(p1, p2)
...
    function y = B(p3)
        ...
    end
...
end
```

Characteristics of Nested Functions

Two characteristics unique to nested functions are

- A nested function has access to the workspaces of all functions inside of which it is nested. A variable that has a value assigned to it by the primary function can be read or overwritten by a function nested at any level within the primary. Similarly, a variable that is assigned in a nested function can be read or overwritten by any of the functions containing that function.
- When you construct a function handle for a nested function, the handle not only stores the information needed to access the nested function; it also stores the values of all variables shared between the nested function and those functions that contain it. This means that these variables persist in memory between calls made by means of the function handle.

Examples

You can find examples of how to use nested functions in MATLAB under “Examples of Nested Functions.”

Summary of New Functions

These functions are new in this release.

| Function | Description |
|---------------------------------|---|
| <code>addtodate</code> | Modify a particular field of a date number |
| <code>genvarname</code> | Construct valid variable name from string |
| <code>intmax</code> | Return largest possible integer value |
| <code>intmin</code> | Return smallest possible integer value |
| <code>intwarning</code> | Control state of integer warnings |
| <code>isfloat</code> | Detect floating-point arrays |
| <code>isinteger</code> | Detect whether an array has integer data type |
| <code>isscalar</code> | Determine if item is a scalar |
| <code>isstrprop</code> | Determine the content of each element of a string |
| <code>isvector</code> | Determine if item is a vector |
| <code>mmfileinfo</code> | Get information about multimedia file |
| <code>recycle</code> | Set option to move deleted files to recycle folder |
| <code>restoredefaultpath</code> | Restore default search path |
| <code>strtrim</code> | Remove leading and trailing whitespace from string |
| <code>textscan</code> | Read data from text file, convert and write to cell array |
| <code>xlswrite</code> | Write matrix to a Microsoft Excel spreadsheet |

New Features in Regular Expression Support

This version of MATLAB introduces the following new features in regular expression support:

- **Multiple Input Strings** — You can use any of the MATLAB regular expression functions with cell arrays of strings as well as with single strings. Any or all of the input parameters (the string, expression, or replacement string) can be a cell array of strings.
- **Selective Outputs** — To select what type of data you want the `regexp` and `regexpi` to return (string indices or text, token indices or text, or token data by name) use one or more of the six qualifiers for these functions.
- **Lookaround Operators** — Lookahead and lookbehind operators enable you to match a pattern only if it is preceded, or followed, by another pattern.
- **New Logical Operators** — New operators for grouping, inserting comments, and finding alternative match patterns
- **New Quantifiers** — Lazy quantifiers match a minimum number of characters in a string. Possessive quantifiers do not reevaluate parts of the string that have already been evaluated
- **Element Grouping** — Group elements together using either `(...)` to group and capture, or `(?:...)` for grouping alone
- **Named Capture Grouping** — Capture characters in a token and assign a name to the token
- **Conditional Expressions** — Process a string in different ways depending on a stated condition
- **New character representations** — New symbolic representations such as `\e` for escape, or `\xN` for a character of hexadecimal value N, are available in this release.
- **Default Tokenizing** — The `regexprep` function now tokenizes by default. There is no longer a 'tokenize' option

Refer to “Regular Expressions” in the MATLAB Programming documentation.

Functions that Use Regular Expressions

The `who`, `whos`, `save`, `load`, and `clear` functions now accept regular expressions as input. This feature enables you to be more selective concerning which variables they operate on.

For example, this statement saves to a MAT-file only those variables with a name that either starts with the letters A or B, or contains ten or more characters:

```
save('mydata.mat', '-regexp', '^[AB].', '{10,}');
```

If the workspace contains the following four variables, two of the four meet the requirements of the regular expression:

```
whos
  Name                Size          Bytes  Class
  _____
  A_stats             10x5           400  double array
  X23456789           1x1            12  char array
  ab                  3x1           536  struct array
  longerVariableName  1x4             8  char array
```

When you perform the save operation and then check the contents of the MAT-file, you see that the variables with names that either start with A or have at least ten characters were saved:

```
save('mydata.mat', '-regexp', '^[AB].', '{10,}');
```

```
whos -file mydata.mat
  Name                Size          Bytes  Class
  _____
  A_stats             10x5           400  double array
  longerVariableName  1x4             8  char array
```

Refer to the reference pages for these functions for more information and examples.

Changes to Error Message Format

The last two lines of MATLAB error messages have changed for Release 14. Error messages now

- Display functions and subfunctions differently than in R13.
- Display nested functions.
- Call out the error in a string that you can use as input to other functions, like `dbstop`.
- Have a hot link to the source of the error.

Each of these changes is discussed below. Examples show the errors generated by both the previous release (V6.5) and current release (7.0) of MATLAB for the purpose of comparison.

Display of Functions and Subfunctions

MATLAB now calls out the source of the error using a consistent format. One of the features of this format is that you can place the string of the message into other MATLAB commands. See “Using the Error Message String as Input to Other Functions” on page 1-53.

Errors Generated by the Primary Function. Errors generated by the primary function of an M-file are displayed as shown below. In version 7.0, the path is not shown in most cases (private functions are one exception). Filename extension is also not shown. The failing line number is shown on third line.

In MATLAB V6.5 —

```
??? Error using ==> strcmp
Too many input arguments.

Error in ==> B:\MATLAB_V70\work\errmsgtest.m
On line 11 ==> strcmp('aa','bb','cc');
```

In MATLAB V7.0 —

```
??? Error using ==> strcmp
Too many input arguments.

Error in ==> errmsgtest at 11
strcmp('aa','bb','cc');
```

Errors Generated by a Subfunction. Errors generated by a subfunction of an M-file are displayed in the previous release and current release of MATLAB as shown below. Comments for primary functions apply here as well. Also, the name of the failing subfunction follows the > character instead of being put in parentheses.

In MATLAB V6.5 —

```
??? Error using ==> strcmp
Too many input arguments.
```

```
Error in ==> B:\MATLAB_V70\work\errmsgtest.m (subFun1)
On line 17 ==> strcmp('aa','bb','cc');
```

In MATLAB V7.0 —

```
??? Error using ==> strcmp
Too many input arguments.
```

```
Error in ==> errmsgtest>subFun1 at 17
strcmp('aa','bb','cc');
```

Error Messages Display Nested Functions

This example shows an error that comes from a nested function (`nestFun2`) called by another nested function (`nestFun1`). It uses the following syntax, where the `>` character follows the name of the primary function and precedes the names of any nested functions.

```
fun>nestfun1/nestfun2/etc at lineno.
```

In MATLAB V6.5 —

Nested functions are not supported prior to version 7.0.

In MATLAB V7.0 —

```
??? Error using ==> strcmp
Too many input arguments.
```

```
Error in ==> errmsgtest>nestFun1/nestFun2 at 6
strcmp('aa','bb','cc');
```

Using the Error Message String as Input to Other Functions

You can copy the text of the line that calls out the source of an error and use this string as input to some of the MATLAB debugging functions. The example shown below uses the string in a call to the `dbstop` function.

Copy the text that begins after

```
Error in ==>
```

In MATLAB V6.5 —

This feature is not supported prior to version 7.0.

In MATLAB V7.0 —

```
??? Error using ==> strcmp
Too many input arguments.
```

```
Error in ==> errmsgtest>nestFun1/nestFun2 at 6
strcmp('aa','bb','cc');
```

Copy and paste text of this error message into the `dbstop` command:

```
dbstop errmsgtest>nestFun1/nestFun2 at 6
```

Hot Link to the Source of an Error

Error messages now contain a blue-underlined hot link to the failing line of the M-file being executed.

Cell Array Support for String Functions

You can now pass a cell array of strings to the `strfind` function. MATLAB searches each string in the cell array for occurrences of the pattern string, and returns the starting index of each such occurrence.

Freestyle Date String Format

When converting between serial date numbers, date vectors, and date strings with the `datenum`, `datevec`, and `datestr` functions, you can specify a format for the date string from the Free-Form Date Format Specifiers table shown on the `datestr` reference page.

Additional Class Output From `mat2str`

The statement `str = mat2str(A, 'class')` creates a string with the name of the class of `A` included. This option ensures that the result of evaluating `str` will also contain the class information.

Change the 16-bit integer matrix to a string that includes 'int16'. Next, evaluate this string and verify that you get the same matrix that you started with:

```
x1 = int16([-300 407 213 418 32 -125]);

A = mat2str(x1, 'class')
A =
    int16([-300 407 213 418 32 -125])
x2 = eval(A);

isa(x2, 'int16') && all(x2 == x1)
ans =
     1
```

datestr Returns Date In Localized Format

The statement `str = datestr(..., 'local')` returns the date string in a localized format. See the `datestr` reference page for more information.

Form and Locale for weekday

The `weekday` function now takes two new inputs that control the output format. These arguments enable you to get a full or abbreviated day name, and a local or US English output.

String Properties

Use the new `isstrprop` function to see what parts of a string or array of strings are alphabetic, alphanumeric, numeric digits, hexadecimal digits, lowercase, uppercase, white-space characters, punctuation characters, contain control characters, or contain graphic characters.

For example, to test for alphabetic characters in a two-dimensional cell array, use

```
A = isstrprop({'abc123def'; '456ghi789'}, 'alpha')
A =
    [1x9 logical]
    [1x9 logical]

A{:,:}
ans =
    1 1 1 0 0 0 1 1 1
    0 0 0 1 1 1 0 0 0
```

Bit Functions on Unsigned Integers

MATLAB bit functions now work on unsigned integers. Instead of using flints (integer values stored in floating point) to do your bit manipulations, consider using unsigned integers. See “Bit Functions Now Work on Unsigned Integers” in the MATLAB Mathematics release notes.

nargin and nargsout Now Work on Built-Ins

In this release, you can now use the `nargin` and `nargout` functions to find out how many inputs and outputs are supported by a built-in function:

```
nargin('sprintf')
ans =
    2
nargout('sprintf')
ans =
    2
```

nargchk Has a New Format for Error Messages

When the `nargchk` function detects an error condition, it returns information on the error in either a string or a MATLAB structure. Use one of these two command syntaxes to specify which format to return. If neither is specified, `nargchk` returns a string:

```
msgstring = nargchk(minargs, maxargs, numargs, 'string')
msgstruct = nargchk(minargs, maxargs, numargs, 'struct')
```

The return structure has two fields: the message string, and a message identifier. When too few inputs are supplied, these fields are

```
message: 'Not enough input arguments.'  
identifier: 'MATLAB:nargchk:notEnoughInputs'
```

When too many inputs are supplied, the structure fields are

```
message: 'Too many input arguments.'  
identifier: 'MATLAB:nargchk:tooManyInputs'
```

Using strtok on Cell Arrays of Strings

You can now use the `strtok` function on a cell array of strings. When used with a cell array of strings, `strtok` returns a token output that is also a cell array of strings, each containing a token for its corresponding input string.

See the `strtok` reference page to see an example of how this works.

Protecting Files from Unwanted Deletion

To protect yourself from unintentionally deleting any files that you want to keep, use the new `recycle` function to turn on file recycling. When file recycling is on, MATLAB moves all files that you delete with the `delete` function to either the recycle bin (on the PC or Macintosh) or a temporary folder (on UNIX). When file recycling is off, any files you delete are actually removed from the system.

You can turn recycling on for all of your MATLAB sessions using the **Preferences** dialog box (Select **File** -> **Preferences** -> **General**). Under the heading **Default behavior of the delete function**, select **Move files to the Recycle Bin**.

inmem Returns Path Information

The `inmem` function now returns not only the names of the currently loaded M- and MEX-files, but the path and filename extension for each as well. Use the `-completenames` option to obtain this additional information:

```
inmem('-completenames')
```

Accessing Cell and Structure Arrays Without deal

In many instances, you can access the data in cell arrays and structure fields without using the `deal` function. Here is an example that reads each of the cells of a cell array into a separate output:

```
C = {rand(3) ones(3,1) eye(3) zeros(3,1)};
```

Use either of the following to access the cells in C:

```
[a,b,c,d] = deal(C{:})  
[a,b,c,d] = C{:}
```

Here is an example that reads each of the fields of a structure array into a separate output:

```
A.name = 'Pat'; A.number = 176554;  
A(2).name = 'Tony'; A(2).number = 901325;
```

Use either of the following to access the name field:

```
[name1,name2] = deal(A(:).name)  
[name1,name2] = A(:).name
```

Calling Private Functions From Scripts

You can now invoke a private function from a script, provided that the script is called from another M-file function, and that the private function being called by the script is within the scope of this M-file function.

New Features for Nondouble Data Types

The section “New Nondouble Mathematics Features” on page 1-24 describes new features affecting the nondouble (`single` and `integer`) data types. These changes affect `single` and `integer` arithmetic operations, and also conversion of `single` and `double` data types to integers.

Unicode-Based Character Classification

Unicode-based character classification APIs are now provided in MATLAB. The new character classification functions work with any locale or language and resolve all locale-specific issues that existed in prior releases.

Compressed Data Support in MAT-Files

The `save` function compresses your workspace variables as they are saved to a MAT-file. When writing a MAT-file that you will need to load using an earlier version of MATLAB, be sure to use the `save -v6` command. When you use the `-v6` switch, MATLAB saves the data without compression and without Unicode character encoding. This makes the resulting file compatible with MATLAB Version 6 and earlier.

You can also compress data when using MAT-file interface library functions (`matPut*`) to write to a MAT-file by opening the file with the command `matOpen wz`. See the section “Compressing Data” in the `save` reference page for more information on this feature.

Comprehensive Function for Reading Text Files

The new `textscan` function reads data from an open text file into a cell array. MATLAB parses the data into fields and converts it according to conversion specifiers passed to `textscan` in the argument list.

The `textscan` function is similar to `textread` but differs from `textread` in the following ways:

- The `textscan` function offers better performance than `textread`, making it a better choice when reading large files.
- With `textscan`, you can start reading at any point in the file. Once the file is open, (`textscan` requires that you open the file first), you can seek to any position in the file and begin the `textscan` at that point. The `textread` function requires that you start reading from the beginning of the file.
- Subsequent `textscan` operations start reading the file at the point where the last `textscan` left off. The `textread` function always begins at the start of the file, regardless of any prior `textread`.
- `textscan` returns a single cell array regardless of how many fields you read. With `textscan`, you don't need to match the number of output arguments to the number of fields being read as you would with `textread`.
- `textscan` offers more choices in how the data being read is converted.
- `textscan` offers more user-configurable options.

Saving Structures with the save Function

Two new syntaxes for the save function enable you to save individual fields of a structure to a file. See the function reference for save for more information.

To save all fields of the scalar structure `s` as individual variables within the file, `myfile.mat`, use

```
save('myfile', '-struct', 's')
```

To save as individual variables only those structure fields specified (`s.f1`, `s.f2`, ...), use

```
save('myfile', '-struct', 's', 'f1', 'f2', ...)
```

New Data Import/Export Features

MATLAB includes the following new features for importing and exporting data.

- “New Features in the `xlsread` Function” on page 1-60
- “New Features in `dlmwrite` Function” on page 1-61
- “Importing Complex Arrays” on page 1-62
- “Using `imread` to Import Subsets of TIFF Images” on page 1-62
- “Getting Information about Multimedia Files” on page 1-64
- “All-Platform Audio Recording and Playback” on page 1-64
- “FTP File Operations” on page 1-64
- “Web Services (SOAP)” on page 1-64

New Features in the `xlsread` Function

The table below shows new input and output arguments to the `xlsread` function. See the function reference for `xlsread` for more information. With the

exception of the **basic** input argument, these arguments are supported only on computer systems capable of starting Excel as a COM server from MATLAB.

| New Input Arguments | Description |
|----------------------------|--|
| -1 | Opens the Excel file in an Excel window, enabling you to interactively select the worksheet to be read and the range of data to import from the worksheet. |
| range | Reads data from the rectangular region of a worksheet specified by range. |
| basic | Imports data from the spreadsheet in basic import mode. |

| New Output Argument | Description |
|----------------------------|---|
| rawdata | Returns unprocessed cell content in a cell array. This includes both numeric and text data. |

Note If you use the `xlsread` function for reading date values, you should also read the release notes section, “Reading Date Values with `xlsread`” on page 3-23.

New Features in `dlmwrite` Function

The `dlmwrite` function now has several new input arguments plus an optional attribute-value format in which to enter these arguments. You can now enter input arguments to `dlmwrite` in an attribute-value format. This format enables you to specify just those arguments that you need and omit any others. This new syntax for `dlmwrite` is

```
dlmwrite('filename', M, attribute1, value1, ...
        attributeN, valueN)
```

The former syntax for `dlmwrite` is still supported for arguments that were available in earlier versions of MATLAB.

The table below shows new input arguments to the `dlmwrite` function. You must specify these new arguments using the attribute-value format. See the function reference for `dlmwrite` for more information.

| Attribute | Value |
|------------------|--|
| append | Either overwrite or append to the file |
| delimiter | Delimiter string to be used in separating matrix elements |
| newline | Character(s) to use in terminating each line |
| roffset | Offset, in rows, from the top of the destination file to where matrix data is to be written |
| coffset | Offset, in columns, from the left side of the destination file to where matrix data is to be written |
| precision | Numeric precision to use in writing data to the file |

For example, to export matrix `M` to file `myfile.txt`, delimited by the tab character, and using a precision of six significant digits, type

```
dlmwrite('myfile.txt', M, 'delimiter', '\t', 'precision', 6)
```

Importing Complex Arrays

The `csvread`, `dlmread`, and `textscan` functions import any complex number as a whole into a complex numeric field, converting the real and imaginary parts to the specified numeric type. Valid forms for a complex number are

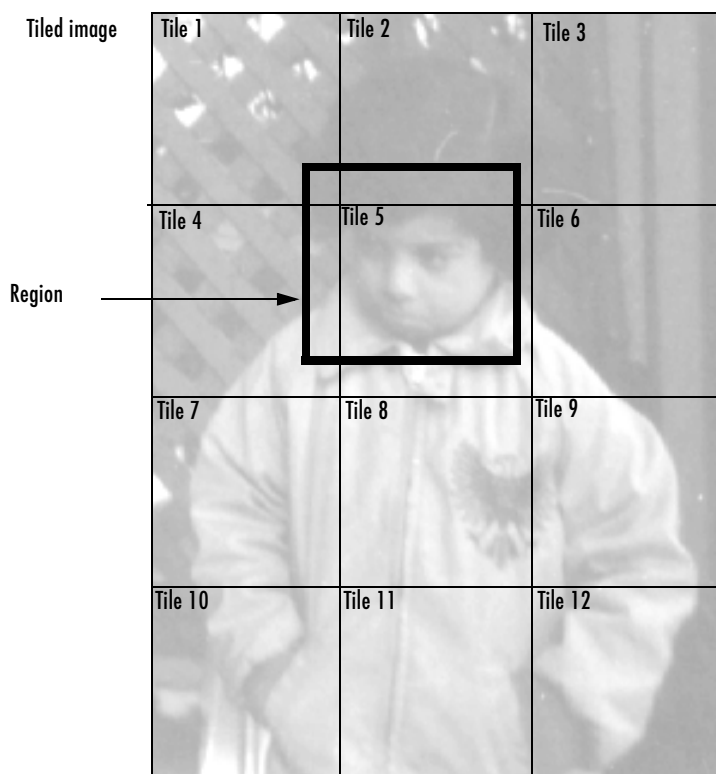
| Form | Example |
|-------------------|----------------|
| -<real>-<imag>i j | 5.7-3.1i |
| -<imag>i j | -7j |

Using `imread` to Import Subsets of TIFF Images

Using the `imread` function with the `'PixelRegion'` parameter, you can now read in portions of an image stored in TIFF format. As the value of this parameter, you specify a cell array containing two vectors: `ROWS` and `COLS`. Each

vector can either be a two-element vector specifying the extent of the region, [START STOP], or a three-element vector that enables downsampling, [START INCREMENT STOP].

When used with tiled images, 'PixelRegion' subsetting can improve memory usage and performance because it only reads in the tiles that encompass the region. For example, in the following figure, if you specify the region defined by the box, `imread` would only read in tiles 1, 2, 4, and 5.



Getting Information about Multimedia Files

MATLAB now includes a function, named `mmfileinfo`, that returns information about the contents of a multimedia file. The file can contain audio data, video data, or both.

This function is only available on Windows platforms.

All-Platform Audio Recording and Playback

The MATLAB `audiorecorder` and `audioplayer` functions can now be used on Windows and UNIX platforms. These functions were previously only available on Windows systems.

Note The `audiorecorder` and `audioplayer` objects are now implemented as MATLAB objects on all platforms. The methods supported by these objects are overloaded functions. You must use standard MATLAB function calling syntax to call methods of these objects; you cannot use dot notation.

FTP File Operations

From within MATLAB, you can connect to an FTP server to perform remote file operations. For more information, see the `ftp` reference page.

Web Services (SOAP)

MATLAB can now consume Simple Object Access Protocol-based (SOAP) Web services with the `createClassFromWSDL` function. For more information, see “Using Web Services in MATLAB” in the online documentation.

MATLAB Performance Acceleration

Release 13 introduced a new performance acceleration feature built into MATLAB. Enhancing performance in MATLAB is an ongoing development project that continues to show significant improvements in the performance of MATLAB programs.

The Performance Acceleration documentation written for Release 13 included suggestions on specific techniques to make the most of this feature. In this release, many of those techniques are no longer necessary. This documentation

has been replaced with more general suggestions on how to improve the performance of your programs.

“Using MATLAB” Documentation Is Now Three Books

Due to the increasing size of the printed “Using MATLAB” manual, we have divided it up into three separate printed books in version 7.0 to make it more manageable. The titles for these books (and their corresponding headings in the MATLAB Help Browser) are

- Desktop Tools and Development Environment
- Mathematics
- Programming

The online structure of this documentation is very similar to what it has been in previous releases, although some topics are now covered more thoroughly. We hope that you find this new format easier to use.

Graphics and 3-D Visualization Features

If you are using the Help browser, view the Graphics new features video demo to see highlights of the new features.

The following sections introduce the new features and enhancements added in MATLAB 7.0.

- “Plotting Tools” on page 1-66
- “Code Generation” on page 1-67
- “Data Exploration Tools” on page 1-68
- “Annotation Features” on page 1-68
- “Plot Objects” on page 1-69
- “Group Objects” on page 1-70
- “Linking Graphics Object Properties” on page 1-70
- “New Behavior for Hold Command” on page 1-70
- “Enhancements to findobj” on page 1-71
- “New Axes Properties” on page 1-71
- “New Figure Properties” on page 1-71
- “New Rootobject Property” on page 1-72

Plotting Tools

If you are using the Help browser, watch the new Plotting Tools video demo for an overview of the major functionality.

The following list links to new or redesigned plotting tool features.

- Figure Toolbars — figure toolbars that provide data exploration, plot editing, and annotation tools
- Interactive Plotting Tools — overview of plotting tools
 - Figure Palette
 - Plot Browser
 - Property Editor

Related functions:

- `plottools`
- `figurepalette`
- `plotbrowser`
- `propertyeditor`

Plotting Tools Not Working on Macintosh

The plotting tools not supported on the Macintosh platform. This means the Figure Palette, Plot Browser, and Property Editor do not work these platforms. To use the MATLAB 6 Property Editor, see the `propedit` command.

Using `-nojvm` Option Prevents Plotting Tools Use

If you use the `-nojvm` option when starting MATLAB, the plotting tools are not available.

MATLAB 6 Version Property Editor

The MATLAB 6 Property Editor is being replaced by a new Property Editor, which is available in this release. However, you can still access the MATLAB 6 Property Editor by issuing the following command.

```
propedit(object_handle, 'v6')
```

Without the `v6` argument, `propedit` displays the new Property Editor. See the `propedit` function for more information. Note that the Property Editor might not work with all objects.

Code Generation

You can save a graph as an M-file that contains the code to regenerate the graph. See [Generating an M-File to Recreate a Graph](#) for more information.

Data Exploration Tools

The following list links to the documentation for the data exploration tools.

- [Data Cursor](#) — displaying data values interactively
- [Zooming](#) — 2-D and 3-D zoom tools
- [Panning](#) — repositioning you view of the graph
- [Rotate 3D](#) — interactive rotation of 3-D views
- [Camera Toolbar](#) — mouse-controlled 3-D view manipulation.

Annotation Features

The following list links to the documentation for annotation features and properties of the annotation objects.

- [Overview of annotation features](#)
- [Rectangles and ellipses](#)
 - [Rectangle properties](#)
 - [Ellipse properties](#)
- [Textbox annotations](#)
 - [Textbox properties](#)
- [Lines and arrows](#)
 - [Line properties](#)
 - [Arrow properties](#)
 - [Textarrow properties](#)
 - [Doublearrow properties](#)
- [Adding a Colorbar to a graph](#) — new positioning options and colormap modification.
 - [colorbar](#) — new command options
- [Adding a legend to a graph](#) — new positioning and appearance options
 - [legend](#) — new command options
- [Pinning](#) — attaching annotation objects to a point in the figure
- [Aligning and Distributing](#) graphics objects

See the `annotation` function for information on programmatic access to annotation objects.

See [Annotation Objects](#) for an overview of this type of graphics object.

Plot Objects

Plot Objects are composite graphics objects that simplify the modification of graphs that employ them. The following list links to reference pages for modified graphing functions and to property descriptions of the new plot objects.

See [Plot Objects](#) for an overview of this type of graphics object.

Functions That Use Plot Objects

- `area`
- `bar`
- `contour`
- `errorbar`
- `plot`, `plot3`, `loglog`, `semilogx`, `semilogy`
- `quiver`, `quiver3`
- `scatter`, `scatter3`
- `stairs`
- `stem`, `stem3`
- `surf`, and `mesh` group

Note that all of the above functions have a `'v6'` optional argument that causes each function to return the core graphics objects that were created in previous releases. See the reference pages for more information.

Plot Objects

- `areaseries`
- `barseries`
- `contourgroup`
- `errorbarseries`
- `lineseries`

- `quivergroup`
- `scattergroup`
- `stairseries`
- `stemseries`
- `surfaceplot`

Refreshing Data Source Properties

The `refreshdata` function enables you to take advantage of the `XDataSource`, `YDataSource`, and `ZDataSource` plot objects properties to update graph data when workspace variables change values.

See [Specifying a Data Source](#) for more information.

Group Objects

Group objects enable you treat a number of objects as one, with respect to certain properties.

See [Group Objects](#) for an overview of this type of graphics object.

Group Object Functions

- `hggroup`
- `hgtransform`
- `makehgtform`

Linking Graphics Object Properties

You can link the corresponding properties of graphics objects so that changing any one object's properties makes the same change to all the linked objects.

See `linkprop` and `linkaxes` for more information.

New Behavior for Hold Command

The `hold` command has a new option `all`. This option holds the plot and the current line color and line style so that subsequent graphing commands do not reset the `ColorOrder` or `LineStyleOrder` property values to the beginning to the list.

Enhancements to findobj

The `findobj` function now supports logical operators and regular expressions. See the `findobj` reference page for more information.

New Axes Properties

You can control the behavior of an axes within a resized figure using the following new axes properties.

- `OuterPosition` — The boundary of the axes including the axis labels, title, and a margin. For figures with only one axes, this is the interior of the figure.
- `ActivePositionProperty` — Specifies whether to use the `OuterPosition` or the `Position` property as the size to preserve when resizing the figure containing the axes.
- `TightInset` — The margins added to the width and height of the `Position` property to include text labels, title, and axis labels.

See [Automatic Axes Resize](#) for more information.

New Figure Properties

There are three new figure properties described below.

Figure `KeyPressFcn` Property

The figure `KeyPressFcn` property now supports an event structure that returns information about the key press event. See the `KeyPressFcn` description for more information.

`DockControls` and `WindowState` Properties

Figures now have a `DockControls` property that determines if the **Desktop** menu appears on the figure. Setting `dockable` to `on` causes the menu to be displayed, the default setting of `off` prevents the menu from being displayed. You can always dock and undock the figure by setting the figure `WindowState` property.

Note that, depending on your preference settings, the figure might first be grouped into a Document window, which can then be docked in the Desktop.

See [Docking Figures in the Desktop](#) for more information.

New Rootobject Property

The `MonitorPosition` property enables you to get the position (width, height, and location) of multiple monitors connected to your computer.

New Dialog for Exporting Figures

You can export MATLAB figures to a variety of standard file formats using the Export Setup dialog. To display the dialog, select **Export Setup** from the figure **File** menu.

Export Setup provides easy access to the graphics properties that affect exported figures. For example, it enables you to control the size of the figure, the font size and type, whether to use line styles or solid lines, and so on.

You can save your own export setting as an export style, or you can use predefined options optimized for PowerPoint and MSWord.

The following picture shows the major components of the Export Setup dialog.

Select the category of properties you want to set.

Select an export format.

Apply the setting to the figure.

Save your own export setting or use predefined styles.

The image shows a dialog box titled "Export Setup: Figure 1". It is divided into two main sections: "Properties" and "Export Styles".

- Properties Section:** Contains a list of property categories on the left: "Size", "Rendering", "Fonts", and "Lines". The "Size" category is selected. To the right of this list are controls for "Width" (set to "auto"), "Units" (set to "inches"), and "Height" (set to "auto"). There is also a checkbox labeled "Expand axes to fill figure" which is currently unchecked.
- Export Styles Section:** Contains three rows of controls:
 - "Load settings from:" with a dropdown menu showing "default" and a "Load" button.
 - "Save as style named:" with a dropdown menu showing "default", "PowerPoint", and "MSWord", and a "Save" button.
 - "Delete a style:" with a dropdown menu and a "Delete" button.
- Buttons:** On the right side of the dialog, there are five buttons: "Apply to Figure", "Restore Figure", "Export...", "OK", and "Cancel".

Annotations with arrows point to various elements:

- "Select the category of properties you want to set." points to the "Size" category in the Properties list.
- "Select an export format." points to the "PowerPoint" option in the "Save as style named:" dropdown.
- "Apply the setting to the figure." points to the "Apply to Figure" button.
- "Save your own export setting or use predefined styles." points to the "Load settings from:" dropdown.

External Interfaces/API Features

MATLAB 7.0 adds the following external interfaces/API features and enhancements:

Importing and Exporting

- “Saving Character Data with Unicode Encoding” on page 1-75
- “Saving Data in Compressed Format” on page 1-75
- “Large File I/O for MEX-Files” on page 1-75

General Features

- “New mx Functions” on page 1-75
- “Automatic Registration of Automation Server on Installation” on page 1-76
- “Support for Multiple COM Type Libraries” on page 1-76
- “COM Interface Supports Custom Interfaces” on page 1-76
- “COM Data Type Support for Scripting Languages” on page 1-77
- “Additional ProgIDs for Latest MATLAB Version” on page 1-78
- “Connecting to an Existing MATLAB Server” on page 1-78
- “Graphical Interface to Listing Available ActiveX Controls” on page 1-79
- “Graphical Interface to Creating ActiveX Controls” on page 1-79
- “New Functions for the MATLAB COM Interface” on page 1-81
- “COM Interface Supports Dot Syntax in Commands” on page 1-81
- “Enumeration in COM Method Arguments” on page 1-82
- “Event Handling for COM Servers” on page 1-82
- “Callbacks to COM Event Handlers Written as Subfunctions” on page 1-83
- “Event Handlers Can Be Function Handles” on page 1-83

MATLAB Interface to Java

- “Java Interface Adds Dynamic Java Class Path” on page 1-83
- “Locating Java Native Method DLLs with File librarypath.txt” on page 1-84

Also see the section, “External Interface/API Upgrade Issues” on page 3-28 for information that may affect you when upgrading to this new release of MATLAB.

Saving Character Data with Unicode Encoding

The save function now saves character data to a MAT-file using Unicode character encoding by default. You can use your system’s default character encoding scheme instead by specifying the -v6 option with save. See “MATLAB Stores Character Data As Unicode” on page 1-44 for a full description of this change.

Caution MAT-files saved in MATLAB version 7.0 without using the new -v6 flag will not be readable in previous versions of MATLAB. See “Making Release 14 MAT-files Readable in Earlier Versions” on page 3-13.

Saving Data in Compressed Format

The save function now saves data to a MAT-file in a compressed format by default. See “Compressed Data Support in MAT-Files” on page 1-59 for more information.

Large File I/O for MEX-Files

MATLAB supports the use of 64-bit file I/O operations in your MEX-file programs. This enables you to read and write data to files that are up to and greater than 2 GB ($2^{31}-1$ bytes). Note that some operating systems or compilers may not support files larger than 2 GB.

See “Large File I/O” in the External Interfaces documentation for more information.

New mx Functions

New functions `mxIsInt64` and `mxIsUint64` return true if an `mxArray` represents its data as signed or unsigned 64-bit integers respectively.

Automatic Registration of Automation Server on Installation

When installing previous versions of MATLAB, system administrators also had to run MATLAB at least once on each machine to register the Automation server. In MATLAB 7.0, the MATLAB installation software does the Automation server installation for you.

Support for Multiple COM Type Libraries

MATLAB now fully supports importing additional type libraries from within an IDL file. Any COM object that depends on an imported type library is now handled correctly.

COM Interface Supports Custom Interfaces

MATLAB now supports custom interfaces to a server component in configurations where MATLAB is the client controlling an ActiveX control, or an in-process or out-of-process server. For those COM components that implement one or more custom interfaces, you can list the interfaces in MATLAB using the new `interfaces` function:

```
h = actxserver('ComponentA.CustomObject')
h =
    COM.componenta.customobject

customlist = interfaces(h)
customlist =
    ICustomObject1
    ICustomObject2
```

Once you select the custom interface that you want, use the `invoke` function to get a handle to it:

```
c1 = invoke(h, 'ICustomObject1')
c1 =
    Interface.componenta_Type_Library.ICustomObject1_Interface
```

You can now use this handle with most of the COM client functions to access the properties and methods of the object through this custom interface. For example, to list the methods available through the `ICustomObject1` interface, use

```
invoke(c1)
  Add = double Add(handle, double, double)
  CustomMethod1 = HRESULT CustomMethod1(handle, int32)
  CustomMethod2 = HRESULT CustomMethod2(handle, int32)
  TripleAdd = [double, double] TripleAdd(handle, double, double)
  method3 = [string, int32, string, string] method3(
    handle, int16, int32, double, string)
  outin = [double, double, double, double] outin(
    handle, double, double)
  strings = string strings(handle, string)
```

You can read more about this feature in the section, “Getting Interfaces to the Object” in the External Interfaces documentation.

COM Data Type Support for Scripting Languages

In previous versions of MATLAB, a COM client program written in VBScript could not retrieve numeric data from or write data to the workspace of a MATLAB client. This was because VBScript does not support the `SAFEARRAY` data type used by MATLAB to pass numeric data to and from the server workspace using the `GetFullMatrix` and `PutFullMatrix` functions.

Release 14 adds two new functions, `GetWorkspaceData` and `PutWorkspaceData`, that pass data using the variant data type, a type that is supported by VBScript. You can use these new functions to pass either numeric or string data to any workspace in the COM server running MATLAB.

Refer to “Exchanging Data with the Server” in the External Interfaces documentation.

Additional ProgIDs for Latest MATLAB Version

There are three additional COM programmatic identifiers (ProgIDs) in MATLAB 7.0:

```
MATLAB.Autoserver  
MATLAB.Autoserver.Single  
MATLAB.Autoserver.7
```

Using any of these identifiers with the `actxserver` function guarantees that the MATLAB server you create always runs the latest version of MATLAB (version 7.0).

Note These new ProgIDs do not replace the `MATLAB.Application` identifier used in previous versions of MATLAB. You can continue using this ProgID, but there is no guarantee that `actxserver` will create a server running MATLAB 7.0.

Connecting to an Existing MATLAB Server

Instead of having to create new instances of a MATLAB server, clients can connect to an existing MATLAB automation server using the `GetObject` command. This sample Visual Basic program connects to a running MATLAB automation server, returning a handle `h` to that server. It then executes a simple plot command in the server:

```
Dim h As Object  
  
' Call GetObject (omit first argument).  
Set h = GetObject(, "matlab.application")  
  
' Handle h should be valid now. Test it by calling Execute  
h.Execute ("plot([0 18], [7 23])")
```

Graphical Interface to Listing Available ActiveX Controls

The `actxcontrollist` function enables you to see what COM controls are currently installed on your system. Type

```
list = actxcontrollist;
```

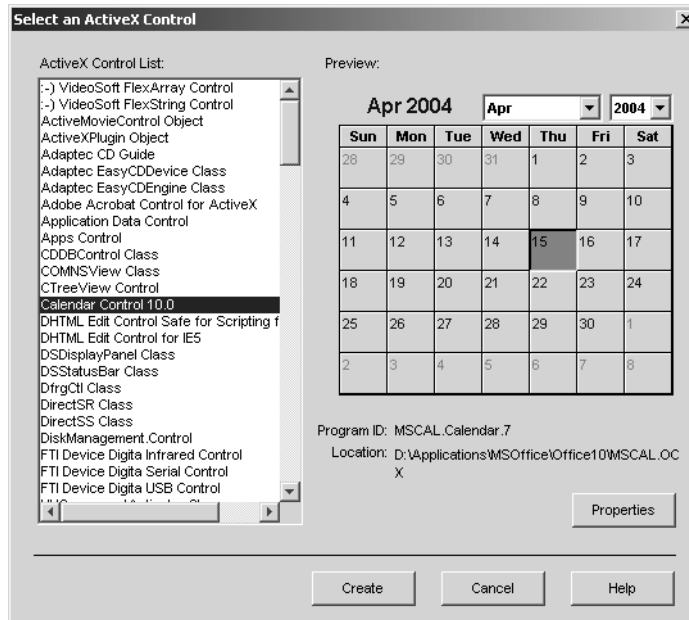
and MATLAB returns a list of each control, including its name, programmatic identifier (or ProgID), and filename, in the output cell array.

Refer to “Finding Out What Controls Are Installed” in the External Interfaces documentation.

Graphical Interface to Creating ActiveX Controls

The simplest way to create a control object is to use the `actxcontrolselect` function. This function displays a graphical interface that lists all controls installed on the system and creates the one that you select from the list.

The `actxcontrolselect` interface has a selection panel at the left of the window and a preview panel at the right. Click on one of the control names in the selection panel to see a preview of the control displayed. (If MATLAB cannot create the control, an error message is displayed in the preview panel.) Select an item from the list and click the **Create** button.



Refer to “Creating Control Objects Using a Graphical Interface” in the External Interfaces documentation.

New Functions for the MATLAB COM Interface

There are five new COM client functions.

| Function | Description |
|--------------------------------|---|
| <code>actxcontrollist</code> | List all currently installed ActiveX controls |
| <code>actxcontrolselect</code> | Display graphical interface for creating an ActiveX control |
| <code>interfaces</code> | List custom interfaces to a COM server |
| <code>iscom</code> | Determine if input is a COM or ActiveX object |
| <code>isinterface</code> | Determine if input is a COM interface |

There are three new COM server functions. When invoked by a MATLAB or Visual Basic client, these functions execute in the server associated with the specified handle parameter.

| Function | Description |
|-------------------------------|---|
| <code>Feval</code> | Evaluate MATLAB function call in the server |
| <code>GetWorkspaceData</code> | Get data from server workspace |
| <code>PutWorkspaceData</code> | Store data in server workspace |

See the function reference pages in the “External Interfaces Reference” documentation for more information.

COM Interface Supports Dot Syntax in Commands

You can now use a simpler form of syntax when invoking either MATLAB COM functions or methods belonging to COM objects. In this *dot syntax* (as it is referred to in the MATLAB documentation), you specify the object name, a dot (`.`), and then the name of the function or method you are calling. Enclose any input arguments in parentheses after the function name. Specify output arguments to the left of the equals sign:

```
outputvalue = object.function(arg1, arg2, ...)
```

For example, Release 13 syntax for invoking the `addproperty` function on a COM object with handle `h` was

```
invoke(h, 'addproperty', 'Position');
```

You can now perform the same operation using

```
h.addproperty('Position');
```

The get and set operations are even simpler:

```

** R13 SYNTAX **
x = get(h, 'Radius');
set(h, 'Radius', 50);

** R14 SYNTAX **
x = h.Radius;
h.Radius = 50;
```

Refer to “Invoking Commands on a COM Object” in the External Interfaces documentation.

Enumeration in COM Method Arguments

In addition to supporting enumeration for the properties of a COM object, MATLAB now supports enumeration for parameters passed to methods of a COM object. The only restriction is that the type library in use must report the parameter as `ENUM`, and only as `ENUM`.

Refer to “Specifying Enumerated Parameters” in the External Interfaces documentation.

Event Handling for COM Servers

In addition to handling events from ActiveX controls, MATLAB now handles events fired by Automation servers as well. Use the same event handling functions that you have been using for events from controls.

| Function | Description |
|-----------------------------|---|
| <code>eventlisteners</code> | Return a list of events attached to listeners |
| <code>events</code> | List all events, both registered and unregistered, a control or server can generate |
| <code>isevent</code> | Determine if an item is an event of a COM object |

| Function | Description (Continued) |
|---------------------|--|
| registerevent | Register an event handler with a control or server event |
| unregisterallevents | Unregister all events for a control or server |
| unregisterevent | Unregister an event handler with a control or server event |

Refer to “How to Prepare for and Handle Events from a COM Server” and “Example — Responding to Events from an Automation Server” in the External Interfaces documentation.

Callbacks to COM Event Handlers Written as Subfunctions

Instead of having to maintain a separate M-file for every event handler routine you write, you can consolidate some or all of these routines into a single M-file using M-file subfunctions.

Refer to “Writing Event Handlers Using M-File Subfunctions” in the External Interfaces documentation.

Event Handlers Can Be Function Handles

In this release, you can now implement ActiveX event handlers as function handles.

Java Interface Adds Dynamic Java Class Path

MATLAB loads Java class definitions from files that are on the Java class path. The Java class path now consists of two segments: the *static path*, and a new segment called the *dynamic path*.

The static path is loaded from the file `classpath.txt` at the start of each MATLAB session and cannot be changed without restarting MATLAB. This was the only path available in previous versions of MATLAB. Thus, there was no way to change the Java path without restarting MATLAB.

The dynamic Java class path can be loaded at any time during a MATLAB session using the `javaclasspath` function. You can define the dynamic path

(using `javaclasspath`), modify the path (using `javaaddpath` and `javarmpath`), and refresh the Java class definitions for all classes on the dynamic path (using `clear java`) without restarting MATLAB. See the function reference pages for more information on how to use these functions.

The `javaclasspath` function, when used with no arguments, displays both the static and dynamic segments of the Java class path:

```
javaclasspath

      STATIC JAVA PATH

D:\Sys0\Java\util.jar
D:\Sys0\Java\widgets.jar
D:\Sys0\Java\beans.jar
      .
      .

      DYNAMIC JAVA PATH

User4:\Work\Java\ClassFiles
User4:\Work\Java\mywidgets.jar
      .
      .
```

You can read more about this feature in the sections, “The Java Class Path” and “Making Java Classes Available to MATLAB” in the External Interfaces documentation.

Locating Java Native Method DLLs with File `librarypath.txt`

Previous versions of MATLAB required that you set a system environment variable to enable Java to locate the shared libraries supporting any native methods you need to use. This environment variable was `PATH` on Windows systems, and `LD_LIBRARY_PATH` on UNIX systems. This is no longer necessary.

Now you can enter the names of those directories that contain native method libraries in a new file called `librarypath.txt` using one line per directory. The `librarypath.txt` file resides adjacent to the similar file `classpath.txt` in the `$matlab/toolbox/local` directory.

Creating Graphical User Interfaces (GUIDE) Features

If you are using the Help browser, view the Creating Graphical User Interfaces new features video demo to see highlights of the major new features.

MATLAB 7.0 adds the following new features and enhancements for creating graphical user interfaces.

- “New Container Components” on page 1-85
- “ActiveX Controls” on page 1-86
- “New Toolbar Component” on page 1-86
- “Menu Editor Enhancements” on page 1-87
- “Layout Resize Behavior” on page 1-87
- “Key Press Detection” on page 1-88
- “Edit Text Box Scroll Bar” on page 1-88
- “Setting Uicontrol Focus” on page 1-88
- “Multiple Selection in uigetfile” on page 1-88
- “Program Suspension Time-Out” on page 1-89
- “Standard Dialog Box Push Buttons” on page 1-89

New Container Components

MATLAB 7.0 introduces two new container components,

- Panel — Groups components
- Button group – Groups components and manages exclusive selection behavior for radio buttons and toggle buttons.

These components are available in the GUIDE Layout Editor and via the functions `uipanel` and `uibuttongroup`.

A container component can be the child of a figure or another container. In general, containers can have as children the same components as figures, including other containers. However, they cannot have menu bars, toolbars, or ActiveX controls as children. The `Position` property of the child of a panel or button group is interpreted relative to the panel or button group. If you move the panel or button group, the components it contains automatically move with it and maintain their positions.

Panel properties and button group properties enable you to control the color, size, border, and position of the panel or button group, assign a title, and specify a context menu. In general, panels and button groups have many of the same properties as `uicontrol` objects.

Working with Container Components in GUIDE

For information about working with panels and button groups in GUIDE, see the following topics in the Creating Graphical User Interfaces collection of the MATLAB documentation.

- Adding Components to the Layout Area
- Working with Components in the Layout Area
- Front-to-Back Positioning
- Setting Panel and Button Group Properties
- Callback Properties
- Viewing the Object Hierarchy
- Setting the Tab Order

ActiveX Controls

GUIDE now enables you to insert an ActiveX control into your GUI if you are running MATLAB on Microsoft Windows. When you drag an ActiveX component from the component palette into the layout area, GUIDE displays a dialog in which you can select any registered ActiveX control on your system. When you select an ActiveX control and click **Create**, the control appears as a small box in the Layout Editor. You can then program the control to do what you want it to.

See MATLAB COM Client Support in the online MATLAB documentation and ActiveX Controls in the GUIDE documentation to learn more about ActiveX controls.

New Toolbar Component

A new function, `uitoolbar`, enables you to add a toolbar to a figure. You can add your own push tools and toggle tools to the toolbar with the `uipushtool` and `uitoggletool` functions.

Uipushtool properties enable you to provide a callback that responds to a mouse click. Uitoggletool properties enable you to provide callbacks that respond to the tool being set on, off, or toggled to either position. Properties for both uipushtools and uitoggletools provide for tooltip strings, separators, and truecolor images to display on the tools. In general, uitoolbar, uipushtool, and uitoggletool objects have many of the same properties as uicontrol objects.

Menu Editor Enhancements

The GUIDE Menu Editor now enables you to:

- Choose a keyboard accelerator for a menu item from a pop-up menu.
- Set an item's Enabled property on or off when the menu is first opened. If the property value is off, the item appears dimmed and the user cannot select it.
- Open the Property Inspector where you can change all uimenu properties.
- Display the callback subfunction in an editor. If the callback does not yet exist, GUIDE creates it before displaying it.

The Menu Editor is now better synchronized with other GUIDE tools:

- Property changes made in the Menu Editor or in the Property Inspector are immediately reflected in the other.
- uimenu objects in the Menu Editor now also appear in the Object Browser. If you select a uimenu object in either, it is automatically selected in the other.
- If a component is selected in the Layout Editor and you select a menu item in the Menu Editor, the component is deselected in the Layout Editor.

See Menu Editor in the MATLAB documentation for more information.

Layout Resize Behavior

In the GUIDE Layout Editor, components you have placed in the layout area now maintain their visual position relative to the upper left corner of their parent container (figure, panel, or button group) when you resize the container. However, the values of the Position property are determined relative to the lower left corner, and these values will change accordingly when you increase or decrease the height of the container.

Key Press Detection

A new `uicontrol` callback property, `KeyPressFcn`, specifies a key press callback function with which you can detect a key press when the callback's `uicontrol` object has focus. If no `uicontrol` has focus, the figure's key press callback function, if any, is invoked. This property is available in the `uicontrol` function and in `GUIDE`.

If you specify `KeyPressFcn` as an M-file, the callback routine can query the figure's `CurrentCharacter` property to determine what particular key was pressed and thereby limit the callback execution to specific keys. If you specify `KeyPressFcn` as a function handle, the callback routine can retrieve information about the key that was pressed from its event data structure argument.

As an example, you can use this property to enable a user to press **Enter**, rather than the space bar, after giving focus to a `uicontrol` push button. Use the push button's key press callback function to determine if the user pressed the **Enter** key. If it was the **Enter** key, call the push button callback.

See the `Uicontrol` Properties for more information.

Edit Text Box Scroll Bar

For `uicontrol` editable text fields, i.e. the `Style` property is set to `'edit'`, if `Max-Min>1`, then multiple lines are allowed. For multi-line edit boxes, a vertical scroll bar enables you to scroll the text. You can also use the arrow keys to scroll.

Setting Uicontrol Focus

The `uicontrol` function now enables you to transfer focus programmatically to a specified `uicontrol` object. The syntax `uicontrol(uich)` transfers focus to the `uicontrol` object with handle `uich`.

Multiple Selection in uigetfile

The `uigetfile` function can now create a dialog that enables the user to select and retrieve multiple files using the **Shift** and **Ctrl** keys. You can turn this capability on or off using the new `'MultiSelect'` parameter. The default setting is off.

Program Suspension Time-Out

A new `uiwait` argument, `timeout`, enables you to specify the number of seconds after which program execution will resume, unless `uiresume` is called first or the specified figure is deleted. For example,

```
uiwait(h,5)
```

causes the suspended program to resume execution, if it has not already, after five seconds.

Standard Dialog Box Push Buttons

For standard dialog boxes with more than one `uicontrol` push button, you can now give focus to another button while retaining the default button. Focus is denoted by a border or a dotted border, respectively, in UNIX and Microsoft Windows. The default button has a shadow.

In such a case, if the user presses the space bar, the button with focus gets the key press and can choose to execute its own callback or the callback of the default button. If the user presses **Enter**, the default push button gets the key press and its callback executes. This code provides an example.

```
ButtonName=questdlg('What is your wish?', ...  
                    'Genie Question', ...  
                    'Food','Clothing','Money','Money')
```


Platform Limitations

The MATLAB functionality described in these Release Notes and in the MATLAB documentation applies to MATLAB 7.0, with the exception of any limitations listed in this section.

This discussion of new MATLAB platform limitations is organized into the following categories:

- Graphics Platform Limitations (p. 2-2)

Graphics Platform Limitations

The MATLAB 7.0 graphics features have the platform limitations described in this section.

Cannot Dock Figures on Macintosh

You cannot dock figures in the Desktop, because MATLAB uses native figure windows on the Macintosh platform.

Plotting Tools Not Working on Macintosh

The plotting tools are not supported on the Macintosh platform. This means the Figure Palette, Plot Browser, and Property Editor do not work these platforms. To use the MATLAB 6 Property Editor, see the `propedit` command.

Not All Macintosh System Fonts Are Available

MATLAB figures do not support the same fonts as native Macintosh applications. Use the `uifont` functions to see which fonts are available in MATLAB.

Upgrading from an Earlier Release

This section describes the upgrade issues involved in moving from MATLAB Version 6.5 to Version 7.0. This discussion of new MATLAB upgrade issues is organized into the following categories:

- Desktop Tools and Development Environment Upgrade Issues (p. 3-2)
- Mathematics Upgrade Issues (p. 3-5)
- Programming Upgrade Issues (p. 3-12)
- Graphics Upgrade Issues (p. 3-27)
- External Interface/API Upgrade Issues (p. 3-28)
- Creating Graphical User Interface (GUIDE) Upgrade Issues (p. 3-30)

If you are upgrading from a release earlier than Release 12.1, then you should see “Upgrading from an Earlier Release” on page 6-22 in the MATLAB 6.5.1 Release Notes.

Desktop Tools and Development Environment Upgrade Issues

The issues involved in upgrading from MATLAB 6.5 to MATLAB 7.0, in terms of desktop tools and development environment features, are discussed below.

Desktop and General Changes

- The `terminal` function was removed.
- The data returned by the `license` command is now sorted in alphabetical order and uses only lowercase characters.

Command Window

- Parentheses matching is not supported.
- There is no longer a preference allowing you to limit the number of tab completions that display. MATLAB always displays all possible completions.

Help Browser

- The `web` function no longer opens the specified URL in the Help browser by default, but instead opens the page in the MATLAB Web browser. Use the `-helpbrowser` option to open the page in the Help browser.
- Favorites are not migrated from the previous version.
- If you start MATLAB using the `-nojvm` option, you cannot view the HTML documentation files from within MATLAB. The `docopt` function no longer supports that option. You can view the HTML documentation files at the MathWorks Web site.

File Operations, Workspace, and Path

- MATLAB no longer considers built-in functions differently from any other M-files on the search path. MATLAB now looks for a given name first as a variable, then as an M-file in the current directory, and finally as an M-file on the search path. Previously MATLAB looked for a given name as a built-in function after looking for it as a variable.

If you have a function name that is the same as a MATLAB built-in function, your function might run instead of the built-in function, whereas in previous releases the built-in function would have run. For the built-in function to run, remove or rename your function, or change the directory order in the search path.

- The `path2rc` function has been replaced by a new function, `savepath`. If you use `path2rc`, it will run `savepath` instead. The new function, `savepath`, performs the same actions as `path2rc` did, but uses a more intuitive name. In addition, `savepath` is case-sensitive on PC platforms, whereas `path2rc` was not. Use `savepath` instead of `path2rc`, and replace existing instances of `path2rc` with `savepath`.

Editing and Debugging

- Because of the new block comment symbols, if you have any files with lines that consist only of `%{` and `%}`, they might be misinterpreted as block comment start and end symbols, and might cause errors in your file.
- Because of the new symbols for cell publishing, if you have any files with lines that consist only of `%%`, those lines might be misinterpreted as the start of a cell. Your files will still run without problems, but if you publish the M-files, you might need to modify those lines.
- The `dbstack` function was updated to support nested functions. If you use `dbstack` in M-files, you might need to update your files because of this change. When you run `dbstack` and return results to a structure, there are now three fields, whereas in previous versions, there were only two fields. The fields are:
 - `file`, the file in which the function appears
 - `name`, the function name within the file
 - `line`, the line number in the functionThe `file` field does not contain a complete pathname, as the `name` field did in previous versions. To get the complete pathname, use `dbstack(' -completenames')`.
- The `dbstatus` function was updated to support conditional breakpoints. As a result there have been changes to some of the fields in the structure returned with `s = dbstatus(...)`. If you use `dbstatus` in M-files, you might need to update your files because of this change. For details on the new format, see the `dbstatus` reference page.

Source Control

In MATLAB 6.5 (R13) and MATLAB 7.0 (R14), only source control systems that comply with the Microsoft Common Source Control standard are supported. If there is a compliant source control system installed on your machine, it will be listed in the Source Control options in the MATLAB Preferences dialog.

There are several vendors who provide and interface into Revision Control Systems (RCS), Concurrent Versions System (CVS), and other such tools using Microsoft Source Code Control API. ComponentSoftware provides one such interface layer.

Mathematics Upgrade Issues

The issues involved in upgrading from MATLAB 6.5 to MATLAB 7.0, in terms of mathematics features, are discussed in the following sections:

- “Integer Data Type Functions Now Round Instead of Truncate” on page -5
- “max and min Now Have Restrictions on Inputs of Different Data Types” on page -6
- “Changes to Behavior of Concatenation” on page -7
- “Changes to the Behavior of Sum” on page -8
- “FFT Functions Applied to Integer Data Types are Becoming Obsolete” on page -9
- “Matrix, Trigonometric, and Other Math Functions No Longer Accept Inputs of Type char” on page -9
- “New Names for Demos expm1, expm2, and expm3” on page -9
- “Matrix, Trigonometric, and Other Math Functions No Longer Accept Inputs of Type char” on page -9
- “Colon Operator on char Now Returns a char” on page -10
- “Obsolete Functions” on page -10

Integer Data Type Functions Now Round Instead of Truncate

The following integer data functions now round noninteger inputs instead of truncating:

- int8
- uint8
- int16
- uint16
- in32
- uint32
- int64
- uint64

For example,

```
int8(3.7)
```

returns

```
ans =
```

```
4
```

in MATLAB 7.0. In previous releases, the same command returned 3. If you have code that contains these functions, it might return different results in Version 7.0 than in previous releases, in particular, results that differ by 1 after converting floating-point inputs to an integer data type.

You can turn the following warning on to help diagnose these differences:

```
warning on MATLAB:intCovertNonIntVal
```

See “New Warnings for Integer Arithmetic” on page -27 for more information about this and other new warning messages.

max and min Now Have Restrictions on Inputs of Different Data Types

In MATLAB 7.0, the functions `max` and `min` now have the following restrictions on inputs of different data types:

- If any input has an integer data type, all other inputs must have the same integer data type or type `scalar double`.
- If any input is of type `single`, all other inputs must have type `double` or `single`.

Other combinations of inputs now return an error message. In previous releases, inputs to `max` or `min` could have any combination of data types.

For the allowed mixed-type combinations listed above, `max` and `min` now return results of a different data type than in previous releases.

- If one input has an integer data type, while another has type `double`, the result now has the same integer data type. In previous releases, the result had type `double`.
- If one input has type `single`, while another has type `double`, the result now has type `single`. In previous releases, the result had type `double`.

You can turn on the following warning messages to diagnose any issues that might result from this change in behavior:

- warning on `MATLAB:max:mixedIntegersScalarDoubleInputs`
- warning on `MATLAB:max:mixedSingleDoubleInputs`
- warning on `MATLAB:min:mixedIntegersScalarDoubleInputs`
- warning on `MATLAB:min:mixedSingleDoubleInputs`

Changes to Behavior of Concatenation

When you perform concatenation (`[a, b]`, `[a;b]`, and `cat(a,b,dim)`) on mixed integer and other numeric or logical inputs, the left-most integer type among the inputs is the type of the result. As a result, the other inputs might lose values when they are converted to the integer data type. In Version 7.0, MATLAB now returns a warning when you concatenate these mixed data types.

For example,

```
[int8(100) uint8(200)]
Warning: Concatenation with dominant (left-most) integer class
may overflow other operands on conversion to return class.
(Type "warning off MATLAB:concatenation:integerInteraction" to
suppress this warning.)
```

```
ans =

    100    127

class(ans)
```

```
ans =

    int8
```

Concatenating an input of any nondouble numeric data type (single and integer data type) with type char now returns a result of type char. In previous releases, the same operation returned a result of the same type as the numeric data type.

Changes to the Behavior of Sum

In Version 7.0, sum applied to a vector of type single performs single accumulation and returns a result of type single. In previous releases, sum performed this operation in double accumulation. To restore the previous behavior, call sum with the syntax

```
sum(X, 'double')
```

or

```
sum(X, dim, 'double')
```

See “New Class Inputs for sum” on page -30 for more information on this new syntax.

FFT Functions Applied to Integer Data Types are Becoming Obsolete

In previous releases, the following fast Fourier transform (FFT) and related functions cast integer inputs of type `uint8` and `uint16` to `double`, used the `double` algorithm, and returned a `double` result:

- `fft`
- `fftn`
- `ifft`
- `ifftn`
- `conv2`

In Version 7.0, these operations return warning messages that recommend convert the inputs to `double` before applying the function, for example, by `fft(double(x))`.

New Warnings for Complex Inputs to `atan2`, `log2`, and `pow2`

The following functions now return a warning for inputs that are not real numbers:

- `atan2(y,x)`
- `[f,e] = log2(x)`
- `pow2(f,e)`

New Names for Demos `expm1`, `expm2`, and `expm3`

The demos `expm1`, `expm2`, and `expm3` have been renamed `expmdemo1`, `expmdemo2`, and `expmdemo3`, to avoid a name conflict with the new function `expm1`.

Matrix, Trigonometric, and Other Math Functions No Longer Accept Inputs of Type `char`

Matrix functions, such as `chol`, `lu`, and `svd`, and trigonometric functions, such as `sin` and `cos`, no longer accept inputs of type `char`. In previous releases, these functions simply converted `char` inputs to `double` before performing operations on them. To restore the previous behavior of these functions, create an M-file

that converts its input to double before applying the function. For example, to restore the behavior of `sin`,

- 1 Create a directory called `@char` in a directory on the MATLAB path, for example, your work directory.

- 2 Create an M-file with the following commands:

```
function s = sin(x)
s = sin(double(x));
```

- 3 Save the file as `sin.m` in the directory `@char`.

Colon Operator on char Now Returns a char

Applying the colon operator to inputs of type `char` now returns a result of type `char`. For example,

```
'a':'g'
```

```
ans =
```

```
abcdefg
```

In previous releases, the same operation returned a result of type `double`.

Obsolete Functions

The functions listed in the left column of the following table are obsolete and will be removed from a future version of MATLAB. Use the replacement functions listed in the right column instead.

| Obsolete Function | Replacement Function |
|---------------------|----------------------|
| <code>colmmd</code> | <code>colamd</code> |
| <code>quad8</code> | <code>quadl</code> |
| <code>symmmd</code> | <code>symamd</code> |

The following obsolete functions are no longer included in MATLAB:

- `fmin`
- `fmins`
- `icubic`
- `interp4`
- `interp5`
- `interp6`
- `meshdom`
- `nls`
- `saxis`

Programming Upgrade Issues

The issues involved in upgrading from MATLAB 6.5 to MATLAB 7.0, in terms of programming features, are discussed below.

Changes You Should Note

- “Making Release 14 MAT-files Readable in Earlier Versions” on page 3-13
- “MAT-Files Generated By Release 14 Beta2 Must Be Reformatted” on page 3-13
- “Reserved Bytes in MAT-File Header” on page 3-14
- “Case-Sensitivity in Function and Directory Names” on page 3-15
- “Differences Between Built-Ins and M-Functions Removed” on page 3-19

Other Programming Issues

- “Function Handles and Backward Compatibility” on page 3-20
- “Changes to Error Message Format” on page 3-21
- “Regular Expression Functions No Longer Support Character Matrices” on page 3-21
- “bin2dec Ignores Space Characters” on page 3-22
- “isglobal Function To Be Discontinued” on page 3-22
- “getfield and setfield Not To Be Deprecated” on page 3-22
- “Warning on Concatenating Different Integer Classes” on page 3-22
- “Mathematic Operations on Logical Values” on page 3-23
- “Reading Date Values with xlsread” on page 3-23
- “64-Bit File Handling on MacIntosh” on page 3-24
- “Importing Dates from Excel Worksheets” on page 3-24
- “Change in Output from xlsfinfo” on page 3-24
- “Change to How evalin Evaluates Dispatch Context” on page 3-24
- “Warning on Naming Conflict” on page 3-25
- “Enabling and Disabling Warning Messages” on page 3-26

Making Release 14 MAT-files Readable in Earlier Versions

Caution Release 14 MATLAB writes character and figure data to MAT-files using Unicode encoding by default. Unicode encoded MAT-files are not readable by earlier versions of MATLAB. Thus, if you save data to a MAT-file using MATLAB Release 14, and you intend to load this MAT-file into an earlier release of MATLAB, you must override the Unicode default during the save, as explained in this section.

In Release 14, MATLAB uses Unicode character data encoding in `mxArrays` and `mxArray` storage in MAT-files. This is now the default encoding used by MATLAB when writing to MAT-files with the `save` and `hgsave` functions or with the MAT-file external interface functions.

You can override the default encoding by using the `-v6` switch with `save` and `hgsave`:

```
save filename -v6
hgsave filename -v6
```

or, when saving with MAT functions, by setting the mode to "wL" on the `matOpen` operation:

```
matOpen(filename, "wL");
```

MAT-Files Generated By Release 14 Beta2 Must Be Reformatted

Any MAT-files that you created with Release 14 Beta 2 were written using an internal format that is no longer supported by MATLAB. As a result, if you need to import data from these files using any release besides Release 14 Beta 2, you must first regenerate the files as described in this section. You cannot read these files using other releases of MATLAB 7.0, and attempting to read them with MATLAB 6.5 or 6.5.1 will corrupt memory.

There are two ways in which you can regenerate your MAT-file:

- If you want to use the MAT-file with earlier versions of MATLAB, regenerate the file using the local character set for your system. To do this, run MATLAB R14 prerelease or MATLAB R14 Beta 2, load the MAT-file, and rewrite the file using the command

```
save filename -v6
```

- If you want to use the MAT-file with R14 LCS or later, regenerate the file using Unicode encoding. To do this, run MATLAB R14 prerelease, load the MAT-file, and rewrite it using the following command that uses the `-unicode` default.

```
save filename
```

Caution The final R14 release of MATLAB does not allow you to import a MAT-file written with Release 14 Beta 2. You will get an error if you attempt to do this. To use a Beta 2 MAT-file with Release 14, you must first reformat the file with MATLAB R14 prerelease as described above.

If you no longer have access to Release 14 Beta2 or the R14 prerelease, then you must regenerate the data and save it again.

Reserved Bytes in MAT-File Header

In previous releases of MATLAB, the last 4 bytes of the 128-byte MAT-file header were reserved for use by the MathWorks. In Release 14, the last 12 bytes of this header are reserved. See the PDF file “MAT-File Format” for more information.

New Features for Nondouble Data Types

The section “New Nondouble Mathematics Features” on page 1-24 describes new features affecting the nondouble (`single` and `integer`) data types. These changes affect `single` and `integer` arithmetic operations, and also conversion of `single` and `double` data types to integers.

Case-Sensitivity in Function and Directory Names

Prior to this release, filenames for MATLAB functions and Simulink models (M, P, MEX, DLL, and MDL files), and also directory names were interpreted somewhat differently by MATLAB with regards to case sensitivity, depending upon which platform you were running on. Specifically, earlier versions of MATLAB handled these names with case sensitivity on UNIX, but without case sensitivity on Windows.

This release addresses the issue of case sensitivity in an effort to make MATLAB consistent across all supported platforms. By removing these differences, we hope to make it easier for MATLAB users to write platform independent code.

This change is described under the following topics:

- “Case Sensitivity in MATLAB 6 and Earlier” on page 3-15
- “Case Sensitivity in MATLAB 7” on page 3-16
- “Comparing Case Sensitivity in MATLAB 6 and MATLAB 7” on page 3-16
- “Turning Off Warnings Caused by Case Mismatch” on page 3-19

Case Sensitivity in MATLAB 6 and Earlier

There are several rules regarding case sensitivity that were already consistent across all platforms in MATLAB 6, and remain in effect on all platforms in MATLAB 7. MATLAB interprets each of the following with case sensitivity on both Windows and UNIX:

- Function names that correspond to MATLAB built-ins
- M-file subfunction names
- The names of functions imported from another language environment, such as Java or COM

UNIX. On all UNIX platforms, including the new implementation on MacIntosh, all function, model, and directory names were case sensitive and required an exact match. This rule remains true for UNIX systems in MATLAB 7.

Windows. On Windows platforms, MATLAB 6 obeys the following rules. These rules are changing in MATLAB 7:

- Function and model names were not case sensitive.
- Directory names, including MATLAB class directory names (e.g., @MyClass) and private directory names (e.g., prIVAtE) were not case sensitive.

Case Sensitivity in MATLAB 7

MATLAB 7 removes the platform specific behaviors by adopting its UNIX case sensitivity rules on Windows systems. MATLAB running on Windows now gives preference to an exact (case sensitive) name match, but falls back to an inexact (case insensitive) match when no exact match can be found.

New Warnings Related to Case Sensitivity. Whenever MATLAB 7 detects a potential naming conflict related to case sensitivity, it issues a warning. If you get one of these warnings when running a MATLAB program, you may want to modify the related code to eliminate the warning, or you may wish to simply disable the warning.

Comparing Case Sensitivity in MATLAB 6 and MATLAB 7

There are four main conditions under which MATLAB 7 interprets directory or function names differently in regards to case sensitivity:

- “Two Files of the Same Name” on page 3-16
- “Two Method Files of the Same Name” on page 3-17
- “One File with an Inexact Match” on page 3-17
- “Private Directory Names” on page 3-18

Two Files of the Same Name. Consider the situation in which there are two or more directories on the MATLAB path that contain a function or model file of the same name. The names of these M-files differ only in letter case:

```
H:\released\myTestFun.m
K:\under_test\mytestfun.m
```

Of these two directories, H:\released is closer to the beginning of the MATLAB path and thus has priority over the other:

```
path = H:\released; K:\under_test; ...
```

On Windows Platforms —

- In MATLAB 6, executing the function `mytestfun` invokes
`H:\released\myTestFun.m`.
- In MATLAB 7, executing the function `mytestfun` invokes
`K:\under_test\mytestfun.m` and also displays the following warning:

```
Function call mytestfun invokes K:\under_test\mytestfun.m
however, function H:\released\myTestFun.m, that differs only in
case, precedes it on the path.
```

On UNIX Platforms —

MATLAB 7 does the same as on Windows, except that the warning message is disabled by default.

Two Method Files of the Same Name. In this case, there are two M-files of the same name that implement methods of a MATLAB base class and one of its subclasses:

```
@baseclass/my_method.m
@subclass/My_Method.m
```

On Windows Platforms —

- In MATLAB 6, the command `my_method(subclass)` invokes
`@subclass/My_Method`.
- In MATLAB 7, the same command invokes `@baseclass/my_method` because
it is an exact match.

On UNIX Platforms —

MATLAB 7 does the same as on Windows.

One File with an Inexact Match. Another situation that MATLAB now handles differently involves just one function or model file that matches the function being called:

```
H:\released\myTestFun.m
```

However, the name of this M-file does not match the called function (`mytestfun`) in letter case.

On Windows Platforms —

- In MATLAB 6, calling the function `mytestfun` invokes
`H:\released\myTestFun.m`.
- In MATLAB 7, calling the function `mytestfun` invokes the same M-file but also displays the following warning:

```
Function call mytestfun invokes inexact match  
H:\released\myTestFun.m.
```

On UNIX Platforms —

- In MATLAB 6, calling the function `mytestfun` results in an error.
- In MATLAB 7, calling `mytestfun` invokes `H:\released/myTestFun.m` and generates the following warning:

```
Function call mytestfun invokes inexact match  
H:/released/myTestFun.m.
```

Private Directory Names. Private functions must reside in a directory named `private` that is one level down from the directory of any calling function. As of this release, the directory name `private` is case sensitive on Windows as it has always been on UNIX.

On Windows Platforms —

- In MATLAB 6, calling function `myprivfun` in an environment where only a subdirectory named `\PriVate` contains the M-file `myprivfun.m` invokes `\PriVate\myprivfun` without displaying a warning.
- MATLAB 7 does the same as MATLAB 6, except that it also displays the following warning:

```
Wrong case spelling of 'private' as a directory name in  
\released\PriVate\myprivfun.m.
```

On UNIX Platforms —

- In MATLAB 6, calling function `myprivfun` in an environment where only a subdirectory named `\Private` contains the M-file `myprivfun.m` results in an error.
- In MATLAB 7, calling `myprivfun` in this same environment invokes `/Private/myprivfun` and also displays the following warning:

```
Wrong case spelling of 'private' as a directory name in
/released/Private/myprivfun.m.
```

Turning Off Warnings Caused by Case Mismatch

You can disable warnings caused by case mismatch with the following command:

```
warning off MATLAB:dispatcher:InexactMatch
```

To disable this warning for all of your MATLAB sessions, add this command to your `startup.m` or `matlabrc.m` file.

Differences Between Built-Ins and M-Functions Removed

MATLAB implements many of its core functions as built-ins. In previous releases of MATLAB, there have been several significant differences between the way MATLAB handles built-in and M-file functions. As of this release, MATLAB handles both types of functions the same. This change affects the following:

- “Function Dispatching” on page 3-19
- “Return Value from the functions Function” on page 3-20
- “Output from the which Function” on page 3-20

Function Dispatching

MATLAB now dispatches both built-in and M-file functions according to the same precedence rules, (see “Function Precedence Order” in the *Programming and Data Types* section of the MATLAB documentation). In previous releases, subfunctions, private functions, and class constructor functions took precedence over M-functions of the same name, but not over built-ins. In this

release, built-in functions follow the same rules given to M-functions, and thus are lower in precedence than the three function types named above.

This change addresses a potential problem in that changes to the internal implementation of MATLAB functions could potentially affect the operation of your own M-code. For example, if a new version of MATLAB were to change an internal function from being M-based to being built-in, the function in the new version would now be subject to different precedence rules. If one of your M-code modules had a subfunction with the same name as this function (now obeying the built-in rules), then this subfunction would never be called.

This release resolves this potential conflict by using the same precedence rules for both M-functions and built-ins.

Return Value from the functions Function

The MATLAB `functions` function returns information about a function handle such as the function name, type, and filename. In previous releases, functions returned the filename for a built-in function as the string

```
'MATLAB built-in function'
```

In this release, MATLAB associates each built-in function with a placeholder file that has a `.bi` extension (for example, `reshape.bi` for the built-in `reshape` function).

Output from the which Function

The `which` function now displays the pathname for built-in functions, as well as for overloaded functions when only the overloaded functions are available.

Function Handles and Backward Compatibility

In Releases 12 and 13, you could form an array of function handles using the array constructor operator `[]`, and refer to handles within this array using the array indexing operator `()`. To call the function referred to by a function handle value, you needed to use the `feval` function.

In Release 14, you invoke a function handle in the same way that you would call a function by name. For example, if the handle to a function was stored in variable `h`, you would call the function as if the handle `h` were a function name:

```
h(arg1, arg2, ...)
```

Using `feval` for this purpose is no longer necessary and is, in fact, slower.

This change is not backward compatible. This release, however, has a transition strategy that will leave almost all Release 12 & Release 13 programs working:

- Constructing a non-scalar array of function handles is only a warning, not an error.
- Parenthesis notation on a non-scalar function handle means subscripting, just as in Release 13, while the same notation on scalar function handles means function call, as described above.

Incompatibility can arise only if you construct a scalar array of function handles and actually index it, necessarily with an index of 1.

Changes to Error Message Format

The last two lines of MATLAB error messages have changed for Release 14. Two advantages of this change are that error messages are now displayed in a consistent format, and part of the text of the message can be used directly with certain MATLAB commands, like `dbstop`.

An example of the new format is

```
??? Error using ==> strcmp
Too many input arguments.
```

```
Error in ==> errmsgtest at 11
strcmp('aa','bb','cc');
```

See “Changes to Error Message Format” on page 1-51 under “Programming Features” for more information on features of the new error message format.

Regular Expression Functions No Longer Support Character Matrices

You can now pass a vector of strings in a cell array to any of the MATLAB regular expression functions (`regexp`, `regexpi`, and `regexprep`). Because this is the preferred method of passing a string vector, MATLAB no longer supports using character matrices for this purpose.

bin2dec Ignores Space Characters

The `bin2dec` function now ignores any space (' ') characters in the input string. Thus, the binary string '010 111' now yields the same result as the string '010111'.

In Release 13, `bin2dec` interpreted space characters as zeros:

```
bin2dec('010 111')
ans =
    39
```

In this release, `bin2dec` ignores all space characters:

```
bin2dec('010 111')
ans =
    23
```

isglobal Function To Be Discontinued

Support for the `isglobal` function will be removed in a future release of MATLAB. In Release 14, invoking `isglobal` generates the following warning:

```
Warning: isglobal is obsolete and will be discontinued.
Type "help isglobal" for more details.
```

getfield and setfield Not To Be Deprecated

There are no plans to remove the `getfield` and `setfield` functions from the MATLAB language, as stated in the release notes for MATLAB Release 13.

Warning on Concatenating Different Integer Classes

If you concatenate integer arrays of different integer classes, MATLAB displays the warning

```
Concatenation with dominant (left-most) integer class may
overflow other operands on conversion to return class.
```


The class of the resulting array is the same as the dominant (or left-most) value in the concatenation:

```
a = int8([52 37 89; 23 16 47]);
b = int16([74 61 32; 98 73 25]);

% Combine int8 and int16 (int8 is dominant)
c = [a b];
class(c)
ans =
    int8

% Combine int16 and int8 (int16 is dominant)
c = [b a];
class(c)
ans =
    int16
```

Mathematic Operations on Logical Values

Most mathematic operations are not supported on logical values.

Reading Date Values with xlsread

When reading date fields from a Microsoft Excel file using earlier versions of MATLAB, it was necessary to convert the Excel date values into MATLAB date values. This was necessary because Excel and MATLAB calculated date values based on a different reference date. This is explained in the section, “Handling Excel Date Values” on the function reference for `xlsread`.

With MATLAB 7.0, you no longer have to do this conversion because `xlsread` now imports dates as strings rather than as numerical values. If your existing code converts Excel date values to MATLAB values, you will need to remove this step so that you end up with the correct results.

64-Bit File Handling on Macintosh

The release notes for MATLAB Release 13 should have included Macintosh in the list of those platforms that support 64-bit file handling. This support is available on the following platforms:

- Windows
- Solaris
- Linux 2.4.x
- HP-UX 11.0, 9000/785
- Macintosh

Importing Dates from Excel Worksheets

Prior to this release, `xlsread` imported date information from an Excel file and returned the results as a double. In Release 14, `xlsread` returns this information as a cell array containing data of class `char`. The reason for this is that MATLAB now imports Excel files using an Excel COM server. Excel returns dates as strings, and there is really no indication that what is returned is a date.

Change in Output from `xlsinfo`

`xlsinfo` now returns the names of all worksheets in an Excel file instead of just the ones with numbers in them (as in Release 13).

Change to How `evalin` Evaluates Dispatch Context

In Release 13 and earlier, the `evalin` function evaluated its input in the specified workspace, but not the workspace's corresponding dispatching context. Hence, running the following example used to succeed, calling the subfunction `MySubfun` but using the value of `x` from the base workspace:

```
function demo
    evalin('base', 'MySubfun(x)')

function MySubfun(in)
    disp(in)
```

When you call `evalin` in Release 14, MATLAB tries to find a function named `MyLocalFunction` that is accessible in the base workspace, i.e. at the command

prompt. Since `MySubfun` is a subfunction and therefore not in scope at the command prompt, MATLAB errors, reporting that `MySubfun` is undefined.

There are two ways to change your existing code to work with this new behavior. First, if your code only needs to get the value of the subfunction's inputs from the base workspace (as `demo.m` does above), and does not care what context `MySubfun` is run in, then you can change your code to use `evalin` only to get the values of the inputs from the base workspace, like this:

```
function demo_workaround1
MySubfun(evalin('base', 'x'))

function MySubfun(in)
disp(in)
```

If, however, it is important that the subfunction itself be run in the context of the base workspace, you can place a function handle to the subfunction in the base workspace and then evaluate that:

```
function demo_workaround2
assignin('base', 'MySubfunHandle', @MySubfun);
evalin('base', 'MySubfunHandle(x)')

function MySubfun(in)
disp(in)
```

You can also substitute `'caller'` for `'base'` in the workaround code if your original code uses `evalin('caller', ...)`.

Warning on Naming Conflict

The following warning was added to identify the case when you first use a name as a function and later use it as a variable:

```
Warning: File: D:\Work\MATLAB_XL\theworks\my_yprime.m Line: 17
Column: 1 Variable 'getdata' has been previously used as a
function name.
(Type "warning off MATLAB:mir_warning_variable_used_as_function:
to suppress this warning.)
```

For example, this code generates such a warning:

```
X = i; % Calls the function i() to get sqrt(-1)
for i = 1:10 % uses i as a variable. This produces the warning.
... end
```

Enabling and Disabling Warning Messages

The following message, which MATLAB appended to all warning messages in the previous release, is no longer displayed.

(Type "warning off <msgid:msgstr>" to suppress this warning.)

Use the off and on options of the warning function to control the display of all or selected warnings. This example disables a selected warning, and then enables all warnings:

```
% All warnings are enabled by default.
A = 5/0;
Warning: Divide by zero.

% Disable the most recent warning
[msgstr msgid] = lastwarn;
warning('off', msgid);

% Try it again. This time there is no warning.
A = 5/0;

% Enable all warnings
warning('on', 'all')

% Verify that the warning is reenabled.
A = 5/0;
Warning: Divide by zero.
```

Graphics Upgrade Issues

The issues involved in upgrading from MATLAB 6.5 to MATLAB 7.0, in terms of graphics features, are discussed in this section.

Plotting Tools Not Working on Macintosh

The plotting tools not supported on the Macintosh platform. This means the Figure Palette, Plot Browser, and Property Editor do not work these platforms. To use the MATLAB 6 Property Editor, see the `propedit` command.

Backward Compatible Fig-Files

To save figures in a Fig-File that is compatible with versions of MATLAB before Version 7, follow these two steps.

- Ensure that any plotting functions used to create the contents of the figure are called with the 'v6' argument, where applicable.
- Use the '-v6' option with the `hgsave` command.

See Plot Objects and Backward Compatibility for more information.

Figure Window Menu Changes

The **Window** menu on MATLAB figures no longer lists open Simulink models. However, the Desktop **Window** menu continues to show open Simulink models.

External Interface/API Upgrade Issues

The issues involved in upgrading from MATLAB 6.5 to MATLAB 7.0, in terms of external interface/API features, are discussed in the Programming and Data Types Upgrade Issues, and below:

- “Changes to MAT-Files” on page 3-28
- “Optional Input Arguments to COM Methods” on page 3-28
- “Display of Interface Handles” on page 3-29
- “Identifying Dependencies When MEX-Files Don’t Load” on page 3-29
- “Recompile MEX-Files on GLNX86 and Macintosh” on page 3-29
- “Shared Libraries Now In /bin/\$ARCH” on page 3-29

Changes to MAT-Files

Changes to MAT-files in MATLAB 7.0 are described under “Programming Features” on page 1-42:

- “Making Release 14 MAT-files Readable in Earlier Versions” on page 3-13
- “MAT-Files Generated By Release 14 Beta2 Must Be Reformatted” on page 3-13
- “Reserved Bytes in MAT-File Header” on page 3-14

Caution MAT-files saved in MATLAB version 7.0 without using the new `-nunicode` flag will not be readable in previous versions of MATLAB. See “Making Release 14 MAT-files Readable in Earlier Versions” on page 3-13.

Optional Input Arguments to COM Methods

When calling a method that takes optional input arguments, you can skip any optional argument by specifying an empty array (`[]`) in its place. The syntax for `invoke` with the second argument (`arg2`) not specified is as follows:

```
invoke(handle, 'methodname', arg1, [], arg3);
```

See the section, “Optional Input Arguments” in the External Interfaces documentation for more information.

Display of Interface Handles

MATLAB has changed the way it displays a COM interface in this release. For example, the string used to represent an interface in MATLAB 6.5 was

```
[1x1 Interface.excel.application.Workbooks]
```

MATLAB 7.0 represents this same interface with the following string:

```
[1x1 Interface.Microsoft_Excel_9.0_Object_Library.Workbooks]
```

Identifying Dependencies When MEX-Files Don't Load

If MEX-files don't load on the PC the error message is not informative. The dependency is a very useful tool distributed with MSVC. It is also freely available from www.dependencywalker.com. In R15, we will incorporate some kind of dependency walker into our MEX loader but for now, we will point users to the web site.

Recompile MEX-Files on GLNX86 and Macintosh

On GLNX86 and Macintosh systems, all MEX-files that can throw errors need to be recompiled for R14.

In Release 14, MATLAB uses C++ exception handling. MEX-files built prior to R14 did not support C++ exceptions.

For example, write a C MEX-file that just calls `mexErrMsgTxt`. If you build this with a release prior to Release 14 and run it, the program aborts MATLAB. If you build this with Release 14 and run it, MATLAB will handle the exception correctly.

Shared Libraries Now In /bin/\$ARCH

Shared libraries previously residing in directory `$MATLAB/extern/lib/$ARCH` are now in `$MATLAB/bin/$ARCH`.

Creating Graphical User Interface (GUIDE) Upgrade Issues

The issues involved in upgrading from MATLAB 6.5 to MATLAB 7.0, in terms of graphical user interface features, are discussed below.

New Syntax for `uigetfile` and `uiputfile`

The `uigetfile` and `uiputfile` syntax that enables you to position dialog boxes that are used to retrieve and save files is changed. The new syntaxes are `uigetfile('FilterSpec','DialogTitle','Location',[x y])` and `uiputfile('FilterSpec','DialogTitle','Location',[x y])`.

The old syntaxes, `uigetfile('FilterSpec','DialogTitle',x,y)` and `uiputfile('FilterSpec','DialogTitle',x,y)`, are obsolete and will be removed in a later release.

Frames Not Available in GUIDE Layout Editor

The frame component no longer appears in the GUIDE Layout Editor component palette. It has been replaced by the panel and button group components. See “New Container Components” on page 1-85 for information about these new components.

GUIDE continues to support frames in those GUIs that contain them, but it is recommended that you replace them with panels or button groups.

Exporting a GUI Containing a Panel or Button Group

You can export a GUI that contains a panel or button group from GUIDE to a single M-file that does not require a FIG-file. However, you will not be able to run that M-file in MATLAB versions earlier than 7.0.

Major Bug Fixes

MATLAB 7.0 includes several bug fixes made since the last MATLAB release. This section describes the particularly important Version 7.0 bug fixes.

If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

If you are upgrading from a release earlier than MATLAB 6.5.1 (Release 13 SP1), then you should also see “Major Bug Fixes” on page 6-11 in the MATLAB 6.5.1 Release Notes.

Known Software and Documentation Problems

This section includes a link to a description of known software and documentation problems in MATLAB 7.0.

If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

For a list of bugs reported in MATLAB 6.5.1 that remain open, see “Known Software and Documentation Problems” on page 6-23 in the MATLAB 6.5.1 Release Notes.

MATLAB 6.5.1 Release Notes

| | |
|---|------|
| New Features | 1-2 |
| MATLAB Interface to Generic DLLs | 1-2 |
| Relational Operators Work with int64, uint64 | 1-3 |
| Reading HDF5 Files | 1-3 |
| Reading and Writing Data with JPEG Lossless Compression | 1-9 |
| Reading and Writing L*a*b* Color Data | 1-10 |
| | |
| Major Bug Fixes | 1-11 |
| Seeking Within a File | 1-11 |
| Reshaping to More Than Two Dimensions | 1-12 |
| mkdir No Longer Fails On Windows NT | 1-12 |
| Using sqrt with Complex Input | 1-12 |
| Multiplying Matrices with Non-Double Entries | 1-12 |
| Sorting a Sparse Row Vector or Matrix | 1-13 |
| diff Produces Correct Results With Logical Inputs | 1-13 |
| Opening Modal Dialog with Third-Party GUI Open | 1-13 |
| Serial Port Object with Latest Windows Service Pack | 1-13 |
| OpenGL Problem Using Notebook | 1-13 |
| Lcc C Compiler Fixed to Handle Large C Files | 1-14 |
| Bug Fixes in MATLAB Interface to COM | 1-14 |
| Bug Fixes in Creating GUIs | 1-19 |
| | |
| Upgrading from an Earlier Release | 1-22 |
| Rebuild Macintosh MEX-files | 1-22 |
| Function and Data Type Names in Generic DLL Interface | 1-22 |
| | |
| Known Software and Documentation Problems | 1-23 |
| Using xlsinfo on Systems Without Excel | 1-23 |

New Features

This section introduces the following new features and enhancements added in MATLAB 6.5.1 since Version 6.5 (Release 13):

- “MATLAB Interface to Generic DLLs” on page 6-2
- “Relational Operators Work with int64, uint64” on page 6-3
- “Reading HDF5 Files” on page 6-3

If you are upgrading from a release earlier than Release 13, then you should also see “New Features” on page 7-2 in the MATLAB 6.5 Release Notes.

MATLAB Interface to Generic DLLs

A shared library is a collection of functions that are available for use by one or more applications running on a system. On Windows systems, the library is precompiled into a dynamic link library (.dll) file. At run-time, the library is loaded into memory and made accessible to all applications. The MATLAB Interface to Generic DLLs enables you to interact with functions in dynamic link libraries directly from MATLAB.

Documentation

For help on this new feature, see “MATLAB Interface to Generic DLLs” in the External Interfaces documentation.

The examples used in the documentation use library (.dll) and header (.h) files located in the MATLABROOT\extern\examples\shrlib directory. To use these example files, first add this directory to your MATLAB path with the following command:

```
addpath([matlabroot ' \extern\examples\shrlib'])
```

Or you can make this your current working directory with this command:

```
cd([matlabroot ' \extern\examples\shrlib'])
```

Restrictions for This Release

- At this time, the MATLAB Interface to Generic DLLs is supported on Windows systems only.

- Passing a `void **` argument to a function in a dynamic link library is not supported in this release.
- Passing a complex structure argument to a function in a dynamic link library is not supported in this release. (The term *complex structure argument* refers to a structure constructed from other structures.)
- Passing an array of pointers, is not supported in this release. An example of an array of pointers is an array of strings.
- MATLAB does not support manipulation of pointers returned by functions in a dynamic link library at this time. An example of this type of operation is the addition or subtraction of pointers.

Function and Data Type Names in Generic DLL Interface

Minor changes have been made to the naming of some functions and data types in the Generic DLL interface. If you are upgrading from the post-release 13 download of MATLAB, see “Function and Data Type Names in Generic DLL Interface” on page 6-22 of these release notes.

Relational Operators Work with `int64`, `uint64`

All relational operators such as `<`, `>`, `<=`, `>=`, `~=`, and `==` now support `int64` and `uint64` data types.

Reading HDF5 Files

This release includes support for reading files that use the Hierarchical Data Format, Version 5 (HDF5). HDF5 is a product of the National Center for Supercomputing Applications (NCSA). The NCSA develops software and file formats for scientific data management.

This section includes this information:

- An overview of the structure of an HDF5 file
- Determining the contents of an HDF5 file
- Reading data from an HDF5 file
- Mapping HDF5 data types to MATLAB data types

Note MATLAB has supported reading and writing HDF files for several releases. The HDF and HDF5 specifications are not compatible.

Overview of HDF5 File Structure

HDF 5 files can contain multiple *datasets*. A dataset is a multidimensional array of data elements. Datasets can have associated metadata. HDF5 files store the datasets and attributes in a hierarchical structure, similar to a directory structure. The directories in the hierarchy are called *groups*. A group can contain other groups, datasets, attributes, links, and data types.

To illustrate this structure, the following figure shows the contents of the sample HDF5 file included with MATLAB, `example.h5`.

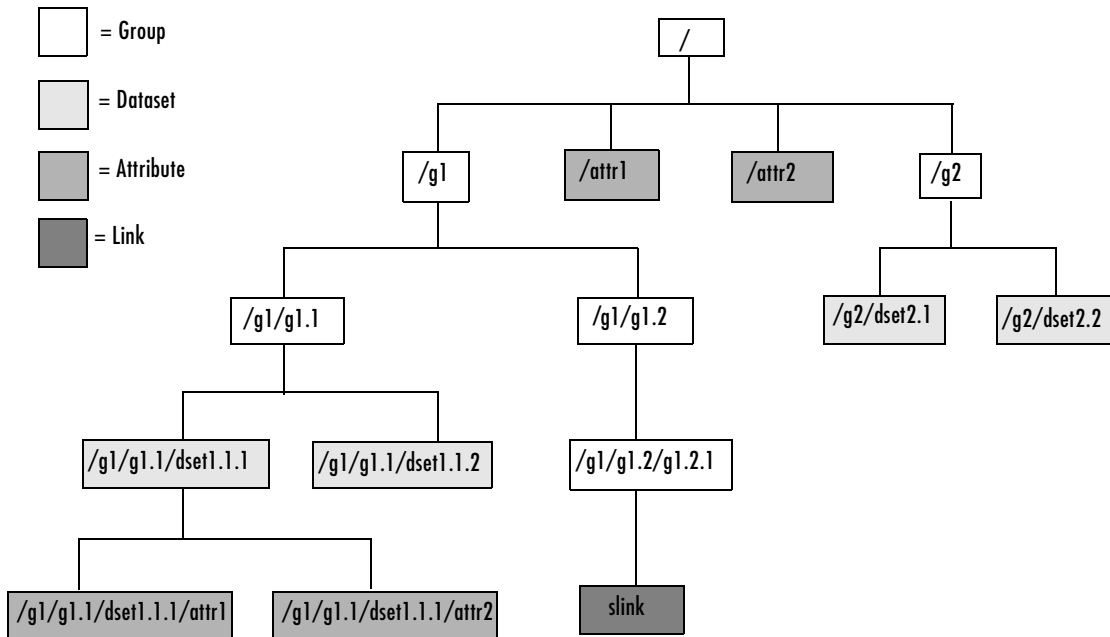


Figure 6-1: Hierarchical Structure of example.h5 HDF5 File

Determining the Contents of an HDF5 File

To extract an attribute or dataset from an HDF5 file, you must know the name of the attribute or dataset. You specify the name as an argument to the `hdf5read` function, described in “Reading Data from an HDF5 File” on page 6-6.

To find the names of all the datasets and attributes contained in an HDF5 file, you can use the `hdf5info` function. For example, to find out what the sample HDF5 file, `example.h5`, contains, use this syntax.

```
fileinfo = hdf5info('example.h5');
```

The `fileinfo` structure returned by `hdf5info` contains various information about the HDF5 file, including the name of the file and the version of the HDF5 library that MATLAB is using.

```
fileinfo =  
    Filename: 'example.h5'  
    LibVersion: '1.4.2'  
    Offset: 0  
    FileSize: 8172  
    GroupHierarchy: [1x1 struct]
```

To explore the contents of the file, examine the `GroupHierarchy` field.

```
level1 = fileinfo.GroupHierarchy  
  
level1 =  
  
    Filename: 'C:\matlab\toolbox\matlab\demos\example.h5'  
    Name: '/'  
    Groups: [1x2 struct]  
    Datasets: []  
    Datatypes: []  
    Links: []  
    Attributes: [1x2 struct]
```

The `GroupHierarchy` structure describes the top-level group in the file, called the root group. HDF5 uses the UNIX convention and names this top-level group `/` (forward slash), as seen in the `Name` field. The other fields in the structure describe the contents of the group. In the example, the root group contains two groups and two attributes. All the other fields, such as the

Datasets field, are empty. To traverse further down the file hierarchy, look at one of the structures in the Groups field.

```
level2 = level1.Groups(2)

level2 =

    Filename: 'C:\matlab\toolbox\matlab\demos\example.h5'
      Name: '/g2'
     Groups: []
   Datasets: [1x2 struct]
  Datatypes: []
     Links: []
  Attributes: []
```

In this group, the Groups field is empty and the Datasets field contains two structures. To get the names of the datasets, examine the Name field of either of these Dataset structures. This structure provides other information about the dataset including how many dimensions it contains (Dims) and the data type of the data in the dataset (Datatype).

```
dataset1 = level2.Datasets(1)

dataset1 =

    Filename: 'L:\matlab\toolbox\matlab\demos\example.h5'
      Name: '/g2/dset2.1'
      Rank: 1
   Datatype: [1x1 struct]
      Dims: 10
    MaxDims: 10
    Layout: 'contiguous'
  Attributes: []
     Links: []
  Chunksize: []
  Fillvalue: []
```

Reading Data from an HDF5 File

To read an HDF5 file, use the `hdf5read` function, specifying the name of the file and the name of the dataset as arguments. For information about finding the

name of a dataset, see “Determining the Contents of an HDF5 File” on page 6-5.

For example, to read the dataset, `/g2/dset2.1` from the HDF5 file `example.h5`, use this syntax:

```
data = hdf5read('example.h5', '/g2/dset2.1');
```

The return value `data`, contains the values in the dataset, in this case a 1-by-10 vector of single precision values.

```
data =

    Columns 1 through 8

    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000
    1.6000    1.7000

    Columns 9 through 10

    1.8000    1.9000
```

Mapping HDF5 Data Types to MATLAB Data Types

The `hdf5read` function maps HDF5 data types to MATLAB data types, depending on whether the data in the dataset is in an *atomic* data type or a *non-atomic* data type.

HDF5 Atomic Data Types. If the data in the dataset is stored in one of the HDF5 atomic data types, `hdf5read` uses the equivalent MATLAB data type to represent the data. Each dataset contains a `Datatype` field that names the data type. For example, the dataset `/g2/dset2.2` in the sample HDF5 file includes this data type information.

```
dtype = dataset1.Datatype
dtype =

    Name: []
    Class: 'H5T_IEEE_F32BE'
    Elements: []
```

The `H5T_IEEE_F32BE` class name indicates the data is a four-byte, big-endian, IEEE floating point data type. (See the HDF5 specification for more information about atomic data types.)

HDF5 Non-Atomic Data Types. If the data in the dataset is stored in one of the HDF5 non-atomic data types, `hdf5read` represents the dataset in MATLAB as an object. To access the data in the dataset, you must access the `Data` field in the object.

To illustrate, this example uses `hdf5read` to read a dataset called `/dataset2` from the HDF5 file, `my_hdf5_file.h5`. The dataset contains four elements; each element is an HDF5 array.

```
data = hdf5read('my_hdf5_file.h5', '/dataset2');
```

In MATLAB, the `hdf5read` function creates a `1x4` array of `hdf5.h5array` objects to represent this data.

```
whos

Name      Size      Bytes      Class

data      1x4                hdf5.h5array

Grand total is 4 elements using 0 bytes
```

Index into the MATLAB array to view the first element in the dataset.

```
data(1)

hdf5.h5array:

Name: ''
Data: [4x5x3 int32]
```

To look at the raw data in the HDF5 array element, access the `Data` field in the object.

```
data(1).Data

ans(:,:,1) =
0 1 2 3 4
10 11 12 13 14
```

```
20 21 22 23 24
30 31 32 33 34

ans(:,:,2) =
100 101 102 103 104
110 111 112 113 114
120 121 122 123 124
130 131 132 133 134

ans(:,:,3) =
200 201 202 203 204
210 211 212 213 214
220 221 222 223 224 230 231 232 233 234
```

The `hdf5read` function uses any of the following objects to represent HDF5 non-atomic data types.

- `hdf5.h5array`
- `hdf5.h5enum`
- `hdf5.h5vlen`
- `hdf5.h5compound`
- `hdf5.h5string`

Reading and Writing Data with JPEG Lossless Compression

MATLAB now supports reading and writing data that has been compressed using JPEG lossless compression. With lossless compression, you can recover the original image from its compressed form. Lossless compression, however, achieves lower compression ratios than its counterpart, lossy compression.

Using the `imread` function, you can read data that has been compressed using JPEG lossless compression.

Using the `imwrite` function, you can write data to a JPEG file using lossless compression. For the `imwrite` function, you specify the `Mode` parameter with the `'lossless'` value.

Reading and Writing $L^*a^*b^*$ Color Data

The `imread` function can now read color data that uses the $L^*a^*b^*$ color space from TIFF files. The TIFF files can contain $L^*a^*b^*$ values that are in 8-bit or 16-bit CIELAB encodings or in 8-bit or 16-bit ICCLAB encodings.

If a file contains 8-bit or 16-bit CIELAB data, `imread` automatically converts the data into 8-bit or 16-bit ICCLAB encoding. The 8-bit or 16-bit CIELAB data cannot be represented as a MATLAB array because it contains a combination of signed and unsigned values.

The `imwrite` function can write $L^*a^*b^*$ data to a file using either the 8-bit or 16-bit CIELAB encoding or the 8-bit or 16-bit ICCLAB encoding. You select the encoding by specifying the value of the `ColorSpace` parameter.

Major Bug Fixes

MATLAB 6.5.1 includes these major bug fixes:

- “Seeking Within a File” on page 6-11
- “Reshaping to More Than Two Dimensions” on page 6-11
- “mkdir No Longer Fails On Windows NT” on page 6-12
- “Using sqrt with Complex Input” on page 6-12
- “Multiplying Matrices with Non-Double Entries” on page 6-12
- “Sorting a Sparse Row Vector or Matrix” on page 6-12
- “diff Produces Correct Results with Logical Inputs” on page 6-13
- “Opening Modal Dialog with Third-Party GUI Open” on page 6-13
- “Serial Port Object with Latest Windows Service Pack” on page 6-13
- “OpenGL Problem Using Notebook” on page 6-13
- “Lcc C Compiler Fixed to Handle Large C Files” on page 6-13
- “Bug Fixes in MATLAB Interface to COM” on page 6-14
- “Bug Fixes in Creating GUIs” on page 6-19

Note In addition to the bug fixes described on this page, there are several bug fixes relating to MATLAB mathematics that are documented in a separate HTML bug-fix report.

Seeking Within a File

In Release 13, when you opened a file in write-only ('wb') mode, you could not seek to a position in the file without first seeking to the beginning of the file. The `fseek` function has been fixed to allow seeking from any position of the file.

Reshaping to More Than Two Dimensions

In Release 13, under certain circumstances, reshaping an array to have more than two dimensions produced a two dimensional result. This has been corrected.

mkdir No Longer Fails On Windows NT

In Release 13, if on Windows NT you called the `dir`, `exist`, `isdir`, or what function on a nonexistent directory name on a network drive, it caused a windows handle to remain open to that directory name until you exit the MATLAB session. This condition caused any attempts to use `mkdir` on that directory to fail. This problem also affected the `mkdir` command when run from a DOS command prompt. This condition would persist until you exited MATLAB, thus freeing the handle.

This bug is fixed in this release.

Using sqrt with Complex Input

In Release 13, under certain circumstances, the `sqrt` function incorrectly produced a real result when called with a complex input. This bug has been corrected.

Multiplying Matrices with Non-Double Entries

In Release 13, MATLAB gave an incorrect answer or crashed for expressions of the following forms:

- $A' * B$
- $A * B'$
- $A' * B'$
- $A.' * B$
- $A * B.'$
- $A.' * B.'$
- $A' * B.'$
- $A.' * B'$

when either A or B was a numeric, non-double value (`single`, `int32`, etc.). This has been fixed for this release.

Sorting a Sparse Row Vector or Matrix

In Release 13, a segmentation violation occurred when you used the command `sort(S, 2)` to sort a sparse row vector or to sort a sparse matrix along its rows. This bug is fixed in this release.

diff Produces Correct Results with Logical Inputs

In Release 13, the `diff` function could produce an incorrect result when you passed a logical array to it. This bug is fixed in this release.

Opening Modal Dialog with Third-Party GUI Open

In Release 13, MATLAB would occasionally hang if the user tried to open a modal dialog box when a third-party GUI was open. This no longer happens.

Serial Port Object with Latest Windows Service Pack

Under certain hardware configurations, or when using the latest Service Pack from Microsoft Windows, the serial port object in both MATLAB and the Instrument Control Toolbox could cause MATLAB to crash or hang. This problem is resolved in this release.

Several additional problems affecting the serial port have also been identified and fixed:

- 1 The serial port object now obeys all supported parity configurations.
- 2 The serial port object now obeys all supported flow control configurations.
- 3 On Windows, serial ports higher than COM8 were not recognized by MATLAB. As of this release, MATLAB supports up to 256 ports.
- 4 The serial port object generates output empty events after running the serial port object continuously.

OpenGL Problem Using Notebook

This version of MATLAB uses an improved algorithm for selecting pixel formats when using the `UseGenericOpenGL` feature on Windows. This improvement fixes rendering problems seen with Notebook.

For information on graphics rendering, see Tech Note 1201.

Lcc C Compiler Fixed to Handle Large C Files

Lcc version 2.4.1 MathWorks patch 1.29 corrects a bug encountered when compiling very large C files. Although this bug has only been observed when

using large Stateflow[®] models, we suggest that you upgrade to the new version to avoid potential problems when compiling MEX-files.

If you choose not to upgrade your version of Lcc, you can select a different C compiler using `mex -setup` from the MATLAB command line.

Bug Fixes in MATLAB Interface to COM

This release includes the following bug fixes in the COM interface:

- “Blank Spreadsheet Cells Returned as NaNs” on page 6-14
- “Importing Excel Worksheets Containing Currency Format” on page 6-15
- “Getting the Forms Font Interface” on page 6-15
- “Programmatic Identifiers Containing Space Characters” on page 6-15
- “Naming of Interfaces Returned by `invoke` or `get`” on page 6-15
- “Optional Input and Output Arguments Supported” on page 6-16
- “Memory Leak with MATLAB as COM Client” on page 6-16
- “Support for Multiple Type Libraries” on page 6-16
- “MATLAB Now Supports Skipping an Optional Argument” on page 6-16
- “Saving COM Objects Created with `actxserver`” on page 6-17
- “Creating Certain Servers That Do Not Have Type Libraries” on page 6-17
- “Creating Microsoft Controls” on page 6-18
- “ActiveX Controls Created with Visual Basic 6.0” on page 6-18
- “Type Mismatch Error Fixed” on page 6-18

Blank Spreadsheet Cells Returned as NaNs

When reading from a Microsoft Excel spreadsheet in a COM environment where MATLAB is the COM client and Excel the server, MATLAB now returns any empty cells in the spreadsheet as NaNs. In MATLAB 6.5 (Release 13), this same operation had returned a matrix of empty (`[]`) values.

For example, if the range A1 to D3 in a currently active workbook sheet contains no data, MATLAB 6.5.1 returns the following matrix of NaN values:

```
eActiveSheet = get(e, 'ActiveSheet');  
eActiveSheetRange = get(eActiveSheet, 'Range', 'A1', 'D3');  
  
eActiveSheetRange.Value
```

```
ans =
    [NaN]    [NaN]    [NaN]    [NaN]
    [NaN]    [NaN]    [NaN]    [NaN]
    [NaN]    [NaN]    [NaN]    [NaN]
```

Importing Excel Worksheets Containing Currency Format

In MATLAB 6.5, using a COM interface to Excel to import worksheet data containing currency format failed with either a field access error or segmentation violation. This bug is fixed in this release.

Getting the Forms Font Interface

In MATLAB 6.5, attempts to get the Font interface from a `forms.textbox.1` control, as done in the second line below, caused MATLAB to crash.

```
h=actxcontrol('forms.textbox.1')
font = h.Font
```

This bug is fixed in this release.

Programmatic Identifiers Containing Space Characters

Using the `actxcontrol` function with a ProgID argument containing one or more spaces failed in MATLAB 6.5. This bug is fixed in this release. For example, the following command now works:

```
h = actxcontrol('rmocx.RealPlayer G2 Control.1')
h =
    COM.rmocx.realplayer g2 control.1
```

Naming of Interfaces Returned by `invoke` or `get`

In MATLAB 6.5, interfaces returned by the `invoke` and `get` functions were given a name composed of the programmatic identifier (ProgID) for the component and the name of the method or property being invoked. In cases where a method or property implemented multiple interface types, this naming algorithm resulted in interface names that were not always unique.

For example, when invoking a method that returns an Excel and a Word interface, you could obtain any number of either type of interface (Excel or Word), but you could not obtain interfaces of both types. In such cases, you might be unable to access methods and properties of this interface.

In this release, interface names constructed by MATLAB are composed of the name of the type library and the class name, thus ending this potential naming conflict. If you invoke the method described in the last paragraph, MATLAB now returns the following for any Excel interfaces that you request:

```
Interface.Microsoft_Excel_9.0_Object_Library._Application
```

And MATLAB returns a different handle for Word interfaces:

```
Interface.Microsoft_Word_9.0_Object_Library._Application
```

Optional Input and Output Arguments Supported

MATLAB now supports optional input and output arguments passed in COM method calls. These arguments are declared as [in, optional] and [out, optional] respectively.

Memory Leak with MATLAB as COM Client

In Version 6.5, a memory leak developed under certain circumstances when MATLAB was configured as a COM client. This was caused by internal MATLAB code failing to release memory allocated by the method `StringFromCLSID`. This bug is fixed in this release.

Support for Multiple Type Libraries

MATLAB now supports multiple type libraries. If a COM object has many interfaces that are described in multiple type libraries, MATLAB can now retrieve the information correctly.

MATLAB Now Supports Skipping an Optional Argument

When calling ActiveX automation server methods, you can skip any optional arguments in the argument list by specifying that argument value as an empty matrix (`[]`). For example, the `Add` method shown below accepts as many as four optional arguments:

```
Add(Before, After, Count, Type)
```

To call this method, specifying values for `After` and `Count`, but no values for `Before` or `Type`, use this syntax.

```
addedsheet = invoke(Sheets, 'Add', [], Sheet1, 5);
```

Use [] for any arguments you skip, and that also precede the ones you do specify in the argument list. In this case, the Before argument is not specified but two subsequent arguments are.

Saving COM Objects Created with actxserver

Release 13 does not support saving COM objects that have been created with the actxserver function. You can use save only on control objects (created with actxcontrol). Attempting to use save on a COM server object causes MATLAB to hang temporarily, and eventually crash.

This bug has been fixed in this release so that if you now attempt to save a COM server object, MATLAB saves the object and any base properties of the object, but does not attempt to save any interfaces that might exist.

The same behavior applies to the pack function on COM objects.

This example creates a server running Microsoft Excel, adds a new property to the object, and saves it to the file excelserver.mat. It then reloads the server from the MAT-file.

```
e = actxserver ('Excel.Application');
addproperty(e, 'NewProperty');
set(e, 'NewProperty', 500);
get(e, 'NewProperty')
ans =
    500

save('excelserver.mat')
clear
get(e, 'NewProperty')
??? Undefined function or variable 'e'.

load('excelserver.mat')
get(e, 'NewProperty')
ans =
    500
```

Creating Certain Servers That Do Not Have Type Libraries

In the Release 12.1 and Release 13 releases, the actxserver function generated an error when attempting to create a COM object for certain servers. One error commonly returned by actxserver in these releases was

```
h = actxserver('msdev.application')
??? Error using ==> actxserver
Cannot find type library. COM object creation failed.
```

This has now been fixed in this release.

```
h = actxserver('msdev.application')
h =
    COM.msdev.application
```

Creating Microsoft Controls

Earlier versions of MATLAB would crash if you attempted to create certain Microsoft COM controls with the `actxcontrol` function. Examples of these controls, by programmatic identifier (ProgID), are shown below. MATLAB now successfully creates the controls.

```
mschart20lib.mschart      msdatalistlib.datacombo
msdatagridlib.datagrid    MComCtl2.DTPicker.2
msdatalistlib.datalist    MSHierarchicalFlexGridLib.MSHFlexGrid.6
```

ActiveX Controls Created with Visual Basic 6.0

In Release 13, if you attach a callback routine to an event, and this event is eventually fired by a control created in Visual Basic 6.0, an error dialog box appears with the message “Run-Time error.”

This has been fixed in this release.

Type Mismatch Error Fixed

Some COM objects may define methods that pass scalar inputs by reference. This might appear in a type library signature as shown here for the `x` input:

```
functionname(double *x, [out] double *y)
```

Note that when input or output is not specifically stated, as is the case here for `x`, MATLAB defaults to input (`[in]`). So the line shown above is interpreted by MATLAB as

```
functionname([in] double *x, [out] double *y)
```

In MATLAB, the `[in]` and by-reference (`*`) specifications are considered incompatible for scalar arguments. In Release 13, MATLAB ignores the by-reference specifier for scalar inputs and passes such arguments by value

instead. Thus, any modified value for such an argument is not received by the calling function. You may also see a type mismatch error displayed, even when trying to access valid control methods.

MATLAB 6.5.1 fixes this bug by treating the [in] specifier for scalar references as if it were [in,out].

In this example using MATLAB syntax, the GetWinVersionX function passes six double arguments by reference, yet none are returned in MATLAB 6.5:

```
GetWinVersionX = int32 GetWinVersionX(  
    handle, double, double, double, double, double, double)
```

In MATLAB 6.5.1, all scalar reference arguments specified (or defaulting to [in]) are treated as [in,out], and all references cause a value to be returned:

```
GetWinVersionX = [int32, double, double, double, double,  
    double, double] GetWinVersionX(  
    handle, double, double, double, double, double, double)
```

Note that this bug affects only scalar arguments. The VT_DISPATCH and VT_VOID types are not affected.

Bug Fixes in Creating GUIs

This release includes the following bug fixes related to creating, converting, and exporting GUIs:

- “Converting a MATLAB 5.3 GUI to MATLAB 6.5” on page 6-19
- “Using GUIDE on Existing GUIs with Empty Tag Property” on page 6-20
- “Exporting GUIs from GUIDE to a Single M-file” on page 6-20
- “MATLAB Hangs when Using Property Inspector from GUIDE” on page 6-20
- “Recursion Limit Error when Running Existing GUIs from GUIDE” on page 6-20

Converting a MATLAB 5.3 GUI to MATLAB 6.5

Converting a MATLAB 5.3 (R11) GUI to MATLAB 6.5 sometimes resulted in the error:

```
Unhandled internal error in guidemfile. Reference to non-existent  
field 'blocking'
```

This problem has been fixed.

Using GUIDE on Existing GUIs with Empty Tag Property

In MATLAB Version 6.5, editing a GUI that contained a uicontrol whose **Tag** property was set to [] (empty) sometimes generated the following error message:

```
Unhandled internal error in guidefunc.  
Error using ==> set  
Value must be a string
```

This problem has been fixed.

Exporting GUIs from GUIDE to a Single M-file

In MATLAB Version 6.5, some GUIs exported from GUIDE failed to open. In other cases, attempting to export a GUI resulted in one of the following errors:

```
??? Error using ==> guidefunc  
Error using ==> ==  
Matrix dimensions must agree.  
  
??? Error using ==> guidefunc  
Error using ==> ==  
Function '==' is not defined for values of class 'struct'.
```

These problems have been fixed.

MATLAB Hangs when Using Property Inspector from GUIDE

Using the Property Inspector from GUIDE sometimes caused MATLAB Version 6.5 to hang. This problem has been fixed.

Recursion Limit Error when Running Existing GUIs from GUIDE

In MATLAB Version 6.5, running some existing GUIs from GUIDE generated the following error message:

```
??? Error using ==> guidefunc  
Maximum recursion limit of 500 reached. Use  
set(0,'RecursionLimit',N) to change the limit. Be aware that  
exceeding your available stack space can crash MATLAB and/or  
your computer.
```


Could not create figure:
127

This problem has been fixed.

Upgrading from an Earlier Release

If you are upgrading from a release earlier than Release 13, see “Upgrading from an Earlier Release” on page 7-49 of MATLAB 6.5 Release Notes.

Rebuild Macintosh MEX-files

Macintosh MEX-files (named `.mex`) built with MATLAB 5.2 or older will not work with MATLAB 6.5 or later. You must recompile these files, creating a new file with the file extension `.mexmac`.

Function and Data Type Names in Generic DLL Interface

The following functions have been renamed since the initial download release of the Generic DLL Interface:

- The `libmethods` function is now called `libfunctions`.
- The `libmethodsview` function is now called `libfunctionsview`.

All data types ending in `Ref` are now suffixed with `Ptr`. For example, `doubleRef` is now called `doublePtr`, and `int16Ref` is now `int16Ptr`.

All data types ending in `RefPtr` are now suffixed with `PtrPtr`. For example, `doubleRefPtr` is now called `doublePtrPtr`, and `int16RefPtr` is now `int16PtrPtr`.

Known Software and Documentation Problems

For a list of bugs reported in the previous release that remain open, see “Known Software and Documentation Problems” on page 7-86 in the MATLAB 6.5 Release Notes.

Using xlsfinfo on Systems Without Excel

There is a bug in the `xlsfinfo` function that causes it to fail with the following error message when run on systems where Microsoft Excel is not installed.

```
Undefined function or variable 'xlsfinfo_old'.
```

We intend to fix this in the next release of MATLAB.

MATLAB 6.5 Release Notes

| | |
|--|----------|
| New Features | 1-2 |
| Development Environment Features | 1-2 |
| Mathematics Features | 1-15 |
| Programming and Data Types Features | 1-21 |
| Programming Tips Documentation | 1-30 |
| Graphics Features | 1-31 |
| External Interfaces/API Features | 1-32 |
| Creating Graphical User Interfaces (GUIDE) Features | 1-40 |
| Major Bug Fixes | 1-43 |
| Platform Limitations | 1-44 |
| Patch Required for HP-UX 11.0 | 1-44 |
| Development Environment Limitations | 1-44 |
| Mathematics Limitations | 1-46 |
| Graphics Limitations | 1-47 |
| Creating Graphical User Interfaces (GUIDE) Limitations | 1-47 |
| You May Need to Overwrite the MATLAB Default Choice of BLAS | 1-47 |
| Upgrading from an Earlier Release | 1-49 |
| Development Environment Upgrade Issues | 1-49 |
| Mathematics Upgrade Issues | 1-51 |
| Programming and Data Types Upgrade Issues | 1-52 |
| Graphics Upgrade Issues | 1-75 |
| External Interfaces/API Upgrade Issues | 1-76 |
| Creating Graphical User Interfaces (GUIDE) Upgrade Issues | 1-85 |
| Known Software and Documentation Problems | 1-86 |

New Features

This section introduces the new features and enhancements added in MATLAB 6.5 since Version 6.1 (Release 12.1). This discussion of new MATLAB features is organized into the following categories:

- “Development Environment Features” on page 7-2
- “Mathematics Features” on page 7-15
- “Programming and Data Types Features” on page 7-21
 - “Programming Tips Documentation” on page 7-30
- “Graphics Features” on page 7-31
- “External Interfaces/API Features” on page 7-32
- “Creating Graphical User Interfaces (GUIDE) Features” on page 7-40

If you are upgrading from a release earlier than Release 12.1, then you should also see “New Features” on page 8-2 in the MATLAB 6.1 Release Notes.

Development Environment Features

MATLAB 6.5 adds the following development environment features and enhancements.

JVM Version

On the Windows, Linux, Solaris, and Macintosh platforms, MATLAB uses Java Virtual Machine 1.3.1. Other platforms that support Java continue to use the JVM version they used for Release 12. To see the Java version that MATLAB uses, type

```
version -java
```

The HP-UX and IBM platforms do not support Java-based graphical user interfaces in MATLAB, and related products that rely on Java are not available on these platforms. See “Platform Limitations” on page 7-44 for details.

Startup

The toolbox path caching preference is on by default. This can result in significantly faster startup when MATLAB runs over a network or when you have many toolboxes. You will not see the improvement the first time you run MATLAB 6.5, but will after that. If you add or remove files and directories from `$matlabroot/toolbox` directories, you may need to update the cache.

Desktop

Start Button. Click the **Start** button,  **Start**, located in the lower left corner of the desktop, to readily access common MATLAB tools and features. It offers capabilities similar to those in the Launch Pad.

Status Bar. The status bar in the desktop now indicates the current state of MATLAB operations. For example, a Busy message appears while MATLAB is running an M-file.

New Profiler. Use the new Profiler graphical interface to assess your M-files so you can make changes to improve their performance. Select **View -> Profiler** from the desktop, or type `profile viewer`. The Profiler helps you take advantage of the new performance improvements that are part of the JIT Accelerator for MATLAB.

The new Profiler is based on the results returned by the `profile` function. You can still use the `profile` and `profreport` functions as you used them in Release 12.1.

Check for Updates. From the **Web** menu, select **Check for Updates**. A dialog box appears, listing the versions for all MathWorks products installed on your system. Click **Check for Updates** in the dialog box, which accesses the MathWorks Web site to determine if more recent versions are available.

Access MATLAB Central. From the **Web** menu, select **MATLAB Central** to access a page on the MathWorks Web site for exchanging M-files with other users and for accessing the `comp-sys.soft.matlab` Usenet newsgroup.

Change Current Directory. On UNIX platforms, you can now change the current directory field in the desktop toolbar using the `...` button to browse for the directory.

Apply Preferences. There is now an **Apply** button in the **Preferences** dialog box. When you click **Apply**, the preference change is made, but the dialog box remains open. This allows you to more easily experiment with changes to preferences.

Command Window

Find Feature. To find a term in the Command Window, select **Edit -> Find**. The **Find** dialog appears, in which you can enter a term and look for the previous or next occurrence.

Incremental Search. This is similar to the Emacs incremental search feature. In the Command Window, press **Ctrl+S** (or **Ctrl+Shift+S** for Windows key bindings) to display an incremental search field. Type a string in the field and the next occurrence of that string in the Command Window is highlighted.

Hyperlinks to Run Functions. A new feature, `matlab:`, creates a hyperlink for specified text, which when clicked, runs the specified function. For example,

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

displays the link

Generate magic square

in the Command Window. When the user clicks the "Generate magic square" link, MATLAB runs `magic(4)`. Use this feature, for example, with the `disp` or `fprintf` functions.

Printing. You can now specify options for printing from the Command Window, such as including a header and printing line numbers. Select **File -> Page Setup** to set options.

Preferences. There are new Command Window preferences for **Keyboard and Indenting**:

- **Command line key bindings**—Specify **Emacs (MATLAB standard)** or **Windows**. For example, with Emacs, **Ctrl+F** moves the cursor forward one character, whereas with Windows, **Ctrl+F** opens the **Find** dialog box.
- **Tab key**—These preferences previously existed on the general preferences tab for the Command Window.
- **Parentheses matching options**—MATLAB alerts you to matches and mismatches in pairs of delimiters, (that is, in parentheses (), brackets [], and braces { }), based upon MATLAB language syntax rules.

Open Selection. While in the Command Window, you can select text, right-click, and select **Open selection**. This runs the open function for the item you selected so that it opens in the appropriate tool. For example, you can open a variable in the Workspace browser, or open a file or function in the Editor. If no tool exists for the selected item, **Open selection** calls edit.

Command History

Printing. You can print the contents of the Command History and specify various printing options, such as including a header and printing line numbers. From the Command History window, select **File -> Page Setup** to set options.

Find Feature. To find a term in the Command History, select **Edit -> Find**. The **Find** dialog appears, in which you can enter a term and look for the previous or next occurrence.

Autosave Command History. The Command History is automatically saved to a file on a regular basis. Specify options for what is saved and how often using Command History preferences.

Workspace Browser

You can rename a variable in the Workspace browser by right-clicking it and selecting **Rename** from the context menu. You can also select and copy variables in the Workspace browser, which puts their names (comma separated) onto the clipboard. You can then paste the names, for example, into the Command Window.

If you copy data from another application to the clipboard, use **Ctrl+V** in the Workspace browser to import the data to MATLAB using the Import Wizard.

Set Path

In the **Set Path** dialog box, you can now select multiple directories to remove from or to move within the path.

Current Directory Browser

Find M-Files Only. There are two new **Look in** options in the **Find** dialog box. Use them to limit the search in the current directory or in the entire MATLAB path to find only M-files.

Deleted Files to Recycle Bin. On Windows platforms, files you delete while using the Current Directory browser go to the Recycle Bin. You can bypass the Recycle Bin by using **Shift+Delete**.

Change Current Directory. For UNIX platforms, you can now change the current directory field in the Current Directory browser toolbar using the ... button to browse for the directory.

Changes Automatically Update Display. When you make changes to the current directory outside of MATLAB, the changes are automatically reflected in the Current Directory browser display. You do not have to select **Refresh** to show the changes.

File Operations

Following are functions that are new or have changed since the previous release. For more information, type `doc functionname`.

| Function | New or Changed | Description |
|-------------------------|----------------|--|
| <code>copyfile</code> | Changed | The <code>writable</code> argument has been superseded by the <code>f</code> argument, although <code>writable</code> is still allowed for MATLAB 6.5. The function now also copies directories. It replaces the destination files or directories with the same name as the source files or directories without a warning—in previous versions, there was a warning in that event. If the destination files or directories are read-only and the <code>f</code> (or <code>writable</code>) argument is not used, <code>copyfile</code> will fail. |
| <code>fileattrib</code> | New | Set or get attributes of file or directory. The <code>fileattrib</code> function is like the DOS <code>attrib</code> command and the UNIX <code>chmod</code> command. |
| <code>mkdir</code> | Changed | Modified the message status— <code>mkdir</code> no longer returns 2 if the directory already exists, but instead displays a warning. It also has an enhanced return format. |
| <code>movefile</code> | New | Move file or directory. Can also be used to rename a file or directory. |
| <code>rmdir</code> | New | Remove directory, and optionally its contents as well. |
| <code>winopen</code> | New | For Windows users, allows you to open a file in the appropriate application, as if you double-clicked it in Windows Explorer. |

Array Editor

Spreadsheet Behavior. You can now select and delete columns. You can cut, copy, paste, and delete cells. You can also exchange cells with Microsoft Excel via the operating system clipboard using these features.

You can set a preference to specify where the cursor moves to when you press **Enter**.

Greater Number of Elements. The Array Editor can now show arrays with more than 10,000 elements. It does not support arrays with more than 65,536 (2^{16}) elements.

Editor/Debugger

Column Number, Line Number, and Current Function/Subfunction. In the Editor status bar, you can see the column number, line number, and function or subfunction for the current cursor location. When the Editor is docked in the desktop, the information appears in the desktop status bar.

Autosave Files. Files you change in the Editor are now automatically saved to a backup file on a regular basis. Use **File -> Preferences -> Editor/Debugger -> Autosave** to specify autosave options.

Incremental Search. This is similar to the Emacs incremental search feature. Press **Ctrl+S** (or **Ctrl+Shift+S** for Windows key bindings) to display an incremental search field. Type a string in the field and the next occurrence of that string in the current file is highlighted.

Find Previous. You can find the previous occurrence of a string in a file by using **Edit -> Find Previous** after using any of the other **Edit -> Find** menu items.

Comment Formatting. You can specify a preference for the maximum width of comment lines and then apply that maximum to selected lines. You can also specify a preference to automatically wrap comment lines when they reach the maximum width.

Preferences for Parentheses Matching. There are new preferences for parentheses matching. The Editor/Debugger alerts you to matches and mismatches in pairs of delimiters, that is, in parentheses (), brackets [], and braces { }, based upon MATLAB language syntax rules.


Printing Options. You can specify options for printing a file from the Editor, for example, including a header, by using **File -> Page Setup**.

Invalid Breakpoints. When breakpoints are invalid, they appear gray instead of red. Breakpoints are invalid if there are unsaved changes or if there is a syntax error in the file. The breakpoints become valid when you save the file or when you fix the syntax error and save the file.

Cannot Save in Debug Mode. You cannot save changes to an M-file while in debug mode. First quit debug mode and then save the file.

Integrated Text Editor Preference. If you install EmacsLink, a tool that allows you to debug M-files from the Emacs editor, you can specify an Editor/Debugger preference to use Emacs and EmacsLink for your default editing and debugging tools.

Wider and Resizable Line Number Column. You can view line numbers that contain up to nine digits. Drag the separator bar to the right of the line number column to make the column narrower or wider, allowing you to save space or see more digits.

Subfunctions Listed Alphabetically. When you click the function button  on the toolbar, the subfunctions are listed alphabetically. Previously they were listed in the order that they appeared in the M-file.

Open Selection. In an open M-file, the **Open Selection** feature has been enhanced. You can now jump from a subfunction call to the subfunction code within the current function M-file. To use this feature, select a subfunction call in an M-file, right-click, and select **Open Selection** from the context menu (or select **Open Selection** from the **File** menu). The Editor scrolls to that subfunction in the M-file. If that subfunction does not exist in the file, the open function runs for the selected item, so that it opens in the appropriate tool. For example, you can open a variable in the Workspace browser, or open another file or function in the Editor. If no tool exists for the selected item, **Open selection** looks for a matching file in a private directory in the current directory.

Default Directory in Open Dialog Box and for New Files. The **Open** dialog box now opens to the MATLAB current directory. However, if you access the **Open** dialog box from the Editor, it opens to the directory for the current file in the

Editor. When you create a new file, it is located in the MATLAB current directory.

Bookmark Support for All File Types. You can include bookmarks in any file type. Previously you could include bookmarks only in M-files.

New and Discontinued Features in edit Function. There is a new form of the edit function, `edit fun1 fun2 fun3 ...`, which opens all of the specified files in the default editor.

No longer supported are `edit fun1 in fun2` and `edit fun(a,b,c)`.


Save Not Available. Save is only available if a file has been changed. If there are no unsaved changes in a file, you can still use **Save As**, but you cannot use **Save**.

Help and Help Browser

Demos. To access demonstrations for all the MathWorks products you have installed, use the new **Demos** tab in the Help browser.

Boolean Operators in Search. In the Help browser **Search** pane, you can include the Boolean operators AND, OR, and NOT between terms you enter in the **Search** field. The operators must be in all capital letters and there must be a space before and after each operator.

For example, type `print OR printing AND figure NOT exporting` to find all pages that contain the words `print` and `figure`, or `printing` and `figure`, but only if the page does not contain the word `exporting`. At the top of the results list are any pages that contain all the AND and OR words in page titles.



Changes to Search Term Highlights. When you perform a search and select a resulting page to view, each word in the search term appears highlighted in a different color in the page. To clear the highlights, click the reload button  in the Help browser toolbar.

Setting the Documentation Location. You can now set the location where help files are stored (called the **Documentation location** in Help preferences) using the `docroot` function. You can include a `docroot` command in an M-file, such as a `startup.m` file.

Open Link in New Window. To open a linked page in a separate Help browser window, press **Alt** and right-click the link, or click the middle mouse button.

Visited Links. Visited links usually appear in a different color than unvisited links.

Print Range of Pages. When you print from the Help browser, the **Pages** field in the **Print** dialog box now shows the total number of printed pages required for the currently displayed page and lets you specify which of those pages to print.

Document Type Icons. Icons at the top two levels of the **Contents** pane indicate the type of document so you can quickly find a particular document type in the listing. For example, getting started documentation is represented by  (a green arrow), and function and blockset reference documentation are represented by  (orange pages).

Release Notes Location. Release notes for a product are now listed with that product in the **Contents** pane.

CD-ROMs for Documentation. For Windows platforms, there are now two CD-ROMS for documentation. To read PDF documentation from the CD-ROM, use the PDF Documentation CD.

Source Control

This release features expanded source control capabilities on PC platforms. You can interface to your source control system from by using MATLAB, Simulink, or Stateflow menus, or by using functions from the MATLAB Command Window.

The available source control system interface operations on PC platforms are

- Get latest version of file
- Add file
- Check out file
- Check in file
- Undo check out
- Remove file
- Show file version history
- Show file version differences
- Show file properties
- Start source control system

Notebook

Word Macros. Newer versions of Word have macro security features that might impact your use of Notebook.

Word Versions Supported. Word for Office 2000 and 2002 (Office XP) are now supported. Word for Office 95 is no longer supported.

Using setup Option. The setup option is now easier to use. After running `notebook -setup`, you are prompted for your Windows version. The function performs all the remaining configuration with no additional input required from you. Only if the setup cannot find the files needed will you be prompted for additional information.

General

Demos. View and run demos via the new **Demos** pane in the Help browser. For platforms that do not support Java, run the `demo` command, which opens the Release 12.1 demo interface, and then follow instructions to access the demo files.

New perl Function. A new function, `perl`, calls the Perl script specified by the file `perlfile` using the appropriate Perl executable.

New Startup Option. There is a new startup option, `-logfile log`. It makes a copy of any output to the Command Window in the file `log`, including any crash reports.

Import/Export

New Functions for Exchanging Files with the Internet. MATLAB provides a set of functions for exchanging files with the Internet. These are URL, ZIP, and e-mail functions.

- Downloading URL content—From within MATLAB, you can read and save the content of a URL. The `urlread` function reads the content to a string variable in the MATLAB workspace. The `urlwrite` function saves the content to a file.
- ZIP functions—You can compress and uncompress files and directories from MATLAB using the `zip` and `unzip` functions.
- Sending e-mail—Use `sendmail` to send an electronic mail message, and optionally attachments, to a list of addresses.

File Format Support. The following table lists new import and export functions and highlights changes to existing functions.

| Function | Purpose |
|-----------------------|---|
| <code>cdfepoch</code> | This new function converts a MATLAB date number or date string into the format supported by the Common Data Format (CDF). |
| <code>cdfwrite</code> | This new function supports writing data from MATLAB into Common Data Format (CDF) files. |
| <code>imfinfo</code> | The <code>imfinfo</code> function can now return information about Sun Raster image (RAS) files. In addition, when used with JPEG files, <code>imfinfo</code> now returns any comments that may be included in the graphics file. These comments are returned in the comment field as a cell array. |

| Function | Purpose (Continued) |
|-----------------------------|---|
| <code>imformats</code> | This new function eases the task of adding read and write support for new file formats. |
| <code>imread</code> | The <code>imread</code> function can now read Sun Raster image (RAS) files and files over the internet. |
| <code>imwrite</code> | The <code>imwrite</code> function now supports two new formats: Sun Raster image (RAS) and PNM. PNM is not a file format but represents three other image formats, PBM, PGM, and PPM. When you specify PNM, <code>imwrite</code> chooses one of these three formats based on the contents of the data. In addition, <code>imwrite</code> supports the JPEG-specific parameter comment, in which you can specify any comment you want included in a JPEG file. |
| <code>multibandread</code> | This new function supports importing data from files that contain data divided into multiple bands, sometimes called raw files. |
| <code>multibandwrite</code> | This new function supports writing multidimensional arrays from MATLAB to a file in multiband format. |

MATLAB HDF Import Tool. MATLAB 6.5 includes a new graphical user interface for importing data from an HDF or HDF-EOS files. This tool provides a graphical view of the data sets and metadata in an HDF file and lets you import selected data sets from the file by clicking a button.

Mathematics Features

MATLAB 6.5 adds the following mathematics features and enhancements:

- “Delay Differential Equations” on page 7-15
- “Singular Boundary Value ODE Problems” on page 7-15
- “Integration Over a Volume” on page 7-16
- “Sparse, Square, Banded Matrix Left Division” on page 7-16
- “Sparse Matrix LU Factorization and Solve” on page 7-16
- “Matrix Math Performance Improvements for Triangular Matrices” on page 7-17

These features are described below. At the end of this section are tables that summarize changes to the MATLAB math functions:

- “Summary of New Functions” on page 7-18
- “Summary of Changed Functions” on page 7-18

Delay Differential Equations

MATLAB now provides the capability to solve delay differential equations (DDEs) with constant delays. The DDE solver, `dde23`, provides an interface that is similar to the MATLAB ODE solver interface and is as easy to use. The supporting functions `ddeset`, `ddeget`, and `deval` enable you to set integration properties that affect problem solution and to evaluate the numerical solution obtained with `dde23`.

See “Initial Value Problems for DDEs” and the function descriptions in the MATLAB documentation for detailed information.

Singular Boundary Value ODE Problems

The function `bvp4c` can now solve a class of singular BVPs of the form

$$y' = \frac{1}{x}Sy + f(x, y)$$

$$0 = g(y(0), y(b))$$

posed on an interval $[0, b]$ with $b > 0$. The `bvpset` function provides a new 'SingularTerm' integration property, which you can use to pass the constant matrix S to `bvp4c`.

See “Solving Singular BVPs” and the function descriptions in the MATLAB documentation for more information.

Integration Over a Volume

A new function, `triplequad`, evaluates a triple integral that you provide as a function, `fun(x,y,z)`, over a three dimensional rectangular region. As a default, `triplequad` uses the quadrature function `quad` to perform the integration. You can elect to use `quadl` instead or provide your own quadrature function.

Logarithmic Derivative of the Gamma Function

A new function `psi` evaluates the ψ function, also known as the digamma function, for each element of an array `X`. You can also use `psi` to evaluate the `k`th derivative of ψ , or a sequence of derivatives of different orders, at the elements of `X`.

Sparse, Square, Banded Matrix Left Division

Matrix left division (`\`) now uses banded solvers for $X = A \setminus b$ where `A` is sparse, square, and banded. Band density is defined as $(\# \text{ nonzeros in the band}) / (\# \text{ nonzeros in a full band})$. Band density = 1.0 if there are no zeros on any of the three diagonals.

If `A` is real and tridiagonal, i.e., band density = 1.0, and `B` is real with only one column, `X` is computed quickly using Gaussian elimination without pivoting.

If the tridiagonal solver detects a need for pivoting, or if `A` or `B` is not real, or if `B` has more than one column, but `A` is banded with band density greater than the new `spparms` parameter 'bandden' (default = 0.5, in the interval `[0.0, 1.0]`), then `X` is computed using LAPACK.

Sparse Matrix LU Factorization and Solve

LU factorization and solve for sparse matrices now uses UMFPACK. UMFPACK is a set of routines for solving unsymmetric sparse linear systems, $Ax = b$, using the Unsymmetric MultiFrontal method. It provides a considerable increase in computational speed for these matrices.

lu Function. The `lu` function provides two new syntaxes for sparse matrices. These new syntaxes use UMFPACK for factorization.

```
[L,U,P,Q] = lu(X)
[L,U,P,Q] = lu(X,thresh)
```

The syntaxes return a unit lower triangular matrix `L`, an upper triangular matrix `U`, and permutation matrices `P` and `Q`, so that $P*X*Q = L*U$. The `thresh` argument (default = 0.1, in the interval [0.0, 1.0]) controls pivoting.

\ (backslash). Matrix left division (`\`) uses UMFPACK for square sparse matrices that are not banded. You can control pivoting with the new `spparms` parameter `'piv_tol'` (default = 0.1, in the interval [0.0, 1.0]).

Information about UMFPACK is available online at <http://www.cise.ufl.edu/research/sparse/umfpack/>. The *UMFPACK Version 4.0 User Guide* is available at <http://www.cise.ufl.edu/research/sparse/umfpack/v4.0/UserGuide.pdf>. Type `help umfpack` at the command line for summary copyright and licensing information.

Matrix Math Performance Improvements for Triangular Matrices

The speed for solving linear systems $AX = B$ where `A` is upper or lower triangular, and `B` is an `m`-by-`n` matrix, has been improved through the use of optimized Basic Linear Algebra Subroutines (BLAS). Optimized BLAS is provided by Automatically Tuned Linear Algebra Software (ATLAS).

BLAS has also been used to improve certain matrix multiplication operations, i.e., `matrix*vector`, `vector*matrix`, and `rowvector*columnvector`.

For the first time, ATLAS BLAS have been tuned to the Pentium 4 under both Windows and Linux operating systems, resulting in improved speed for core linear algebra functions.

By making better use of cache, the speed of matrix transposition has been increased for all matrices, but particularly for matrices whose size is a power of 2.

Summary of New Functions

| Function | Purpose |
|------------|---|
| dde23 | Solve initial value problems for delay differential equations (DDEs) with constant delays |
| ddeget | Extract properties from the options structure created with ddeset |
| ddeset | Create/alter a DDE options structure that contains solver integration properties |
| psi | Psi (polygamma) function, i.e., the logarithmic derivative of the gamma function |
| triplequad | Numerically evaluate triple integral |

Summary of Changed Functions

| Function | Enhancement or Change |
|----------------------------|--|
| / (slash) \ (backslash) | <p>Now use banded solvers for sparse, square, banded matrices. See “Sparse, Square, Banded Matrix Left Division” on page 7-16 for more information.</p> <p>Now use UMFPACK for left and right division of square sparse matrices that are not banded. See “Sparse Matrix LU Factorization and Solve” on page 7-16 for more information.</p> |
| / (slash) \ (backslash) | <p>The result of dividing a singular lower or upper triangular matrix by any other matrix, using either left (\) or right (/) division, may change. Previously, for singular square matrices A for which $rcond(A) = 0$, the result was always a matrix of Infs. This change is a result of the performance improvements described above.</p> <p>See “Mathematics Upgrade Issues” on page 7-51 for examples.</p> |

| Function | Enhancement or Change (Continued) | |
|-----------------|--|--|
| bvp4c bvpset | <p>A new option 'SingularTerm' enables you to specify a matrix as the singular term of singular BVPs. Set this option to the constant matrix S for equations of the form</p> $y' = S \frac{y}{x} + f(x, y, p)$ | |
| corrcoef | <p>Provides three new syntaxes:</p> <p>$[R,P] = \text{corrcoef}(\dots)$ returns P, a matrix of p-values for testing the hypothesis of no correlation.</p> <p>$[R,P,RLO,RUP] = \text{corrcoef}(\dots)$ returns matrices RLO and RUP which contain lower and upper bounds for a 95% confidence interval for each coefficient.</p> <p>$[\dots] = \text{corrcoef}(\dots, 'param1', val1, 'param2', val2, \dots)$ accepts parameter-value pairs that enable you to override the default confidence interval, and specify how to treat rows of X that contain NaNs.</p> | |
| deval | Now also accepts output from dde23 | |
| gallery | house | <p>A new syntax</p> <p>$[v,beta,s] = \text{gallery}('house', x, k)$ returns the norm of x. You can use the new argument k to control the sign of s.</p> |
| | leslie | <p>$\text{gallery}('leslie', a, b)$ returns the n-by-n Leslie matrix with average birth numbers $a(1:n)$ and survival rates $b(1:n-1)$.</p> |
| | orthog | <p>$\text{gallery}('orthog', n, k)$ adds a new type, k, of matrix. $k = 6$ specifies a symmetric matrix arising as a discrete cosine transform such that $Q(i, j) = \sqrt{2/n} * \cos((i-1/2)*(j-1/2)*\pi/n)$.</p> |

| Function | Enhancement or Change (Continued) | | |
|----------------------|--|---------|---|
| | <table border="1"> <tr> <td data-bbox="602 309 765 439">randsvd</td> <td data-bbox="765 309 1338 439">For large dimensions, a new argument, method, enables you to specify an alternative method that is much faster for large dimensions even though it uses more flops.</td> </tr> </table> | randsvd | For large dimensions, a new argument, method, enables you to specify an alternative method that is much faster for large dimensions even though it uses more flops. |
| randsvd | For large dimensions, a new argument, method, enables you to specify an alternative method that is much faster for large dimensions even though it uses more flops. | | |
| legendre | A new syntax <code>legendre(n,X,'norm')</code> computes the fully normalized associated Legendre functions $N_n^m(x)$. | | |
| lsqr | A new syntax <code>[x,flag,relres,iter,resvec,lsvect] = lsqr(...)</code> returns, in the vector <code>lsvect</code> , estimates of the scaled normal equations residual at each iteration. | | |
| lu | <p>Uses UMFPACK to improve speed for factorization of sparse matrices, and to add two new syntaxes for sparse matrices.</p> <pre>[L,U,P,Q] = lu(X) [L,U,P,Q] = lu(X,thresh)</pre> <p>The new output Q is the column permutation matrix that is used to reduce fill-in in the sparse case. P is the row permutation matrix that is used for numerical stability. The <code>thresh</code> argument controls pivoting. See “Sparse Matrix LU Factorization and Solve” on page 7-16 for information about UMFPACK.</p> | | |
| qrdelete qrinsert | <p>Two new syntaxes for each function provide for the deletion and insertion of rows, as well as columns, in a QR factorization:</p> <pre>[Q1,R1] = qrdelete(Q,R,j,'col') [Q1,R1] = qrdelete(Q,R,j,'row') [Q1,R1] = qrinsert(Q,R,j,x,'col') [Q1,R1] = qrinsert(Q,R,j,x,'row')</pre> <p>The original syntaxes <code>qrdelete(Q,R,j)</code> and <code>qrinsert(Q,R,j,x)</code> default to 'col'.</p> | | |

| Function | Enhancement or Change (Continued) | |
|----------|---|--|
| spparms | Provides two new parameters for sparse matrix division. | |
| | 'piv_tol' | Pivot tolerance used by the UMFPACK LU-based \ and /. |
| | 'bandden' | Band density used by LAPACK-based \ and / for banded matrices. |

Programming and Data Types Features

MATLAB 6.5 adds the following programming and data types features and enhancements:

- “JIT Accelerator with MATLAB” on page 7-21
- “Regular Expression Support” on page 7-22
- “New Functions” on page 7-22
- “Cell to Matrix Conversion Functions” on page 7-23
- “New Warning and Error Handling Features” on page 7-24
- “Dynamic Field Names for Structures” on page 7-25
- “New Logical AND and OR Operators for Short-Circuiting” on page 7-26
- “New Output from ismember” on page 7-26
- “New true and false Functions” on page 7-27
- “Interrupting try-catch in a Loop” on page 7-27
- “Changes to copyfile and mkdir” on page 7-28
- “mfilename Returns Path and Class Information” on page 7-28
- “Support for the 64-Bit Integers int64 and uint64” on page 7-28
- “64-Bit File Handling” on page 7-29
- “MATLAB Audio Enhancements” on page 7-29
- “New MATLAB Timer Object” on page 7-30

JIT Accelerator with MATLAB

MATLAB 6.5 includes significant changes in the way MATLAB processes M-file functions and scripts. These changes affect the performance of MATLAB

and can give you a substantial performance increase over earlier MATLAB versions for many MATLAB applications.

Speeding up the execution of programs written in MATLAB is an ongoing MathWorks endeavor that will be delivered over a number of product releases. The documentation on “Performance Acceleration” explains how to best make use of the JIT Accelerator, how to use the MATLAB Profiler to optimize your performance, and includes several sample programs to illustrate how you can make your M-file programs run faster.

Regular Expression Support

MATLAB now supports searching and replacing characters using regular expressions. The following new functions support this capability. For more information, see “Regular Expressions” in the MATLAB documentation.

| Function | Description |
|------------------------|---|
| <code>regexp</code> | Match regular expression. |
| <code>regexpi</code> | Match regular expressions, ignoring case. |
| <code>regexprep</code> | Replace string using regular expression. |

New Functions

MATLAB 6.5 added new functions for working with error generation, regular expressions, sorted sets, integer conversion, and for performing file operations.

| Function | Description |
|-----------------------------------|---|
| <code>false</code> | False array |
| <code>fileattrib</code> | Set or get attributes of file or directory |
| <code>int64</code> | Convert to signed 64-bit integer |
| <code>isequalwithequalnans</code> | Determine if arrays are numerically equal, treating NaNs as equal |
| <code>issorted</code> | Determine if set elements are in sorted order |

| Function | Description (Continued) |
|-----------------|--|
| lasterror | Return last error message and related information |
| movefile | Move file or directory |
| orderfields | Order fields of a structure array |
| perl | Call Perl script using appropriate operating system executable |
| rethrow | Reissue error |
| rmdir | Remove directory |
| true | True array |
| uint64 | Convert to unsigned 64-bit integer |
| winopen | Open file in appropriate application (Windows only) |
| xmlread | Parse XML document and return Document Object Model node |
| xmlwrite | Serialize XML Document Object Model node |
| xslt | Transform XML document using XSLT engine |

Cell to Matrix Conversion Functions

The following functions, previously belonging to the Neural Network Toolbox, are now core MATLAB functions.

| Function | Description |
|-----------------|--|
| cell2mat | Combine a cell array of matrices into one matrix |
| mat2cell | Break matrix up into a cell array of matrices |

New Warning and Error Handling Features

These features include:

- “Formatted Error and Warning Strings”
- “Message Identifiers”
- “Warning Control Features”

Formatted Error and Warning Strings. Prior to MATLAB 6.5, the error and warning functions only accepted a simple string as an input argument, as shown here for error:

```
error('error_string')
```

In MATLAB 6.5, error and warning now accept a format string and one or more parameters, using a syntax similar to the MATLAB `sprintf` function. The syntax for error is shown here. Use the same syntax for warning:

```
error('format-string', arg1, arg2, ...)
```

Examples for using this syntax with error and warning are

```
error('File %s not found', filename);  
warning('Ambiguous parameter name, "%s".', param)
```

Message Identifiers. A message identifier is a tag that you attach to an error or warning statement that makes that error or warning uniquely recognizable by MATLAB. You can use message identifiers with warnings to control any selected subset of the warnings in your programs, or with error reporting to better identify the source of an error.

Some examples of message identifiers are

```
MATLAB:divideByZero  
Simulink:actionNotTaken  
TechCorp:notFoundInPath
```

Message identifiers are used in warning control (see next section) and also to enable the `lasterr` and `lasterror` functions to better identify the source of an error.

See “Using Message Identifiers with `lasterr`” in the MATLAB documentation.

Warning Control Features. In this release, MATLAB gives you the ability to control what happens when a warning is encountered during M-file program execution. New options available in this release include

- Display selected warnings
- Ignore selected warnings

Depending on how you set up your warning controls, you can have these actions affect all warnings in your code, specific warnings that you select, or just the most recently invoked warning. See the section “Warning Control” in the MATLAB documentation for more information on this feature.

Also read the section, “Warning Control Upgrade Issues” on page 7-65 in these Release Notes to see how your existing programs might be affected by this change.

Dynamic Field Names for Structures

You can now reference structures using field names that are computed at run-time using the new *dynamic field names* feature in MATLAB. The dot-parentheses syntax shown below tells MATLAB to interpret expression as a dynamic field name:

```
structure_name.(expression)
```

This added capability now completes the following table by providing full dynamic access to all data types.

| Data Type | Static (compile time) | Dynamic (run-time) |
|------------|-----------------------|--------------------|
| Matrix | A(2,3) | A(m,n) |
| Cell Array | C{4} | C{k*2} |
| Structure | S.name | S.(field) |

In many cases, you can use dynamic field names in place of the `getfield` and `setfield` functions. Dynamic field names offer improved execution speed and code readability. Compare the following for readability:

```
S(m,n).(fieldname)(k) = value
```

```
S = setfield(S,{m,n},fieldname,{k},value)
```

Technical Note 32236 provides more information about using the `getfield` and `setfield` functions versus dynamic field names. See “Dynamic Field Names” in the MATLAB documentation for more information on this feature.

New Logical AND and OR Operators for Short-Circuiting

Prior to this release, the MATLAB `&` (AND) and `|` (OR) operators served two purposes: that of a logical operator and also an array operator. These two roles at times conflicted, resulting in technically correct, yet possibly confusing evaluations.

MATLAB 6.5 introduces two additional AND and OR operators: `&&` and `||`. Use these new operators to evaluate a compound logical expression, especially when short-circuiting is required. Use the `&` and `|` operators for element-wise operations on arrays.

See “Short-circuit Operators” in the MATLAB documentation for a discussion on short-circuiting with `&&` and `||`.

New Output from `ismember`

The `ismember` function now returns an optional, second output indicating the indices at which members of a set are located. The syntax is

```
[tf, loc] = ismember(A,S,...)
```

When you use this syntax, `ismember` returns index vector `loc` containing the highest index in `S` for each element in `A` that is a member of `S`. For those elements of `A` that do not occur in `S`, `ismember` returns 0.

For example,

```
a = reshape(1:5, [5 1])
set = [5 2 4 2 8 10 12 2 16 18 20 3];
[tf, index] = ismember(a, set);
```

```
index
index =
     0
     8
    12
     3
     1
```

New true and false Functions

MATLAB has two new functions for logical operations: `true` and `false`. `true` is shorthand for `logical(1)` and `false` for `logical(0)`. Both functions accept input arguments that enable you to build n-dimensional arrays of logical 1 or 0.

This example builds a 3-by-5 array of type logical:

```
a = true(3,5)
a =
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

`true(n)` is equivalent to `logical(ones(n))`, however `true(n)` is easier to use and will improve the readability of your program. Type `help true` or `help false` for more information.

Interrupting try-catch in a Loop

try-catch statements no longer catch **Ctl+C** interrupts. Prior to this release, pressing **Ctl+C** while a try block was executing would result in a jump to the corresponding catch block. In MATLAB 6.5, a **Ctl+C** aborts execution and returns control to the Command Window, regardless of what code is executing.

For example, in MATLAB 6.5, you can use **Ctl+C** to interrupt the loop shown here. You could not interrupt this loop in earlier MATLAB versions:

```
for k=1:100
    try
        pause(1);
    catch
    end
end
```

Changes to copyfile and mkdir

The `copyfile` and `mkdir` functions have changed since the previous release.

Changes to copyfile. The `writable` argument has been superseded by the `f` argument, although `writable` is still allowed for this release. The function now also copies directories. It replaces the destination files or directories of the same name as the source files or directories without a warning. (In previous versions, there was a warning in that event.) If the destination files or directories are read-only and the `f` (or `writable`) argument is not used, `copyfile` will fail.

Change to mkdir Return Status. `mkdir` no longer returns 2 if the directory already exists, but instead displays a warning. It also has an enhanced return format.

mfilename Returns Path and Class Information

You can now request more information from the `mfilename` function:

- `mfilename('fullpath')` — Returns the full path and name of the M-file in which the call occurs, not including the filename extension.
- `mfilename('class')` — In a method, returns the class of the method, not including the leading `@` sign. If called from a non-method, it yields the empty string.

Support for the 64-Bit Integers int64 and uint64

MATLAB now supports signed and unsigned 64-bit integers. Use the `int64` and `uint64` functions to convert a number to a signed or unsigned 64-bit integer.

64-Bit File Handling

MATLAB low-level file handling functions (`fopen`, `fseek`, `ftell`, etc.) now support 64-bit file offsets. This enables you to perform low-level I/O operations on files greater than 2 GB in size. (The limit in previous versions of MATLAB was $2^{31} - 1$ bytes, or 2 GB.)

64-bit support is available on the following platforms:

- Windows
- Solaris
- MacIntosh
- Alpha
- HP/UX 11.0, 9000/785

64-bit support is *not* available on the following platforms, due to limitations imposed by their respective operating systems:

- SGI
- Linux
- HP/UX 10.20, 9000/735
- HP/UX 11.0, 9000/780

On the IBM-AIX platform, 64-bit file I/O is supported for reading only. You can write only up to 2 GB.

MATLAB Audio Enhancements

New `audiodevinfo` Function. On Windows 32-bit machines, `audiodevinfo` returns information about installed audio devices.

UNIX Platform Support for `audiorecorder` and `audioplayer`. `audioplayer` and `audiorecorder` are now available on UNIX platforms running Java Runtime Environment 1.3 or higher.

Enhancements to `audiorecorder` and `audioplayer`. `audioplayer` and `audiorecorder` can now take the audio device ID as input. You can obtain the device ID from `audiodevinfo`.

`audioplayer` can now take an `audiorecorder` as an input.

Support for 24-bit Recording and Playback. On 32-bit Windows machines with an installed 24-bit audio device, audiorecorder and audioplayer now support 24-bit recording and playback, respectively.

Improvement to wavread and wavwrite. wavread and wavwrite now support reading and writing 24- and 32-bit .wav files.

Workspace Browser Support. Right-clicking on an audio object in the Workspace Browser now displays a context menu with player/recorder controls.

New MATLAB Timer Object

MATLAB includes a timer object that you can use to schedule the execution of MATLAB commands. To use a timer, you must perform these steps:

- 1** Create a timer object by calling the timer function.
- 2** Specify which MATLAB commands you want executed and when you want them executed by setting timer object properties. (You can also set timer object properties when you create them, in Step 1.)
- 3** Start the timer by calling the start or startat functions.

Programming Tips Documentation

A number of questions come up repeatedly in our external customer newsgroup and our technical support Web site. Some of these questions arise when MATLAB users are unable to find the information they are looking for in the documentation. The “MATLAB Programming Tips” documentation is a new feature in MATLAB 6.5, designed to make it easier to find help on a wide range of topics. Many of the questions addressed by the tips documentation were taken from discussions in the MathWorks newsgroup and from the technical support site.

“MATLAB Programming Tips” is a chapter in the MATLAB “Programming and Data Types” documentation. It is a categorized compilation of tips, covering topics such as Debugging, Input/Output, Managing Memory, Optimizing for Speed, etc. Each item is relatively brief to help you to browse through them and find information that will be useful. Many of the tips include a link into the MATLAB documentation to give you more complete coverage of the topic.

Graphics Features

MATLAB 6.5 adds the following graphics features and enhancements.

New Text Properties – Control Text Background

Text objects have the following new properties.

| Property | Purpose |
|-----------------|--|
| BackgroundColor | Color of text extent rectangle |
| EdgeColor | Color of the rectangle edge |
| LineStyle | Style of the rectangle edge line |
| LineWidth | Width of the rectangle edge line |
| Margin | Increase the size of the rectangle by adding a margin to the text extent |

Colormap Editor – Modify Colormaps Interactively

The colormap editor is a tool that enables you to modify the colormap of the current figure. See the `colormapeditor` function description for more information and an example.

Redesigned Property Editor

The Handle Graphics® Property Editor and associated help have been redesigned.

Selecting a Printer From the MATLAB Command Line

In earlier versions of MATLAB, you could select a nondefault printer for graphics from the MATLAB command line on UNIX systems only. In MATLAB 6.5, you can do this on Windows systems as well. Specify the printer using the `-P` switch in the print command.

For example, to print Figure No. 3 to a printer called Calliope, type

```
print -f3 -PCalliope
```

If the printer name has spaces in it, put quotes around the -P option, as shown here.

```
print -f3 '-Pmy local printer'
```

Using a Network Print Server

On Windows NT, Windows 2000, and Windows XP systems, you can print to a network print server using the form shown here for a printer named trinity.

```
print -P\\PRINTERS\trinity
```

External Interfaces/API Features

MATLAB 6.5 adds the following external interface and API features and enhancements:

- “New MX and MEX Functions” on page 7-32
- “New COM Client Support Features” on page 7-34

New MX and MEX Functions

There are a number of new logical mx functions provided as part of changing logical from an attribute to a MATLAB class, several additional mx functions, and two new mex error handling functions.

New Logical Functions. This release introduces seven new C mx functions to use with logicals.

| Function | Description |
|-----------------------------|--|
| mxCreateLogicalArray | Create N-dimensional, logical mxArray initialized to false |
| mxCreateLogicalMatrix | Create two-dimensional, logical mxArray initialized to false |
| mxCreateLogicalScalar | Create scalar, logical mxArray initialized to false |
| mxCreateSparseLogicalMatrix | Create unpopulated, two-dimensional, sparse, logical mxArray |

| Function | Description (Continued) |
|------------------------------------|--|
| <code>mxGetLogicals</code> | Get pointer to logical array data |
| <code>mxIsLogicalScalar</code> | True if scalar mxArray of class <code>mxLOGICAL</code> |
| <code>mxIsLogicalScalarTrue</code> | True if scalar mxArray of class <code>mxLOGICAL</code> is true |

Obsolete Logical Functions. The following two functions are now obsolete. Support for these functions will be removed in a future release.

| Function | Description |
|-----------------------------|---------------------------------|
| <code>mxClearLogical</code> | Convert mxArray to numeric type |
| <code>mxSetLogical</code> | Convert mxArray to logical type |

New mx Functions in C API. MATLAB 6.5 also introduces these new C mx functions.

| Function | Description |
|-----------------------------------|--|
| <code>mxGetChars</code> | Get pointer to character array data |
| <code>mxCreateDoubleScalar</code> | Create scalar, double-precision array initialized to the specified value |

Note `mxCreateDoubleScalar` replaces `mxCreateScalarDouble`, although the latter function is still supported at this time.

New mex Functions for Error Handling. There are two new C and Fortran MEX functions that enable you to specify a message identifier and message string when reporting an error or warning. These functions also accept formatting

conversion characters, such as those used with the MATLAB `sprintf` function, in the error or warning message string.

| Function | Description |
|---------------------------------|---|
| <code>mexErrMsgIdAndTxt</code> | Issue error message with identifier and return to MATLAB prompt |
| <code>mexWarnMsgIdAndTxt</code> | Issue warning message with identifier |

New COM Client Support Features

In an effort to provide a consistent interface to object-oriented technologies, MATLAB 6.5 introduces several changes that affect the way you interact with Component Object Model (COM) controls and servers through MATLAB.

Key benefits of this change are

- Robust memory management — Objects and interfaces are destroyed automatically when the variable that represents the object or interface is either reassigned or goes out of scope.
- Flexibility in event handling — Register and unregister a control's events with callback or event handler routines at any time after the control has been created.
- Custom properties — You can attach your own properties to a control and store any kind of data in the control.
- A graphical user interface for viewing and modifying COM properties.
- Multiple arguments with the `set` function.
- More useful information on methods returned by `invoke`.
- More detail in error messages.

See “Client Support for COM” in the MATLAB documentation for more information on these features. You may also want to read through the “External Interfaces/API Upgrade Issues” on page 7-76 to find out how these changes may affect your existing programs.

COM Demo. MATLAB includes three demos showing how to use the COM client interface. To run any of the demos, click on the **Demos** tab in the MATLAB Help Browser. Then click to expand the folder called Automation Client Interface (COM).

New MATLAB Functions for COM. There are a number of new functions available for the COM interface.

| Function | Description |
|---------------------|--|
| addproperty | Add custom property to COM object |
| deleteproperty | Remove custom property from COM object |
| eventlisteners | Return a list of events attached to listeners |
| events | Return a list of events that the control can trigger |
| isevent | Determine if an item is an event of a COM control |
| ismethod | Determine if an item is a method of a COM object |
| isprop | Determine if an item is a property of a COM object |
| registerevent | Register an event handler with a control's event |
| unregisterallevents | Unregister all events for a control |
| unregisterevent | Unregister an event handler with a control's event |

MATLAB Functions New to COM. You can now use these MATLAB functions in the COM environment.

| Function | Description |
|-----------------|--|
| fieldnames | Return property names of a COM object |
| inspect | Display graphical interface to list and modify property values |
| methods | List all methods for the control or server |
| methodsview | Display graphical interface to list method information |

Specifying Property Names. You may abbreviate the names of properties, as long as you include enough letters in the name to make it unambiguous. Also,

property names are not case-sensitive. For example, to get the value of the `OrganizationName` property from a COM server running an Excel application, you can use

```
get(h, 'org')
ans =
    The MathWorks, Inc.
```

Get and Set on Multiple Objects. You can use the `get` and `set` functions on more than one object at a time by putting the object handles into a vector and then operating on the vector. See “Get and Set on Multiple Objects” in the MATLAB documentation.

Enumerated Values for COM Properties. When setting the value for a COM property in MATLAB, you can now use an enumerated string in place of a numeric value. An enumerated string, such as `xlUnicodeText`, is much easier to remember than its equivalent numeric value, and thus makes it unnecessary to spend time looking up valid settings for a property.

To list all possible enumerated values for a property of object `h`, use

```
set(handle, 'propertyname')
```

To set a property to the value represented by an enumerated string, use this syntax. The `enumstring` argument can be abbreviated, as long as you use enough letters to make it unambiguous:

```
set(handle, 'propertyname', 'enumstring')
```

To get the current enumerated value of a property, use

```
get(handle, 'propertyname')
```

See “Using Enumerated Values for Properties” in the MATLAB documentation for more information.

Custom Properties. You can attach your own properties to a control using the `addproperty` function. The syntax shown here creates a custom property for control, `h`:

```
addproperty(handle, 'propertyname')
```

This example creates the `mwsamp` control, adds a new property called `Position` to it, and assigns the value `[200 120]` to that property:


```
h = actxcontrol('mwsamp.mwsampctrl.2', [200 120 200 200]);
addproperty(h, 'Position');
set(h, 'Position', [200 120]);
```

To remove custom properties from a control, use `deleteproperty` with the following syntax:

```
deleteproperty(h, 'propertyname')
```

See “Custom Properties” in the MATLAB documentation for more information.

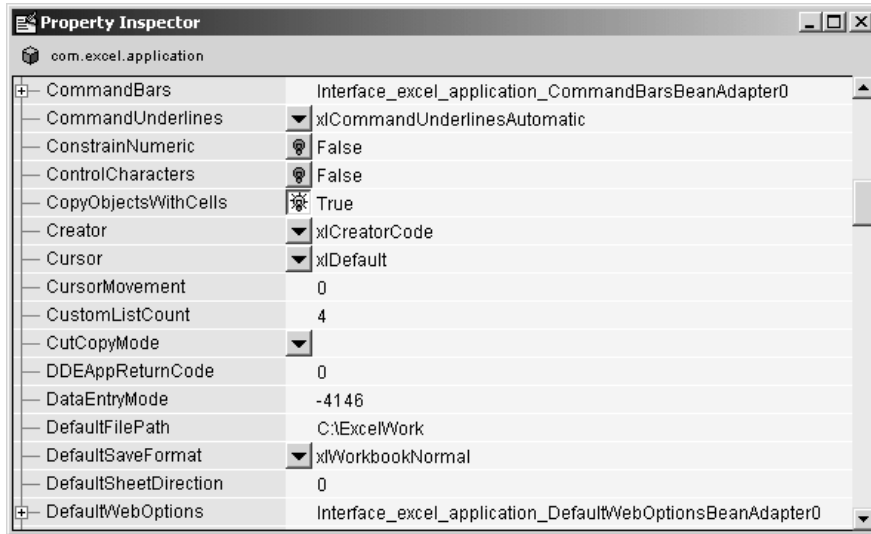
New Event Handling Functions. With earlier versions of MATLAB, you could register events for a control only at the time the control was created. In MATLAB 6.5, you can register and unregister events at any time using the `registerevent`, `unregisterevent`, and `unregisterallevents` functions.

You can also list all events that a control can respond to, or just those events that are currently registered, using the `events` and `eventlisteners` functions, respectively. The `events` function supersedes the COM `send` function.

See “COM Control Events” in the MATLAB documentation for more information on event handling.

GUI Interface to Get and Set Properties. Use the new `inspect` function to see a list of all properties belonging to a COM object or interface. Create an Excel server object and invoke `inspect` to bring up the Property Inspector window shown below:

```
h = actxserver('excel.application');
inspect(h)
```



To change the value of one of the properties, click on the property name at the left and then type in the new value in the field at the right.

See “Using the Property Inspector” in the MATLAB documentation.

set Accepts Multiple Arguments. You can now set more than one property value with one set command. The syntax is

```
set(h, property1, newvalue1, property2, newvalue2, ...);
```

Each property argument must be followed by a newvalue argument. The example shown here changes the Label and Radius for an mwsamp control:

```
h = actxcontrol('mwsamp.mwsampctr1.2', [0 0 200 200]);
get(h)
ans =
    Label: 'Label'
    Radius: 20

set(h, 'label', 'Hello', 'radius', 35);
```

```

get(h)
ans =
    Label: 'Hello'
    Radius: 35

```

Returning More Than One Output Argument. A MATLAB client can now return more than one output argument from COM server applications. If you know that a server function supports multiple outputs, you can return any or all of those outputs to the MATLAB client.

With previous versions of MATLAB, you could only get back a single return value (shown here as `ret`) from a function call to the server, even for those functions that could return more than one output value:

```
ret = functionname(in1, in2, ...);
```

In MATLAB 6.5, you can specify additional output arguments (shown here as `out1 out2 ...`) in a function call, enabling the client access to all values returned by the function:

```
[ret out1 out2 ...] = functionname(in1, in2, ...);
```

If there are multiple output arguments, the return value is always the first argument on the left hand side (lhs).

MATLAB makes use of the pass by reference capabilities in COM to implement this feature. Note that pass by reference is a COM feature. It is not available in MATLAB at this time.

Argument Types Listed By Invoke. The `invoke` function now lists data types for input and output arguments. The function `m1` defined in an Interface Definition Language (IDL) file is shown here:

```
m1([in,out] BSTR* strInOut, [out] short* shortOut, [in] long
longIn, [out,retval] double* doubleRet);
```

MATLAB 6.1 `invoke` lists the function as

```
m1 = double m1(Variant(Pointer), Variant(Pointer), Int)
```

while MATLAB 6.5 `invoke` lists it as

```
m1 = [double, string, int16] m1(handle, string, int32)
```

More Detail in Error Messages. MATLAB now returns more detailed information when a COM error is generated. This includes the source and description of the error, along with the location of help resources provided to assist in resolving the error:

```
h = actxserver('excel.application');
Repeat(h)
???: Invoke Error, Dispatch Exception:
Source: Microsoft Excel
Description: Repeat method of Application class failed
Help File: D:\Applications\MSOffice\Office\1033\xlmain9.chm
Help Context ID: 0.
```

Creating Graphical User Interfaces (GUIDE) Features

MATLAB 6.5 adds the following features and enhancements to GUIDE:

- New structure for the generated M-file makes it easier to understand and program
- New GUIDE Quick Start dialog and GUI templates. When you first open GUIDE, the GUIDE Quick Start dialog box provides access to new GUI templates—simple examples of GUIs that you can modify for your own purposes.
- Tab Order Editor enables you to change the order in which GUI components are selected when a user clicks the **Tab** button.
- Changes to component tags automatically update callbacks and M-file code
- **Export** option in the **File** menu enables you to export a GUI to a single M-file that does not require a FIG-file.
- MATLAB Editor icon on the toolbar provides easier access to the editor.

Changes to the M-file Generated by GUIDE

The associated M-file generated by GUIDE has the following differences for Release 13:

- Most of the GUI initialization is performed in a separate function, which you do not need to edit.
- The GUI M-file contains an opening function, where you can add code to create data or perform other tasks before the GUI becomes visible to the user.

- The GUI M-file contains an output function for returning variables to the command line.
- A new calling syntax simplifies passing user defined arguments to the GUI M-file.
- You can pass figure property name/value pairs as arguments when creating the GUI.
- Generated callback function stubs no longer include a `varargin` argument. If you want to add more arguments to a callback subfunction, you must add the arguments to the function definition.

The following sections describe changes to the associated M-file in greater detail.

New Calling Syntax for GUI M-File. You can call the GUI M-file with the following syntax:

```
my_gui
my_gui('PropertyName', PropertyValue, ...)
my_gui(UserArgs, ...)
my_gui('PropertyName', PropertyValue, ..., UserArgs, ...)
```

- `my_gui` without arguments starts the GUI.
- Calling `my_gui('Property', Value, ...)`, where 'Property' is a valid figure property, creates a new `my_gui` using the given property value pairs.
- Calling `my_gui('My_function', hObject, eventdata, handles)` calls the subfunction `my_function` in the GUI M-file with the given input arguments.

Opening Function Code. The generated GUI M-file now includes a subfunction for any initialization code you want to execute. If you call the GUI with input arguments, they are passed to the opening function. The GUI M-file calls the opening function with the following arguments:

```
function myGUI_OpeningFcn(hObject, eventdata, handles, varargin)
```

- `hObject`—handle to figure
- `eventdata`—to be defined in a future version of MATLAB
- `handles`—structure with handles and user data (see `guidata`)
- `varargin`— command line arguments to `my_gui` (see `varargin`)

Output Function Code. The GUI M-file now includes a subfunction for passing output arguments to the command line. The GUI M-file calls the output function with the following arguments:

```
function varargout = myGUI_OutputFcn(hObject, eventdata, handles)
```

- `hObject`—handle to figure
- `eventdata`—to be defined in a future version of MATLAB
- `handles`—structure with handles and user data (see `guidata`)
- `varargin`—unrecognized property name/value pairs from the command line

GUIDE Quick Start Dialog and Templates

GUIDE now provides four templates that make it easier to construct GUIs. The templates are simple examples of GUIs that you can modify for your purposes. You can access the templates from the new **GUIDE Quick Start** dialog that appears when you open GUIDE, or when you select **New** from the file menu. It is often easier to build a GUI from an existing template rather than starting with a blank GUI.

openfig Accepts Property Name/Value Pair—Returns User Args

The `openfig` function enables you to specify figure property name/value pairs that are applied to the figure before it is displayed. See the `openfig` reference page for more information.

uiputfile and uigetfile Return Filter Index

The `uigetfile` and `uiputfile` functions can now optionally return an index value that enables you to determine which filter was selected by the user. See the `uigetfile` and `uiputfile` reference pages for more information.

uigetdir

The new `uigetdir` function displays a dialog box in which the user can select a directory, and returns the directory name as a string.

Major Bug Fixes

MATLAB 6.5 includes several bug fixes made since the last MATLAB release. This section describes the particularly important Version 6.5 bug fixes.

If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

If you are upgrading from a release earlier than Release 12.1, then you should also see “Major Bug Fixes” on page 8-18 in the MATLAB 6.1 Release Notes.

Platform Limitations

The MATLAB functionality described in these Release Notes and in the MATLAB documentation applies to MATLAB 6.5, with the exception of the limitations listed below for the HP and IBM platform.

This discussion of new MATLAB platform limitations is organized into the following categories:

- “Patch Required for HP-UX 11.0” on page 7-44
- “Development Environment Limitations” on page 7-44
- “Mathematics Limitations” on page 7-46
- “Graphics Limitations” on page 7-47
- “Creating Graphical User Interfaces (GUIDE) Limitations” on page 7-47

Another platform limitation involves the use BLAS on certain processors. See “You May Need to Overwrite the MATLAB Default Choice of BLAS” on page 7-47 for details.

Patch Required for HP-UX 11.0

To run MATLAB on HP-UX 11.0, you must install a patch available from Hewlett-Packard. To get the patch, go to www.itrc.hp.com, the IT Resource Center page. The patch is available to registered customers from the individual patches link. The patch name is below.

PHSS_21959 1.0 X/Motif 32 bit
Runtime 2000 Periodic Patch

Development Environment Limitations

The MATLAB 6.5 development environment features have the platform limitations described below. These include limitations that have existed since MATLAB 6.0.

The MATLAB desktop and most of the development environment tools are not available on the HP-UX and IBM platform. Following are the specific limitations for each tool and available alternatives.

| Feature | Limitation and Alternatives |
|---------------------------|---|
| Desktop | Not supported. Instead, the MATLAB prompt appears in an X window. Use function alternatives for various tools. |
| Array Editor | Not supported. Instead, view and edit variables at the command line. |
| Command History | Not supported. To recall previous lines, use the up arrow key, or use the <code>diary</code> function or the <code>logfile</code> startup option. |
| Current Directory browser | Not supported. Use function alternatives documented for the Current Directory browser, including <code>cd</code> , <code>delete</code> , <code>ls</code> , and <code>mkdir</code> . |
| Demos | Demos for non-Java platforms run the way they did in Release 12.1. You do not access them from the Help browser, but rather by using the <code>demo</code> function. |
| Editor/Debugger | Not supported. For editing M-files, use the <code>edit</code> function, and another text editor, such as Emacs—see the <code>edit</code> reference page to specify the other text editor. To debug M-files, use MATLAB debugging functions. |
| Help browser | Not supported. Help displays in your default browser. The Index and Search features are not available. You get a broken link message from your browser if you try to access documentation that you do not have installed. |
| HDF Import Tool | Not supported. Use function alternatives documented in Using the HDF Import Tool. |
| Import Wizard | Not supported. Use <code>import</code> function equivalents for the various features. |

| Feature | Limitation and Alternatives (Continued) |
|---------------------------|---|
| Launch Pad | Not supported. Access documentation and demos using functions, such as <code>help</code> and <code>demo</code> . |
| Preferences | Not supported. Set location of help files using <code>docopt</code> . |
| Profiler | The new desktop Profiler is not supported. The Version 6.1 <code>profile</code> and <code>profreport</code> functions are supported. |
| Set Path dialog box | Not supported. Use the <code>path</code> , <code>addpath</code> , and <code>rmpath</code> functions instead. |
| Source Control menu items | Not supported. Use the <code>checkin</code> , <code>checkout</code> , <code>cmopts</code> , and <code>undocheckout</code> functions on UNIX platforms or the <code>verctrl</code> function on PC platforms instead. |
| Workspace browser | Not supported. Use <code>who</code> , <code>whos</code> , <code>save</code> , <code>load</code> , and <code>clear</code> functions instead. |

Mathematics Limitations

The MATLAB 6.5 mathematics features have the platform limitation described below.

Basic Fitting Interface

The Basic Fitting interface is not supported. Instead, use curve fitting functions such as `polyfit` and `spline`. See also “Data Analysis and Statistics” in the MATLAB documentation for more information.

Graphics Limitations

The MATLAB 6.5 graphics features have the platform limitations described below.

| Feature | Limitation and Alternatives |
|-----------------|--|
| Data Statistics | Not supported. |
| Printing | Uses the Release 11 Page Setup , Print Setup , and Print dialog boxes. For information about these interfaces, see “Printing MATLAB Graphics” in the online MATLAB documentation. |
| Property Editor | Not supported. Similar graphical user interfaces provide access to figure, line and text objects. Use the set and get functions to modify Handle Graphics object properties. |

Creating Graphical User Interfaces (GUIDE) Limitations

The MATLAB 6.5 GUIDE-related features have the platform limitations described below.

GUIDE is not supported on the following platforms:

- IBM_RS
- HPUX
- HP700

You May Need to Overwrite the MATLAB Default Choice of BLAS

On the PC, under both Linux and Windows operating systems, MATLAB determines at startup time what processor your computer has, for example Genuine Intel Pentium II, Pentium III, or AMD Athlon. MATLAB then automatically selects the most appropriate BLAS for your processor. The same is true on the SUN, where MATLAB distinguishes between UltraSPARCs and non-Ultra machines.

However, on the remaining platforms you get the default BLAS, which is usually targeted for a reasonably modern or common processor:

- ALPHA 21264
- HP700 PA-RISC1.1
- HPUX PA-RISC2.0
- IBM_RS Power3
- SGI R12000

If you have reason to believe that your processor is closer to another of the flavors of BLAS distributed with MATLAB, for example 21164 on the ALPHA or PA-RISC2.0 on the HP700, you might want to override the default choice of BLAS. Look in your <MATLAB>/bin/\$ARCH directory for libraries beginning with atlas_ to see your options.

Overriding the Default

The way to override the default choice is to set the environment variable BLAS_VERSION before invoking MATLAB. For example (in csh):

```
setenv BLAS_VERSION atlas_21164.so
setenv LAPACK_VERBOSITY 1
matlab
```

The environment variable LAPACK_VERBOSITY simply confirms that your choice of BLAS is being loaded once you start up MATLAB.

Restoring the Default

If you would like to return to using the default provided by MATLAB, you may use the command (in csh)

```
unsetenv BLAS_VERSION
```

Upgrading from an Earlier Release

This section describes the upgrade issues involved in moving from MATLAB 6.1 to Version 6.5. This discussion of new MATLAB upgrade issues is organized into the following categories:

- “Development Environment Upgrade Issues” on page 7-49
- “Mathematics Upgrade Issues” on page 7-51
- “Programming and Data Types Upgrade Issues” on page 7-52
- “Graphics Upgrade Issues” on page 7-75
- “External Interfaces/API Upgrade Issues” on page 7-76
- “Creating Graphical User Interfaces (GUIDE) Upgrade Issues” on page 7-85

If you are upgrading from a release earlier than Release 12.1, then you should see “Upgrading from an Earlier Release” on page 8-23 in the MATLAB 6.1 Release Notes.

Development Environment Upgrade Issues

The issues involved in upgrading from MATLAB 6.1 to MATLAB 6.5, in terms of development environment features, are discussed below.

Toolbox Path Caching Now On By Default

Toolbox path caching is now on by default—see “Startup” on page 7-3.

Release 13 Prerelease users might not see the toolbox path caching option on by default. To turn it on, select **File -> Preferences -> General**, set the **Enable toolbox path cache** check box, and click **OK**. The next time you start MATLAB, it will create the cache file, and startups after that will be faster.

Changes to ver Function

The ver function header now displays more detailed operating system output and the version, if any, of the Java Virtual Machine MATLAB uses. The `hostid` is no longer in the ver header.

The ver header displays when you run ver with an argument, for example, `ver('simulink')`. The header is not displayed when ver returns the results to a structure, for example, `simver = ver('simulink')`.

The `ver` output no longer includes a date column. The output is now ordered with MATLAB first, Simulink second, if installed, and then all other installed products in alphabetical order.


Migration of Files Used by Desktop Tools

Most files associated with desktop tools are maintained when you upgrade from Release 12.1 to Release 13. Specifically, preferences, the Command History, Help favorites, and current directory entries in the desktop toolbar and Current Directory browser lists are maintained. However, there may be some invalid current directory and favorites entries if the locations of Release 13 files are different from the locations of Release 12 files.

pathdef.m. If you want Release 13 to use your existing `pathdef.m` file, save it to another location outside of `$matlabroot` before installing Release 13, and then after installing, copy it back.

Editor/Debugger

Cannot Save in Debug Mode. You cannot save changes to an M-file while in debug mode. First quit debug mode and then save the file.

Subfunctions Listed Alphabetically. When you click the function button  on the toolbar, the subfunctions are listed alphabetically. Previously they were listed in the order that they appeared in the M-file.

Use Delete Instead of Clear. The **Edit -> Clear** menu item was removed. Use **Edit -> Delete** instead.

Discontinued Form of edit. The `edit` function no longer supports the forms `edit fun1 in fun2` or `edit fun(a, b, c)`.

Running Playshow Demos from the Command Line

To run playshow demos from the command line, you now need to type `playshow` followed by the demo name. In previous releases, you only needed to type the playshow demo name to run it.

For example, if you type `quake`, the demo does not run. View the H1 line for `quake.m`, that is, the first comment line. It begins with two comment symbols (`%`), indicating that `quake` is a playshow demo. Therefore, type `playshow quake` to run the demo.

Mathematics Upgrade Issues

The issues involved in upgrading from MATLAB 6.1 to MATLAB 6.5, in terms of mathematics features, are discussed below.

Singular Triangular Matrix Division

The result of dividing a singular lower or upper triangular matrix by any other matrix, using either left (`\`) or right (`/`) division may change. Previously, for singular square matrices A for which `rcond(A) = 0`, the result was always a matrix of `Infs`.

This change is a result of performance improvements described in “Mathematics Features” on page 7-15.

Example 1.

In MATLAB Version 6.5,

```
A = [1 2 3;0 4 5;0 0 0];
b = [1;2;3];
A\b
Warning: Matrix is close to singular or badly scaled.
        Results may be inaccurate. RCOND = 0.000000e+000.

ans =
     NaN
    -Inf
     Inf
```

Previously, the result was

```
[ Inf
  Inf
  Inf]
```

Example 2.

In MATLAB 6.5, a zero matrix is treated as a singular triangular matrix.

```
[0 0;0 0] \ [0 0]'
```

```
Warning: Matrix is close to singular or badly scaled.  
Results may be inaccurate. RCOND = 0.000000e+000.
```

```
ans =  
    NaN  
    NaN
```

Previously the result was

```
[ Inf  
  Inf]
```

Programming and Data Types Upgrade Issues

The issues involved in upgrading from MATLAB 6.1 to MATLAB 6.5, in terms of programming and data types features, are discussed below. MATLAB 6.5 introduces important changes to the inner structure of MATLAB that may affect existing programs. These changes are

- “Maximum Name Length Changed for Variables, Functions, Files” on page 7-53
- “The logical Attribute Is Now a Class” on page 7-55
- “Changes to Definition of “Truth”” on page 7-59
- “The sparse Class Is Now an Attribute” on page 7-61
- “Logical Indexing and find” on page 7-65
- “Warning Control Upgrade Issues” on page 7-65
- “Formatted Error and Warning Strings” on page 7-68
- “getfield and setfield Superseded” on page 7-68
- “New Behavior in break” on page 7-68
- “isequal and Structure Field Creation Order” on page 7-68
- “Set Operations on Cell Arrays of Strings” on page 7-68
- “Using mat2cell on an Empty Array” on page 7-69
- “Concatenating with Empty Arrays” on page 7-69
- “Function Names Redefined As Variable Names” on page 7-70

- “Specifying More Outputs Than a Function Defines” on page 7-71
- “Consistent Handling of Subscripting Errors” on page 7-72
- “Consistent Handling of Logical Errors” on page 7-72
- “Operations That Are Now Considered Invalid” on page 7-73

Maximum Name Length Changed for Variables, Functions, Files

Prior to this release, the length of MATLAB identifiers (variable names, function and subfunction names, structure fieldnames, M-file names, MEX-file names, and MDL-file names) was restricted to 31 characters. Names using more than 31 characters were either truncated by MATLAB or caused a warning or error to be generated.

In MATLAB 6.5, any of these names can be up to 63 characters long.

A new function, `namelengthmax`, returns the maximum length for MATLAB identifiers.

```
namelengthmax
ans =
    63
```

If you have MATLAB programs that hard-code the maximum identifier length as 31, you should replace these hard-coded limits with a call to `namelengthmax`.

If you use identifiers that exceed 63 characters, MATLAB issues a warning and truncates any characters beyond the 63rd.

Characters Beyond 31 Are No Longer Ignored. In previous versions of MATLAB, if you had two or more long identifiers in which the first 31 characters were identical, MATLAB ignored any characters beyond the thirty first and thus recognized only one of the identifiers. For example, these two Stateflow filenames

```
stateflow_modelname_with_40_characters_1.mdl
stateflow_modelname_with_40_characters_2.mdl
```

both appeared to MATLAB 6.1 as shown below, and MATLAB recognized only one of the files:

```
stateflow_modelname_with_40_cha.mdl
```

In MATLAB 6.5, with the maximum MDL-file name length increased to 63, MATLAB recognizes both files. You should be aware of this change, as it could possibly lead to unexpected behavior.

Warning for Identifiers Longer Than 31. In MATLAB 6.5, if you use an identifier that exceeds the previous limit of 31 characters, MATLAB can optionally generate a warning of the form:

```
<identifier> exceeds MATLAB's previous maximum name length limit  
of 31 characters.
```

This warning is disabled by default. You can enable it by typing

```
warning on MATLAB:usinglongnames
```

Note Unlike most MATLAB warnings, you cannot enable this warning with the commands, `warning on` or `warning on all`. You must use the command shown above.

Warning for Identifiers Longer Than `namelengthmax`. If you specify an identifier that exceeds the new character limit, MATLAB generates the following warning:

```
<identifier> exceeds MATLAB's maximum name length of  
<namelengthmax> characters and has been truncated to  
<truncated_identifier>
```

This warning is enabled by default. You can disable it by typing the following command. However, we strongly encourage you to leave it enabled:

```
warning off MATLAB:namelengthmaxexceeded
```

MATLAB Toolbox Functions Updated. MATLAB toolbox functions, such as `isvarname`, have been updated in MATLAB 6.5 to make use of the `namelengthmax` function, and thus return the correct values.

Effect On P-Code and MEX-files. You should recompile the following files:

- Any P-Code files that contain identifiers longer than 31 characters.
- Any MEX-files that use the constant, `mxMAXNAME`.

The logical Attribute Is Now a Class

In previous versions of MATLAB, logical was an attribute of any numeric data type. This is illustrated in the following example (executed in MATLAB 6.1), where `b = a > 10` produces a double array with a logical attribute:

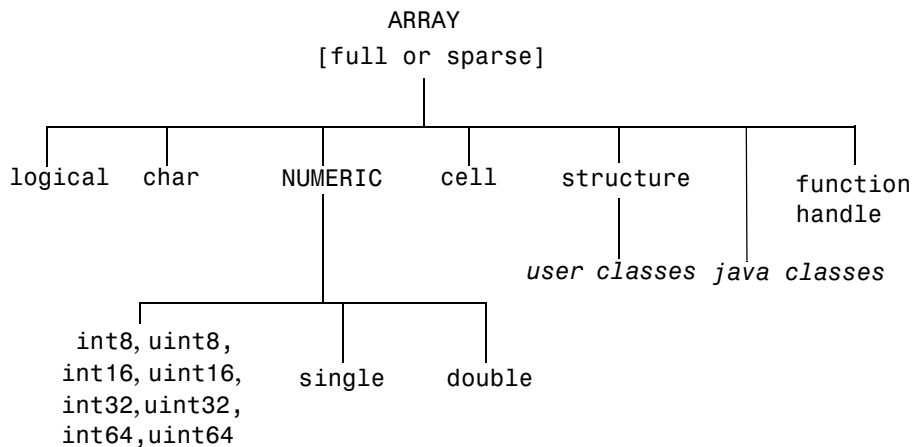
```
a = magic(4);
b = a > 10;
```

```
whos b
  Name      Size      Bytes  Class

  b         4x4         128    double array (logical)
```

Grand total is 16 elements using 128 bytes

In MATLAB 6.5, logical is a first class data type and MATLAB class. The class hierarchy diagram below shows logical to be a class, equivalent to other first class types like character and cell arrays.



The same example, executed in MATLAB 6.5, produces a result of class logical array:

```
a = magic(4);
b = a > 10;

whos b
  Name      Size      Bytes  Class

  b         4x4         16  logical array

Grand total is 16 elements using 16 bytes
```

Note Logical arrays in MATLAB 6.5 also require less storage space. In the example above, it took 128 bytes to store the array in MATLAB 6.1, and only 16 bytes in Version 6.5.

Effect on Related Functions. The table below compares the results obtained from a number of functions that operate on logical types. The variable a in the table is derived as follows:

```
a = (magic(4) > 10);
```

| Command | MATLAB 6.1 Result | MATLAB 6.5 Result |
|------------------|--------------------------|--------------------------|
| whos a | double array (logical) | logical array |
| class(a) | double | logical |
| islogical(a) | 1 | 1 |
| isnumeric(a) | 1 | 0 |
| isa(a,'double') | 1 | 0 |
| isa(a,'logical') | 0 | 1 |
| double(a) | double array (logical) | double array |

Valid logical Values. logicals can only have the values 0 and 1. When you convert real finite values other than 0 or 1 to logical, MATLAB gives them a logical 1 value and issues a warning message:

```
x = logical(5)
Warning: Values other than 0 or 1 converted to logical 1
x =
     1
```

Behavior of islogical is Unchanged. Note in the table above that `islogical` continues to return 1 for a logical array, as it did in previous releases for arrays with a logical attribute.

Array Manipulation. All generic array manipulation functions (e.g., subscripting, reference, assignment, concatenation, `size`, `length`, `numel`, `ndims`, `permute`, `diag`, etc.) work as they do in MATLAB 6.0, subject to the behaviors described in this section.

Boolean Functions. All Boolean functions (e.g., `and`, `or`, `not`, `xor`, `any`, `all`) work as they do in MATLAB 6.0, subject to the behaviors described in this section.

MAT-Files. MAT-files created with earlier versions of MATLAB that contain logical arrays will load correctly in MATLAB 6.5. Values other than 0 or 1 will be converted to 1's. A double array with a logical attribute, when loaded in MATLAB 6.5, will be a logical array.

Mixed-Mode Arithmetic. Mixed-mode arithmetic (e.g., arithmetic involving a logical and a double) dispatches to the function registered for the nonlogical data type. The logical is converted to that type and the operation proceeds. This behavior is fully backward compatible with how MATLAB works in MATLAB 6.0.

Converting logical to double. You can use the double function to convert a logical array to a double array:

```
b = magic(4) > 10;
whos b
  Name      Size      Bytes  Class

  b         4x4         16  logical array
```

Grand total is 16 elements using 16 bytes

```
b = double(b);
whos b
  Name      Size      Bytes  Class

  b         4x4        128  double array
```

Grand total is 16 elements using 128 bytes

Indexed Assignment. As a rule, MATLAB data types are preserved on indexed assignment. This now holds true for logical, as it is now a MATLAB data type.

This example creates an empty array of type logical. The indexed assignment to double that follows, `a(1) = 1`, does not change the type to double. Its logical type is preserved:

```
a = logical([]);
whos a
  Name      Size      Bytes  Class

  a         0x0         0  logical array
```

Grand total is 0 elements using 0 bytes

```
a(1) = 1;
whos a
  Name      Size      Bytes  Class

  a         1x1         1  logical array
```

Grand total is 1 element using 1 bytes

Passing NaN or Complex to logical Functions. Attempting to pass NaN or complex values to an if or while statement, or to and, or, not, or logical, now consistently generates an error:

```
logical(NaN)
??? Error using ==> logical
NaN's cannot be converted to logicals.
```

```
not(2j)
??? Error using ==> not
Operands to NOT must not be complex.
```

Creating Logical Matrices with the sparse Function. Previously, when creating sparse logical matrices, the sparse function accumulated entries when it encountered repeated indices. For example,

```
A = sparse([1 1 1], 1, logical([1 0 1]))
A =
    (1,1) 2
```

In MATLAB 6.5, sparse now returns an error, because the only valid logical values are 0 and 1:

```
A = sparse([1 1 1], 1, logical([1 0 1]))
??? Error using ==> sparse
Repeated indices are not supported for sparse logical matrices.
```

Changes to Definition of “Truth”

As MATLAB has evolved, its definition of *truth* has become complicated and inconsistent. One goal of MATLAB 6.5 is to present a simple and self-consistent definition of truth that applies in all situations.

This change affects the following types of operations.

Comparing Empty with Empty or Scalar. MATLAB now returns an empty array ([]) when you compare two 0-by-0 empty arrays, or a 0-by-0 empty array with a scalar. This behavior also affects comparisons performed inside if and while statements.

Comparing two 0-by-0 empty arrays:

```
a = [];  
  
b = (a == [])  
b =  
    []
```

Comparing a 0-by-0 empty array with a scalar:

```
b = (a == 5)  
b =  
    []
```

This behavior is now consistent with all other binary operators (e.g., >, <, ~=, +, -, .* , etc.).

Comparing Empty with Nonscalar. MATLAB now returns a dimension mismatch error when you compare a 0-by-0 empty array with a sized array:

```
a = [];  
  
a == [1 2 3]  
??? Error using ==> ==  
Matrix dimensions must agree.
```

Using NaN with any or all. The any and all functions now ignore NaN. Thus any now returns 0 for a vector having NaNs as its only nonzero elements. The behavior of all is unaffected by this change, but it means that any and all now behave consistently with other reduction operators like min and max:

```
a = [0 0 NaN 0 NaN];  
any(a)  
ans =  
    0
```

Interaction with Objects. In previous versions of MATLAB, passing a user-defined object as the argument to if or while caused the interpreter to call the object's double method (assuming it had one) in order to convert it to something whose *truth* could be determined. MATLAB 6.5 handles this situation by looking first for a logical method for the object and, upon failing to find one, calls its double method, if one exists.

If you have objects that will participate in truth evaluation, you should provide a logical method for those objects. The logical method must return 0 or 1 (false or true).

The sparse Class Is Now an Attribute

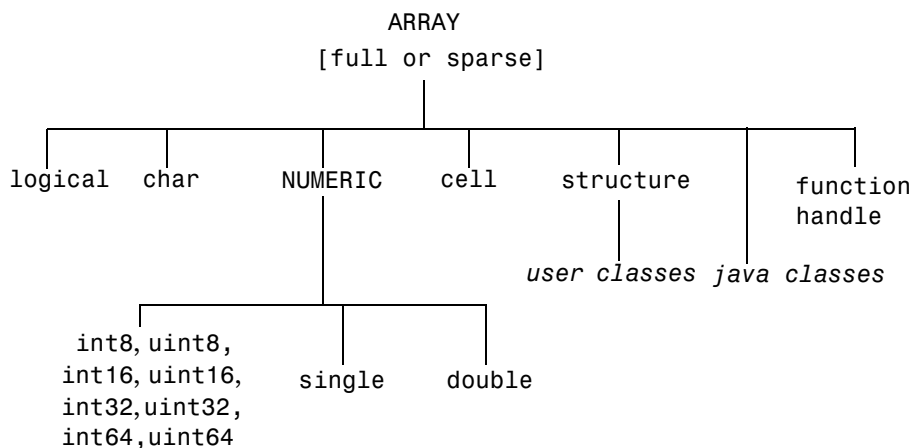
In previous versions of MATLAB, sparse was a first class data type and MATLAB class (subclass of double). This is illustrated in the following example (executed in MATLAB 6.1), where `sparse(eye(3))` produces a sparse array:

```
s = sparse(eye(3));
whos s
  Name      Size      Bytes  Class

  s         3x3         52  sparse array
```

Grand total is 3 elements using 52 bytes

In MATLAB 6.5, sparse becomes an attribute of a MATLAB class. The class hierarchy diagram below represents the MathWorks long-range plan for sparse, where full and sparse are attributes of all MATLAB classes.



Note In MATLAB 6.5, the sparse attribute is supported for the double and logical classes only.

The same example, executed in MATLAB 6.5, produces a double array with a sparse attribute:

```
s = sparse(eye(3));
whos s
      Name      Size      Bytes  Class
      ----      ---      -----  ----
      s          3x3          52  double array (sparse)

Grand total is 3 elements using 52 bytes
```

Effect on Related Functions. The table below compares the results obtained from a number of functions that operate on sparse arrays. The variable *a* in the table is derived as follows:

```
a = sparse(eye(3));
```

| Command | MATLAB 6.1 Result | MATLAB 6.5 Result |
|-----------------|--------------------------|--------------------------|
| whos a | sparse array | double array (sparse) |
| class(a) | sparse | double |
| issparse(a) | 1 | 1 |
| isa(a,'sparse') | 1 | 0 |
| isa(a,'double') | 1 | 1 |
| double(a) | double array | double array (sparse) |
| full(a) | double array | double array |
| x = logical(a) | sparse array (logical) | logical array (sparse) |
| class(x) | sparse | logical |
| full(x) | double array (logical) | logical array |

Note `issparse(a)` and `isa(a, 'sparse')` give different results. The former indicates that `a` is a sparse matrix; the latter that `a` is not of the sparse class.

Behavior of `issparse` is Unchanged. Note in the table above that `issparse` continues to return 1 for arrays with a sparse attribute, as it did in previous releases for sparse arrays.

Arithmetic Operations. Arithmetic operations continue to work on sparse doubles as they do today.

Determining Storage Type. Make sure that your programs do not use `class` or `isa` to determine if a matrix uses sparse storage. Use `issparse` instead. It is both simpler and faster.

Methods in @sparse Directories. If you have written your own algorithms for dealing with sparse matrices and placed them in an @sparse directory, MATLAB will not access them because sparse is no longer a class.

MAT-Files. MAT-files created with earlier versions of MATLAB that contain sparse arrays will load correctly in MATLAB 6.5. A sparse array, when loaded in MATLAB 6.5, will be a double array with a sparse attribute, or a logical array with a sparse attribute if the array originally had the logical attribute. (See “The logical Attribute Is Now a Class” on page 7-55 for information on changes to logical).

Converting to Full. Use `full(x)` instead of `double(x)` to ensure that variable `x` is full. The `double` function no longer removes the sparseness of an array:

```
s = sparse(eye(3));
s = double(s);
whos s
  Name      Size      Bytes  Class
  s         3x3         52  double array (sparse)

Grand total is 3 elements using 52 bytes
```

Use full instead to make the array full:

```
s = full(s);
whos s
  Name      Size      Bytes  Class

  s         3x3         72  double array
```

Grand total is 9 elements using 72 bytes

Testing for full arrays. You should no longer use the following statement to test whether an array is full (nonsparse). Because the sparse class has been removed in MATLAB 6.5, this statement now returns 1 for both full and sparse arrays:

```
strcmp(class(s), 'double') == 1
```

In place of the above statement, use the following to test for a full array:

```
~issparse(s)
```

For example, create a sparse array, s, and test to see if it is a full array:

```
s = sparse(eye(3));

~issparse(s)
ans =
    0
```

Indexed Assignment. As a rule, MATLAB data attributes are not preserved on indexed assignment. This now holds true for sparse, as it is now an attribute.

This example creates a sparse double array. The indexed assignment to a full double that follows, (s(:) = rand(3)), removes the sparse attribute from the array:

```
s = sparse(eye(3));
whos s
  Name      Size      Bytes  Class

  s         3x3         52  double array (sparse)
```

Grand total is 3 elements using 52 bytes

```
s(:) = rand(3);
whos s
      Name      Size      Bytes  Class
      a         3x3         72  double array

Grand total is 9 elements using 72 bytes
```

Logical Indexing and find

The following two statements are intended to be equivalent in MATLAB. However, prior to this release, the statements were not equivalent in the case where *a* is nondouble and *b* contains only zeros:

```
a(find(b))
a(logical(b))
```

This has been fixed in this release. For the *a* and *b* shown here, the three equations that follow return the same result:

```
a = [int8(1) int8(2) int8(3)];
b = a > 5;

r1 = a(find(a > 5));
r2 = a(b);
r3 = a(a > 5);
```

As a result of this fix, you can now use either of the last two, simpler forms in place of the form that requires the use of `find`.

Warning Control Upgrade Issues

See the section on “Warning Control Features” on page 7-25 in these Release Notes for information on how you can control the way MATLAB handles the selected warnings in your programs.

Changes to Functionality. In some cases, these changes to warning control will affect warning statements that currently exist in your code. The following two tables present how Versions 6.0 and 6.5 of MATLAB respond to MATLAB 6.0 warning syntax.

This table shows the MATLAB 6.0 behavior.

| MATLAB 6.0 Syntax | MATLAB 6.0 Behavior |
|--------------------------|---|
| warning backtrace | warning on all; Enable backtraces. |
| warning backtrace off | warning on all; Disable backtraces. |
| warning backtrace on | warning on all; Enable backtraces. |
| warning off backtrace | warning on all; Disable backtraces. |
| warning on backtrace | warning on all; Enable backtraces. |
| warning debug | warning on all; dbstop if warning |
| warning debug off | warning on all; dbclear if warning |
| warning debug on | warning on all; dbstop if warning |
| warning off debug | warning on all; dbclear if warning |
| warning on debug | warning on all; dbstop if warning |
| warning once | warning once ... <each-HG-back-compat-message-identifier> |
| warning always | warning always ... <each-HG-back-compat-message-identifier> |
| warning | ans gets one of on, off, debug, or backtrace. |
| s = warning(...) | s gets one of on, off, debug, or backtrace. Process inputs (if any) as above. |
| [s, f] = warning(...) | s gets one of on, off, debug, or backtrace. f gets one of once or always. Process inputs (if any) as above. |

For backward compatibility, MATLAB will continue to accept every usage of warning shown in the left column of the table above. However, some changes will be made to their actual behavior, as shown in the table below.

This table shows the MATLAB 6.5 behavior in response to MATLAB 6.0 warning command syntax.

| MATLAB 6.0 Syntax | MATLAB 6.5 Behavior |
|--------------------------|--|
| warning backtrace | Enable backtraces. |
| warning backtrace off | Disable backtraces. |
| warning backtrace on | Enable backtraces. |
| warning off backtrace | Disable backtraces. |
| warning on backtrace | Enable backtraces. |
| warning debug | dbstop if warning |
| warning debug off | dbclear if warning |
| warning debug on | dbstop if warning |
| warning off debug | dbclear if warning |
| warning on debug | dbstop if warning |
| warning once | Warning - warning frequency is no longer supported. |
| warning always | Warning - warning frequency is no longer supported. |
| warning | warning query all; (doesn't assign to ans). |
| s = warning(...) | s = warning('query', 'all'); then process inputs (if any) as above. |
| [s, f] = warning(...) | Warning - warning frequency is no longer supported. |

Outputs Returned by Warning. Prior to MATLAB 6.5, warning returned up to two outputs: state and frequency. Neither of these is meaningful anymore, as there is no longer either a single warning state nor a single warning frequency to return.

The frequency output is now disallowed. MATLAB generates a warning if you request this output.

The warning function now returns a structure instead of a string for the state output. Any existing code that uses this output should continue to function normally, but should be examined to make sure that the state value is properly interpreted in this new context.

Formatted Error and Warning Strings

For backward compatibility, if only one input is passed to `error` or `warning`, MATLAB treats it as a fixed string, not a format string. See “Formatted String Conversion” in Errors and Warnings in the MATLAB documentation.

getfield and setfield Superseded

Since dynamic field names improve on the `getfield` and `setfield`, these two functions will eventually be removed from the MATLAB language. In MATLAB 6.5, `getfield` and `setfield` will generate a warning message encouraging you to use dynamic field names instead. See the section, “Dynamic Field Names for Structures” on page 7-25 for more information on this.

New Behavior in break

The `break` function is intended to be used within a `for` or `while` loop. Use of `break` outside of a loop results in a warning being issued.

isequal and Structure Field Creation Order

When comparing structures with `isequal`, MATLAB no longer considers the order in which the fields of the structures were created in determining equality. See Example 2 on the `isequal` reference page.

Set Operations on Cell Arrays of Strings

The `intersect`, `setdiff`, and `setxor` functions have been modified to handle cell arrays of strings having one or more trailing spaces in a manner that is

consistent with its handling of other array types. These set functions no longer ignore trailing spaces when doing the comparison. See the examples below:

| Prior to MATLAB 6.5 | MATLAB 6.5 |
|--|--|
| <pre>intersect({'A'}, {'A '}) ans = 'A'</pre> | <pre>intersect({'A'}, {'A '}) ans = {}</pre> |
| <pre>setdiff({'A'}, {'A '}) ans = {}</pre> | <pre>setdiff({'A'}, {'A '}) ans = 'A'</pre> |
| <pre>setxor({'A'}, {'A '}) ans = {}</pre> | <pre>setxor({'A'}, {'A '}) ans = 'A' 'A '</pre> |

Using mat2cell on an Empty Array

If you invoke `mat2cell` on an empty array, the function now returns an empty cell array rather than issuing an error. This requires that all zero dimensions of the empty input array have a corresponding `mat2cell` argument equal to `[]`.

In the following example, the third input argument to `mat2cell` specifies how MATLAB is to divide up the second dimension of the input array, `X`, in the resultant cell array. (See the `mat2cell` reference page for help on syntax.) Because the second dimension of `X` is of zero size, the only valid division specifier is `[]`.

```
X = rand(3, 0, 4);
C = mat2cell(X, [1 2], [], [2 1 1])
C =
    Empty cell array: 2-by-0-by-3
```

Concatenating with Empty Arrays

Empty arrays in concatenation operations can now affect the data type of the output. The only time you are likely to see this is when concatenating doubles and logicals. In previous versions of MATLAB, the example shown below returned a double array with a logical attribute. Now it returns a double because the empty double input is no longer ignored:

```
a = [ [] logical(0) ];
whos a
      Name      Size      Bytes  Class
      a         1x1         8    double array

Grand total is 1 element using 8 bytes
```

Concatenating Empty Cell Arrays. You cannot concatenate an empty cell array with numeric or character values:

```
a = ['string' {}]
??? Error using ==> horzcat
Conversion to cell from char is not possible.
```

Function Names Redefined As Variable Names

Under the conditions listed below, if you define a variable using a name that already belongs to a function, MATLAB issues the following warning message:

```
Variable <variable name> has been previously used as a function
name.
(Type "warning off MATLAB:mir_warning_variable_used_as_function"
to suppress this warning.)
```

This warning is issued only when all of the following conditions are true:

- The variable definition appears in an M-file function
- Within that M-file function, the name is used in a function call, and then later used as a variable name

For example, the first line of the function shown below uses the term `i` to call the MATLAB function that returns the complex constant. Some time later, in the for loop, the function code redefines `i` so that MATLAB now interprets it as a variable name, and assigns to the variable the values `1:10`.

```
function myfun
x = 5 + i;
```

```
for i = 1:10
    <do something>
end

y = 32 + i;
```

When MATLAB compiles this M-file function, it issues the warning message shown above. The reason for the warning is illustrated in the last line of the code shown in the example. The statement `32 + i` does not add the complex constant `i` to 32 as intended, but instead adds the value 10 to 32. This is because the opening `for` loop statement redefined the name `i` as a variable name and the last value assigned to that variable was 10.

Note that this is a compile-time warning only. M-files run for the first time in a MATLAB session, or run after being cleared from memory (e.g., by `clear` functions) may issue this warning. M-files that are executed from cache do not.

Specifying More Outputs Than a Function Defines

A function call that requests more output values than are generated by the function being called now returns an error instead of a warning. Consider the function below that declares two outputs in the function definition line and assigns a value to one of them.

```
function [A, B] = mult_by_two(C)
A = 2 * C;
```

Calling this function with one output specified in the call completes successfully. Calling the function with two outputs specified returns an error. Even though two outputs are declared in the function definition line, only one output is generated in the function body.

```
[A, B] = mult_by_two(5)
??? One or more output arguments not assigned during call to
'mult_by_two'.
```

In previous versions of MATLAB, this type of call resulted in a warning. As a result, execution of the function continued and assignment to output variable `A` completed successfully.

In MATLAB 6.5, this type of call generates an error and aborts execution of the M-file. As a result, `A` remains undefined.

Consistent Handling of Subscripting Errors

The way in which MATLAB handles invalid subscripting is more consistent in MATLAB 6.5. MATLAB now responds to all of the situations listed below with this one error message:

```
??? Subscript indices must either be real positive integers or
    logicals.
```

The types of invalid subscripting that yield this error are shown in this table.

| Type of Subscript | Example |
|-------------------|--|
| Complex | <code>x(2i)</code> |
| Noninteger | <code>x(1.2)</code> |
| Negative | <code>x(-5)</code> |
| Zero | <code>x(0)</code> |
| NaN | <code>x(NaN)</code> |
| Inf | <code>x(Inf)</code> , <code>x(-Inf)</code> |

The only functional change is that subscripting with noninteger values is now always treated as an error rather than a warning. In previous versions of MATLAB, this was an error only for sparse matrices.

Consistent Handling of Logical Errors

MATLAB now responds to the following types of invalid logical expressions as shown here:

- Bad arguments to logical (e.g., `logical(2)`):
Warning: Logical was assigned values other than 0 or 1.
- Bad assignment to logicals (e.g., `x = logical([1 0 1]); x(2) = 2`):
Warning: Logical was assigned values other than 0 or 1.
- Complex assignment to logical (e.g., `logical(i)`):
Complex argument is not allowed in LOGICAL.

- Using NaN in a logical expression (e.g., `logical(NaN)`):
NaN's cannot be assigned to logical arrays.
- NaN in an expression with `and`, `or`, `not` (e.g., `5 & NaN`):
NaN's cannot be converted to logicals.

Note Logical values are 1 (for true) and 0 (for false). Other nonzero values implicitly convert to true. Complex values and NaN cannot be converted implicitly. Use `~=0` to convert these to logicals. For example, `x = (NaN~=0)`. to make `x` logical.

Empty Array in Comparisons. When you use an empty array in an equal or not equal comparison statement, MATLAB now returns an empty array as the result. In previous versions, MATLAB returned zero and displayed a warning.

For example, this statement now returns an empty array:

```
[] == 5
ans =
    []
```

The previous return value was a vestige of much older MATLAB behavior. The new return value is now consistent with all other binary operations involving empty arrays. For example, `[] + 5` yields `[]`.

Operations That Are Now Considered Invalid

The following operations now return an error or warning.

Passing Complex to `if` or `while`. Passing a complex value to `if` or `while` now returns an error:

```
a = 5j;
if a
    disp 'true'
end
??? Complex values cannot be converted to logicals.
```

Previously, the imaginary part was ignored unless the argument was sparse.

Passing NaN to if or while. Passing NaN to if or while now returns an error:

```
a = NaN;
if a
    disp 'true'
end
??? NaN's cannot be converted to logicals.
```

Previously, this generated an error unless the NaN had the logical attribute.

Passing Complex to logical. Passing a complex argument to the logical function or assigning a complex value to a logical variable returns an error:

```
a = 5j;
x = logical(a);
??? Error using ==> logical
Complex values cannot be converted to logicals.
```

This has always been an error.

Passing NaN to logical. Passing a NaN argument to the logical function or assigning NaN to a logical variable now returns an error:

```
a = NaN;
x = logical(a)
??? Error using ==> logical
NaN's cannot be converted to logicals.
```

Previously, this assigned a the value NaN with a logical attribute.

Passing Complex to and, or, not. Passing a complex argument to the and, or, or not functions now returns an error:

```
a = 5j;
x = ~a;
??? Error using ==> ~
Operands to NOT must not be complex.
```

Previously, this assigned a the value zero.

Passing NaN to and, or, not. Passing NaN to the and, or, or not functions now returns an error:

```
a = NaN;
x = ~a
??? Error using ==> ~
NaN's cannot be converted to logicals.
```

Previously, this assigned a the value zero.

Assigning Nonlogical Values to logical. Passing incompatible arguments to the logical function or assigning them to a logical variable now generates a warning. Note in the example below that the value assigned to the logical array (100) is converted by MATLAB to logical 1.

```
a = (magic(4) > 10);
a(2,3) = 100
Warning: Values other than 0 or 1 converted to logical 1
```

```
a =
     1     0     0     1
     0     1     1     0
     0     0     0     1
     0     1     1     0
```

Previously, this resulted in no warning and assigned 100 to a(2,3).

Graphics Upgrade Issues

The issues involved in upgrading from MATLAB 6.1 to MATLAB 6.5, in terms of graphics features, are discussed below.

Change to smooth3

Calculation of the gaussian filter option of the smooth3 function has been corrected. This change may result in visual changes to graphs made with the smoothed data.

External Interfaces/API Upgrade Issues

The issues involved in upgrading from MATLAB 6.1 to MATLAB 6.5, in terms of external interfaces and API features, are discussed below. These include

- “Changes to logical and sparse Data Types” on page 7-76
- “Functions Replaced in MATLAB 6.5” on page 7-77
- “Compiling C++ Files” on page 7-79
- “LCC Support for LAPACK” on page 7-79
- “Client Support for COM” on page 7-79

Changes to logical and sparse Data Types

In MATLAB 6.5, the sparse data type has been changed to be an attribute of its underlying data type. Also, the logical data attribute has been changed to be a first class data type. See “Programming and Data Types Upgrade Issues” on page 7-52 for more information on this change.

The following sections describe how this change may affect your C programs.

No Change to `mxIsLogical`. The `mxIsLogical` function is unchanged in MATLAB 6.5. It returns `true` for logical arrays, as it did for arrays with a logical attribute in previous releases.

Testing for Numeric. In previous releases, `mxIsNumeric` returned `true` for numeric arrays with the logical attribute. This function now returns `false` for logical arrays, since logical is a nonnumeric data type.

Using `mxGetClassID` on logicals. `mxGetClassID` returns a new `mxLOGICAL_CLASS` value for logical arrays.

Using `mxGetClassID` on Sparse Arrays. `mxGetClassID` no longer returns the enumerated value `mxSPARSE_CLASS`. Instead, it returns the enumerated value corresponding to the underlying data type. Use `mxIsSparse` to determine if an `mxArray` is sparse.

Testing for Sparse. In previous releases, you could use the following statement to determine if a matrix is sparse. This does not work in MATLAB 6.5.

```
mxGetClassID(x) == mxSPARSE_CLASS
```


You should use `mxIsSparse(x)` to determine if a matrix is sparse. The `mxIsSparse` function operates the same as in previous releases and also executes faster than the operation shown above.

Testing for Sparse with `mxIsDouble`. Because sparse has been changed from a MATLAB data type to a data attribute, `mxIsDouble(x)` no longer implies `~mxIsSparse(x)`, as it did in previous releases. Test the sparseness of an array using `mxIsSparse` instead.

No Change to `mxIsSparse`. The `mxIsSparse` function is unchanged in MATLAB 6.5. It returns true for arrays with a sparse attribute, as it did for sparse arrays in previous releases.

Obsolete logical Functions. The following two functions are now obsolete. Support for these functions will be removed in a future release.

| Function | Description |
|-----------------------------|---------------------------------|
| <code>mxSetLogical</code> | Convert mxArray to logical type |
| <code>mxClearLogical</code> | Convert mxArray to numeric type |

Functions Replaced in MATLAB 6.5

MATLAB handles mxArrays more efficiently in version 6.5 by not storing a variable name in the mxArray. When an mxArray name is required, these new C and Fortran functions enable you to pass it in the argument list.

The functions shown in the left column of the table replace those in the right column. The functions shown in the right column are now obsolete and may be unavailable in a future version of MATLAB.

| New Function | Replaces |
|--------------------------------|-----------------------------|
| <code>mexGetVariable</code> | <code>mexGetArray</code> |
| <code>mexGetVariablePtr</code> | <code>mexGetArrayPtr</code> |
| <code>mexPutVariable</code> | <code>mexPutArray</code> |
| <code>engGetVariable</code> | <code>engGetArray</code> |

| New Function | Replaces |
|------------------------|-----------------------|
| engPutVariable | engPutArray |
| matDeleteVariable | matDeleteArray |
| matGetVariable | matGetArray |
| matGetVariableInfo | matGetArrayHeader |
| matGetNextVariable | matGetNextArray |
| matGetNextVariableInfo | matGetNextArrayHeader |
| matPutVariable | matPutArray |
| matPutVariableAsGlobal | matPutArrayAsGlobal |

For example, you should replace the second and third line shown here

```
parr = mxCreateDoubleMatrix(0, 0, 0);
mxSetName(parr, name);
retval = matPutArray(ph, parr);
```

with the second line shown below. The name of the mxArray is passed with `matPutVariable` rather than stored in the mxArray by `mxSetName`:

```
parr = mxCreateDoubleMatrix(0, 0, 0);
retval = matPutVariable(ph, name, parr);
```

mxCreateScalarDouble Replaced. New function `mxCreateDoubleScalar` replaces `mxCreateScalarDouble`. The latter function is still supported at this time, but support may be removed in a future release.

| New Function | Replaces |
|----------------------|----------------------|
| mxCreateDoubleScalar | mxCreateScalarDouble |

Compiling C++ Files

You no longer need to use the preconfigured options file, `cxcopts.sh`, to compile C++ MEX-files. MATLAB recognizes the following file extensions as C++ extensions, and automatically uses the C++ compiler.

```
.cxx  
.cpp  
.cc
```

The `cxcopts.sh` file is no longer available in MATLAB.

Also, on UNIX, you must now use the `-cxx` switch to the MEX script if you are linking C++ objects.

LCC Support for LAPACK

On Windows platforms, you can now compile and link C MEX-files that call LAPACK and BLAS functions using the MATLAB C compiler, `Lcc`. Use the following command to compile the file `myCmexFile.c` and link it with the LAPACK library file, `libmwlpack.lib`.

```
mex myCmexFile.c <matlab>/extern/lib/win32/lcc/libmwlpack.lib
```

The term `<matlab>` stands for the MATLAB root directory.

Client Support for COM

Client support for the MATLAB COM interface has changed significantly in MATLAB 6.5. There are many new features as well as important changes in previously supported features. This section describes how these changes may affect your existing programs.

See “New COM Client Support Features” on page 7-34 in these Release Notes and “MATLAB COM Client Support” in the MATLAB documentation for more information.

Creating an Object or Interface. When you create a COM control or server with `actxcontrol` or `actxserver`, MATLAB returns a COM object which now is displayed as `COM.<string>` rather than as `activex object`:

```
h = actxserver('Excel.Application')  
h =  
    COM.excel.application
```

This also applies to interfaces to a COM object. MATLAB represents the interface as `Interface.<string>` rather than as `activex object`:

```
w = get(h, 'Workbooks')
w =
    Interface.excel.application.Workbooks
```

New Error on Non-Existent ProgID. Both `actxcontrol` and `actxserver` return a different error message when an invalid ProgID is entered:

```
h = actxcontrol('xxxxx')
??? Error using ==> actxcontrol
Control creation failed. Invalid ProgID 'xxxxx'
```

Data Returned by get. Information returned by the `get` function now shows the type for each interface:

```
h = actxserver ('Excel.Application');
get(h)
ans =
    Application: [1x1 Interface.excel.application.Application]
    Parent: [1x1 Interface.excel.application.Parent]
    Windows: [1x1 Interface.excel.application.Windows]
    Workbooks: [1x1 Interface.excel.application.Workbooks]
    .
    .
```

Property names returned by `get` are no longer arranged alphabetically. They are displayed in the order that MATLAB gets them from the Type Library.

Set Invoked with No Arguments. When you invoke the `set` function without any arguments other than the object or interface handle, MATLAB no longer returns an error. Instead it returns a structure array, listing all properties for the object. The structure array also contains enumerated values for those properties that allow you to express values as enumerated strings.

Old error message:

```
set(h)
??? Index exceeds matrix dimensions.
```

Values returned in MATLAB 6.5:

```
set(h)
ans =
    Application: {}
    Creator: {'xlCreatorCode'}
    Parent: {}
    Cursor: {4x1 cell}
    .
    .
```

Change in Method Data Returned by invoke. The invoke function now returns more useful data on the methods of an object or interface. Note the differences shown in the example below:

```
h = actxserver('excel.application');

% Invoke, prior to MATLAB 6.5
invoke(h)
DeleteCustomList = Void DeleteCustomList (Int)
MailLogon = Void MailLogon (Variant[opt], Variant[opt],
    Variant[opt])
NextLetter: 'Variant(Pointer) NextLetter ()'
:
:

% Invoke, in MATLAB 6.5
invoke(h)
DeleteCustomList: 'void DeleteCustomList(handle, int32)'
MailLogon: 'void MailLogon(handle, [Optional]Variant)'
NextLetter: 'handle NextLetter(handle)'
:
:
```

Also note that the required handle argument is now explicitly shown.

Methods Function for COM. The methods function now returns the names for *all* methods of the specified class:

```
h = actxcontrol('MWSAMP.MWSampCtrl.1');  
methods(h)
```

```
Methods for class COM.mwsamp.mwsampctrl.1:
```

```
AboutBox          GetR8Array         SetR8              move  
Beep              GetR8Vector        SetR8Array         propedit  
FireClickEvent   GetVariantArray   SetR8Vector        release  
GetBSTR           GetVariantVector  addproperty        save  
:  
:
```

You can also use the `methodsview` function on COM objects now to get a graphical display of object properties.

Properties with Arguments. Any property that takes arguments is treated as a method in MATLAB 6.5. For example, the `Range` and `Item` properties in an Excel application server are now methods.

So, this statement, where `Item` is a *property* of `Sheets`

```
sheet2 = get(Sheets, 'Item', 2);
```

can now be replaced by the following, where `Item` is now a *method* of `Sheets`.

```
sheet2 = invoke(Sheets, 'Item', 2);
```

If you request a list of properties and methods for `Sheets` (using `get` and `invoke`, respectively), MATLAB now lists `Item` as a method.

For backward compatibility, functions in MATLAB 6.5 support properties that take arguments both as methods and as properties.

Arguments to Event Handlers. When a control triggers an event, MATLAB passes arguments from the control to any registered event handlers. MATLAB now passes two additional arguments:

- A string argument, holding the name of the event.
- A structure argument, holding the event name, control name, event identifier, event argument names, and event argument values.

See “Writing Event Handlers” in the MATLAB documentation for more information on changes affecting event handlers.

Specifying Events Using Identifiers. When registering events with their handler functions using either the `actxcontrol` or `registerevent` function, you can specify events either by event ID number or by event name.

Using event ID numbers:

```
h = actxcontrol('MWSAMP.MwsampCtrl1.2', [0 0 200 200], f, ...
    {-600, 'myclick'; -601 'my2click'; -605 'mymoused'});
```

Using event names:

```
h = actxcontrol('MWSAMP.MwsampCtrl1.2', [0 0 200 200], f, ...
    {'Click', 'myclick'; 'DblClick' 'my2click'; ...
    'MouseDown' 'mymoused'});
```

Use the new `events` function to display the names of all events recognized by the COM object in use. For example, to list all events for the `mwsamp` control, use

```
f = figure ('pos', [100 200 200 200]);
h = actxcontrol ('mwsamp.mwsampctrl1.2', [0 0 200 200], f);
```

```
events(h)
Click = void Click()
DblClick = void DblClick()
MouseDown = void MouseDown(int16 Button, int16 Shift,
    Variant x, Variant y)
```

Boolean Return Values. Invoking `get` on a property that returns a Boolean value now returns 1 to indicate true. Previously, it returned -1 for true:

```
h = actxserver('Excel.Application');
set(h, 'DisplayStatusBar', 1);
get(h, 'DisplayStatusBar')
ans =
    1
```

Also, invoking a method that returns a Boolean value now returns 1 to indicate true. This also previously returned -1 for true:

```
h = actxserver('Excel.Application');
invoke(h, 'Wait', 5)
ans =
    1
```

Argument Callouts in Error Messages. When a MATLAB client sends a command with an invalid argument to a COM server application, the server sends back an error message similar to that shown here, identifying the invalid argument. Be careful when interpreting the argument callout in this type of message.

```
PutFullMatrix(handle, 'a', 'base', 7, [5 8]);
??? Error: Type mismatch, argument 3.
```

In the `PutFullMatrix` command shown above, it is the fourth argument, 7, that is invalid. (It is scalar and not the expected array data type.) However, the error message identifies the failing argument as argument 3.

This is because the COM server receives only the last four of the arguments shown in the MATLAB code. (The `handle` argument merely identifies the server. It does not get passed to the server). So the server sees 'a' as the first argument, and the invalid argument, 7, as the third.

As another example, submitting the same command with the `invoke` function makes the invalid argument fifth in the MATLAB client code. Yet the server still identifies it as argument 3 because neither of the first two arguments are seen by the server.

```
invoke(handle, 'PutFullMatrix', 'a', 'base', 7, [5 8]);
??? Error: Type mismatch, argument 3.
```

Releasing and Deleting Controls or Servers. This release addresses a potential memory leak in MATLAB. The leak was caused by the following:

- MATLAB did not completely clean up COM objects or interfaces without the explicit use of `release` or `delete`. For example, the following `clear` command did not release all memory used by the object. An explicit `release(h)` was required before the `clear`:

```
h = actxcontrol('MWSAMP.MwsampCtrl.1');
clear all
```


- When a variable representing a COM object or interface was successfully assigned a new value, MATLAB did not release all of the memory originally allocated to the object or interface.
- When such a variable went out of scope, MATLAB did not release all of the memory originally allocated.

Explicit release or deletion of a COM object or interface is no longer necessary. MATLAB successfully clears the object or interface from memory when `clear` is invoked, or when the variable that represents the object or interface is either assigned a new value or goes out of scope.

Creating Graphical User Interfaces (GUIDE) Upgrade Issues

Upgrading from MATLAB 6.1 to MATLAB 6.5, in terms of GUIDE-related features, involves the following issue.

Using GUIDE Version 6.5 to Open GUIs Created in Versions 6.0 or 6.1

GUIDE generates a GUI's associated M-file with a new structure for Version 6.5. If you open a GUI that was created in Guide versions 6.0 or 6.1 using GUIDE version 6.5, the GUI will continue to function as it did previously. However, GUIDE does not update the existing code in the GUI's M-file to match the new style. If you add new components to the GUI in GUIDE version 6.5, GUIDE generates callbacks for the new components using the new M-code style, but leaves the original callbacks unchanged.

Known Software and Documentation Problems

This section includes a link to a description of known software and documentation problems in MATLAB 6.5.

If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

For a list of bugs reported in the previous release that remain open, see “Known Software and Documentation Problems” on page 8-29 in the MATLAB 6.1 Release Notes.

MATLAB 6.1 Release Notes

| | |
|--|------|
| New Features | 2-2 |
| Development Environment Features | 2-2 |
| Mathematics Features | 2-5 |
| Programming and Data Types Features | 2-8 |
| Graphics Features | 2-10 |
| OpenGL Renderer Feature — Microsoft Windows | 2-11 |
| External Interfaces/API Features | 2-12 |
| Creating Graphical User Interfaces — GUIDE | 2-17 |
| | |
| Major Bug Fixes | 2-18 |
| Development Environment | 2-18 |
| Mathematics | 2-18 |
| | |
| Upgrading from an Earlier Release | 2-23 |
| Development Environment Issues | 2-23 |
| Mathematics Issues | 2-24 |
| Programming and Data Types Issues | 2-25 |
| Graphics Issue | 2-26 |
| External Interfaces/API Issues | 2-27 |
| | |
| Known Software and Documentation Problems | 2-29 |
| Development Environment Problems | 2-29 |
| Documentation Updates | 2-30 |

New Features

This section introduces the new features and enhancements added in MATLAB 6.1 since MATLAB 6.0 (Release 12.0).

This section about new features is organized into the following subsections:

- “Development Environment Features” on page 8-2
- “Mathematics Features” on page 8-5
- “Programming and Data Types Features” on page 8-8
- “Graphics Features” on page 8-10
- “OpenGL Renderer Feature — Microsoft Windows” on page 8-11
- “External Interfaces/API Features” on page 8-12
- “Creating Graphical User Interfaces — GUIDE” on page 8-17

Development Environment Features

Command Window

MATLAB 6.1 includes two command window enhancements:

- You can set a preference for the command window to wrap lines. Input and output lines wrap to fit within the current width of the command window.
- If an error message appears when running an M-file, click on the underlined portion of the error message, or press **Ctrl+Enter**. The offending M-file opens in the Editor, scrolled to the line containing the error.

Help Browser

When you select documentation for the product filter, you can clear all currently selected products or select all products.

Editor/Debugger

The Editor/Debugger has the following enhancements:

- You can set bookmarks in M-files in the Editor/Debugger so that you can go directly to a particular line in the file. To set a bookmark, position the cursor at the line you want to bookmark, and then select **Set/Clear Bookmark** from the **Edit** menu.

After setting bookmarks, you can go to the next or previous bookmark in a file. This allows you to go directly to a marked spot. Use the **Edit** menu items **Next Bookmark** and **Previous Bookmark** to navigate. Bookmarks are not saved when you close a file.

- You can include line numbers when printing files from the Editor/Debugger. To include line numbers, select **Preferences -> Editor/Debugger -> Printing**. Under **Print options**, check **Print line numbers**.
- You can use keyboard shortcuts to comment or uncomment a selection in the Editor/Debugger. The shortcuts are platform dependent and are listed with the menu items on the Editor/Debugger **Text** menu.
- In the **Find/Replace** dialog box, settings for **Match case**, **Whole word**, and **Wrap around** are remembered for the next MATLAB session.
- You can find the previous occurrence of a selection in the Editor/Debugger by pressing **Ctrl+Shift+F3**. You can also find the previous occurrence of a string you entered into the **Find & Replace** dialog box by pressing **Shift+F3**.
- When you move an arrow key over a token, for example, an opening parenthesis, (, the token and its match are briefly underlined. If there is no matching token, the token appears with a strike-through mark, ~~(~~.
- When you run a file from the Editor/Debugger and the file is not in a directory on the search path or in the current directory, a dialog box appears presenting you with options that allow you to run the file. You can either change the current directory to the directory containing the file, or you can add to the search path the directory containing the file.

If the file you want to run is already in a directory on the search path or in the current directory, the current directory remains as is and there are no actions you need to take.

- When you add a breakpoint to a file that is not in a directory on the search path or in the current directory, a dialog box appears presenting you with options that allow you to add the breakpoint. You can either change the current directory to the directory containing the file, or you can add to the search path the directory containing the file.

If the file you want to run is already in a directory on the search path or in the current directory, the current directory remains as is and there are no actions you need to take.

- If you type `edit filename` and `filename` does not exist, a prompt appears asking if you want to create a new file. If you select **Yes**, a blank file titled `filename.m` is created in the Editor/Debugger. You can turn off this option in preferences for the Editor/Debugger.

Current Directory Browser

In the **Find/Replace** dialog box, settings for **Match case**, **Whole word**, and **Subdirectories** are remembered for the next MATLAB session.

Also, you can delete directories that are not empty. All contents of the directory will be deleted along with the directory.

Workspace Browser

You can select the column on which to sort in the Workspace browser, as well as reverse the sort order of any column. Click on a column heading to sort on that column. Click on the column heading again to reverse the sort order in that column. For example, to sort on **Size**, click the column heading once. To change from ascending to descending, click on the heading again.

Source Control

If you use Merant PVCS with MATLAB source control features, you no longer need to specify the project configuration file using `cmopts`. If you did specify it in previous releases, you do not have to remove it as MATLAB will ignore it.

General

The computer function now displays the endian byte ordering of the computer with the following form.

```
[str,maxsize,endian] = computer
```

Mathematics Features

Evaluation of Solutions to Differential Equation Problems

A new function, `deval`, enables you to evaluate the solution of a differential equation problem at a vector of points from the interval in which the problem was solved. `deval` uses, as input, the output structure `sol` of an initial value problem solver (`ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`, `ode23t`, `ode23tb`) or the boundary value problem solver (`bvp4c`). A new ODE solver syntax returns the structure `sol`.

Additional Functions Use Qhull

These functions are now based on Qhull:

- `delaunay` — two-dimensional Delaunay triangulation
- `convhull` — two-dimensional convex hull

These functions call `delaunay` and therefore are now indirectly based on Qhull:

- `voronoi` — two-dimensional Voronoi diagrams
- `griddata` — data gridding and surface fitting

These functions are in addition to the Qhull-based functions introduced in MATLAB 6.0 (Release 12.0): `convhulln`, `delaunay3`, `delaunayn`, `griddata3`, `griddatan`, and `voronoin`.

Math Function Summary Tables

This section summarizes

- New math functions
- Functions with new or changed capabilities

Note See “Upgrading from an Earlier Release” on page 8-23 for information about obsolete functions.

New Math Functions

| Function | Purpose |
|-----------|---|
| deval | Evaluate the solution of a differential equation problem using the output of ode45, ode23, ode113, ode15s, ode23s, ode23t, ode23tb, or bvp4c. |
| erfcinv | Inverse complementary error function. |
| tetramesh | Tetrahedron mesh plot for use with delaunayn. |
| triplot | 2-D triangular plot for use with delaunay. |

Math Functions with New or Changed Capabilities

| Function | Enhancement/Change |
|-----------|--|
| bvpinit | New syntax <code>solinit = bvpinit(sol,[anew bnew])</code> extrapolates a solution <code>sol</code> as an initial guess for solving a BVP on an extended interval. It can copy parameters from the previous iteration or let the user to provide new ones. For more information, see “Boundary Value Problems for ODEs” in the MATLAB documentation. |
| bvpset | New Vectorized option lets you pass to the solver <code>bvp4c</code> an array of column vectors. This allows <code>bvp4c</code> to reduce the number of function evaluations, and may significantly reduce solution time. For more information see “Boundary Value Problems for ODEs” in the MATLAB documentation. |
| convhull | New syntax <code>[K,a] = convhull(x,y)</code> returns the area <code>a</code> of the convex hull. |
| convhulln | New syntax <code>[K,v] = convhulln(X)</code> returns the volume <code>v</code> of the convex hull. |

Math Functions with New or Changed Capabilities (Continued)

| Function | Enhancement/Change |
|---|---|
| numel | New syntax <code>n = numel(A, varargin)</code> returns the number of subscripted elements, <code>n</code> , in <code>A(index1,index2,...,indexn)</code> , where <code>varargin</code> is a cell array whose elements are <code>index1, index2, ..., indexn</code> . |
| ode45, ode23, ode113, ode15s, ode23s, ode23t, ode23tb | New syntax <code>sol = solver(odefun,[t0 tf],y0...)</code> returns a structure that you can use with the new function <code>deval</code> to evaluate the solution at any point on the interval <code>[t0,tf]</code> . |
| polyeig | New syntax <code>e = polyeig(A0,A1,...,Ap)</code> returns only the eigenvalues of the specified eigenvalue problem. Use <code>[X,e] = polyeig(A0,A1,...,Ap)</code> if you also want the eigenvectors. This capability is available in MATLAB 6.0 (Release 12.0). |
| ppval | New syntax <code>ppval(xx,pp)</code> transposes the input arguments to enable you to use <code>ppval</code> with function functions. |
| qz | New syntax <code>[AA,BB,Q,Z,V,W] = qz(A,B)</code> returns <code>W</code> , the left generalized eigenvectors of <code>A</code> and <code>B</code> . |
| reshape | New syntax <code>reshape(A,...,[],...)</code> calculates the length of the dimension specified by the placeholder <code>[]</code> . |
| svd | Can now return only the first two outputs, <code>U</code> and <code>S</code> , where <code>S</code> is a diagonal matrix of the same dimension as the input argument <code>X</code> , and <code>U</code> is a unitary matrix. |

Programming and Data Types Features

Partial Evaluation of Expressions

Within the context of an `if` or `while` expression, MATLAB does not necessarily evaluate all parts of a logical expression. In some cases, it is possible, and often advantageous, to determine whether an expression is true or false through only partial evaluation. This is sometimes referred to as *short-circuiting*.

For example, if `A` equals zero in statement 1 below, then the expression evaluates to `false`, regardless of the value of `B`. In this case, there is no need to evaluate `B` and MATLAB does not do so. In statement 2, if `A` is nonzero, then the expression is `true`, regardless of `B`. Again, MATLAB does not evaluate the latter part of the expression.

```
1) if (A & B)           2) if (A | B)
```

You can use this property to your advantage to cause MATLAB to evaluate a part of an expression only if a preceding part evaluates to the desired state.

Note Partial evaluation of expressions in `if` and `while` was also available in MATLAB 6.0, but was not documented.

New MATLAB Search String Function

`strfind` is a new character array function in MATLAB. It searches for all occurrences of a string pattern within another, longer string. Placement of the two string arguments in the argument list requires that you be specific about which string is the character pattern to search for and which is the string in which to search. This allows you more control over how the search is performed compared with the MATLAB `findstr` function, particularly when executing searches within a loop.

New File I/O Functions for Scientific Data Formats

There are six new MATLAB 6.1 functions that enable you to retrieve information and data from Common Data Format (CDF), Flexible Image Transport System (FITS), and Hierarchical Data Format (HDF) files.

| Function | Purpose |
|-----------------------|---|
| <code>cdfinfo</code> | Return information about a CDF file |
| <code>cdfread</code> | Read data from a CDF file |
| <code>fitsinfo</code> | Return information about a FITS file |
| <code>fitsread</code> | Read data from a CDF file |
| <code>hdinfo</code> | Return information about an HDF or HDF-EOS file |
| <code>hdread</code> | Read data from an HDF or HDF-EOS file |

New Audio Functions

MATLAB 6.1 includes two new audio functions for 32-bit Windows platforms only.

| Function | Purpose |
|----------------------------|---|
| <code>audioplayer</code> | Create an audio object to play audio data |
| <code>audiorecorder</code> | Create an audio object to record audio data |

Date Conversion Changes

The `datenum` and `datestr` functions can now accept a date vector, as defined by `datevec`, as an input argument. For example, `datestr(clock)` returns the current date and time as string such as 27-Apr-2001 15:58:41.

Graphics Features

Transparent Legend

You can now make the legend box transparent, enabling you to see the plotted data behind the legend. See `legend` for more information.

New Ghostscript Drivers

The following new Ghostscript drivers are available with MATLAB by using the device switch shown below.

| Printer Driver | Device Switch |
|-------------------------------|---------------|
| Canon Color BubbleJet BJC-800 | -dbjc800 |
| HP LaserJet 4.5L and 5P | -dljet4 |
| HP LaserJet 5 and 6 | -dpxlmono |

New Ghostscript Output Filters for Exporting

The following new Ghostscript output filters are available with MATLAB by using the option switch shown below.

| File Format | Option Switch |
|-----------------------|---------------|
| BMP Monochrome BMP | -dbmpmono |
| PDF Color file Format | -dpdf |

Higher Resolution Metafiles

You can now set the resolution of a Windows Enhanced Metafile copied from a MATLAB figure window with the `print -dmeta` command. Set the resolution using the `-d` option of the `print` command. For example, to copy a figure to a metafile having a resolution of 200 dpi, use

```
print -dmeta -r200
```

MATLAB uses the screen resolution as the default.

Default PaperType and PaperUnits Set For International Users

The `matlabrc.m` startup file now sets the default `PaperType` and `PaperUnits` properties based on ISO Country Codes. These default to 'a4' and 'centimeters' respectively for users in countries that normally default to these settings. Other countries still default to 'usletter' and 'inches'.

The same values are used for default Simulink `PaperType` and `PaperUnits` properties in the `matlabrc.m` startup file.

You can still set default `PaperType` or `PaperUnits` values yourself by adding the following to `startup.m`.

```
set(0, 'DefaultFigurePaperType', 'a4')
set(0, 'DefaultFigurePaperUnits', 'centimeters')
```

OpenGL Renderer Feature – Microsoft Windows

If you do not want to use hardware OpenGL, but do want to use object transparency, you can issue the following command.

```
feature('UseGenericOpenGL',1)
```

This command forces MATLAB to use generic OpenGL on Microsoft Windows platforms. Generic OpenGL is useful if your hardware version of OpenGL does not function correctly and you want to use image, patch, or surface transparency, which requires the OpenGL renderer.

To reenable hardware OpenGL, use the command

```
feature('UseGenericOpenGL',0)
```

Note that the default setting is to use hardware OpenGL

To query the current state of the generic OpenGL feature, use the command

```
feature('UseGenericOpenGL')
```

See the `opengl` reference page for additional information.

External Interfaces/API Features

Concatenation of Java Arrays

In MATLAB 6.1, you can concatenate arrays of Java objects that have unlike dimensions. The following example concatenates a 2-by-3 array of `java.lang.Integer` with a 4-by-3 array of the same class.

```
A =  
java.lang.Integer[][]:  
 [ 1] [ 2] [ 3]  
 [ 4] [ 5] [ 6]  
 [17] [18] [19]  
 [20] [21] [22]
```

```
B =  
java.lang.Integer[][]:  
 [11] [12] [13]  
 [14] [15] [16]
```

The vertical concatenation `[A;B]` is simple since both arrays have the same number of columns. The horizontal concatenation `[A B]` merges the two arrays into an irregularly shaped array having six columns in the first and second rows and three columns in the third and fourth rows.

```
C = [A;B]                C = [A B]  
C =                      C =  
java.lang.Integer[][]:  
 [ 1] [ 2] [ 3]          [6 element array]  
 [ 4] [ 5] [ 6]          [6 element array]  
 [11] [12] [13]          [3 element array]  
 [14] [15] [16]          [3 element array]  
 [17] [18] [19]  
 [20] [21] [22]
```

Note “Concatenation of Java Objects” on page 8-27 discusses changes to how Java objects are concatenated.

New Fortran MX, MEX, MAT, and ENG Functions

The following functions have been added to the Fortran MX, MEX, MAT, and Engine external interface. Most of these functions already exist in the MATLAB C language API.

Table 8-1: New Fortran MX Functions

| | |
|--------------------------------------|--|
| <code>mxAddField</code> | <code>mxCalcSingleSubscript</code> |
| <code>mxClassIDFromClassName</code> | <code>mxClearLogical</code> |
| <code>mxCopyComplex8ToPtr</code> | <code>mxCopyInteger1ToPtr</code> |
| <code>mxCopyInteger2ToPtr</code> | <code>mxCopyPtrToComplex8</code> |
| <code>mxCopyPtrToInteger1</code> | <code>mxCopyPtrToInteger2</code> |
| <code>mxCopyPtrToReal14</code> | <code>mxCopyReal14ToPtr</code> |
| <code>mxCreateCellArray</code> | <code>mxCreateCellMatrix</code> |
| <code>mxCreateCharArray</code> | <code>mxCreateCharMatrixFromStrings</code> |
| <code>mxCreateDoubleMatrix</code> | <code>mxCreateNumericArray</code> |
| <code>mxCreateNumericMatrix</code> | <code>mxCreateScalarDouble</code> |
| <code>mxCreateStructArray</code> | <code>mxCreateStructMatrix</code> |
| <code>mxDestroyArray</code> | <code>mxDuplicateArray</code> |
| <code>mxGetCell</code> | <code>mxGetClassID</code> |
| <code>mxGetClassName</code> | <code>mxGetData</code> |
| <code>mxGetDimensions</code> | <code>mxGetElementSize</code> |
| <code>mxGetEps</code> | <code>mxGetField</code> |
| <code>mxGetFieldByNumber</code> | <code>mxGetFieldNameByNumber</code> |
| <code>mxGetFieldNumber</code> | <code>mxGetImagData</code> |
| <code>mxGetInf</code> | <code>mxGetNaN</code> |
| <code>mxGetNumberOfDimensions</code> | <code>mxGetNumberOfElements</code> |
| <code>mxGetNumberOfFields</code> | <code>mxIsCell</code> |

Table 8-1: New Fortran MX Functions (Continued)

| | |
|-------------------------------|---------------------------------|
| <code>mxIsChar</code> | <code>mxIsClass</code> |
| <code>mxIsEmpty</code> | <code>mxIsFinite</code> |
| <code>mxIsFromGlobalWS</code> | <code>mxIsInf</code> |
| <code>mxIsInt8</code> | <code>mxIsInt16</code> |
| <code>mxIsInt32</code> | <code>mxIsLogical</code> |
| <code>mxIsNaN</code> | <code>mxIsSingle</code> |
| <code>mxIsStruct</code> | <code>mxIsUint8</code> |
| <code>mxIsUint16</code> | <code>mxIsUint32</code> |
| <code>mxMalloc</code> | <code>mxRealloc</code> |
| <code>mxRemoveField</code> | <code>mxSetCell</code> |
| <code>mxSetData</code> | <code>mxSetDimensions</code> |
| <code>mxSetField</code> | <code>mxSetFieldByNumber</code> |
| <code>mxSetImagData</code> | <code>mxSetLogical</code> |

Table 8-2: New Fortran MEX Functions

| | |
|-------------------------------------|--------------------------------------|
| <code>mexFunctionName</code> | <code>mexGetArray</code> |
| <code>mexGetArrayPtr</code> | <code>mexIsGlobal</code> |
| <code>mexIsLocked</code> | <code>mexLock</code> |
| <code>mexMakeArrayPersistent</code> | <code>mexMakeMemoryPersistent</code> |
| <code>mexPutArray</code> | <code>mexUnlock</code> |
| <code>mexWarnMsgTxt</code> | |

Table 8-3: New Fortran MAT Functions

| | |
|--------------------------------|------------------------------|
| <code>matDeleteArray</code> | <code>matGetArray</code> |
| <code>matGetArrayHeader</code> | <code>matGetNextArray</code> |

Table 8-3: New Fortran MAT Functions (Continued)

| | |
|-----------------------|-------------|
| matGetNextArrayHeader | matPutArray |
| matPutArrayAsGlobal | |

Table 8-4: New Fortran Engine Functions

| | |
|-------------|-------------|
| engGetArray | engPutArray |
|-------------|-------------|

Property Added to ActiveX and Engine Interfaces

For ActiveX automation server applications and MATLAB Engine applications running on Windows, you can control whether the application windows appear on the Windows desktop with a new property called `Visible`.

When `Visible` is set, the ActiveX application or engine server window is visible on the desktop, thus enabling user interaction with the server. This is the default. When `Visible` is cleared, the application or engine window is removed from the desktop.

ActiveX. This example disables the visibility of an ActiveX automation server application by setting `h.visible` to 0. It checks the visibility setting in line 3 by examining `h.visible`.

```
h = actxserver('Matlab.Application');
h.visible = 0;

h.visible
ans =
    0
```

MATLAB Engine. For a MATLAB engine session, use the `engSetVisible` and `engGetVisible` functions that are new in MATLAB 6.1. Line 4, below, disables the visibility of the MATLAB engine window using `engSetVisible` with an argument of 0. Line 5 checks this setting with `engGetVisible`.

```
Engine *ep;
bool vis;
ep = engOpen(NULL);
engSetVisible(ep, 0);
engGetVisible(ep, &vis);
```

Serial I/O

The MATLAB serial port interface provides direct access to peripheral devices such as modems, printers, and scientific instruments that you connect to your computer's serial port. This interface is established through a serial port object, which you create with the `serial` function.

Freeing the Serial Port on Windows Platforms. The serial port object uses the `javax.comm` package to communicate with the serial port. However, due to a memory leak in `javax.comm`, the serial port object is not released from memory. You can use the `freeserial` function to release the MATLAB hold on the serial port.

`freeserial` is necessary only on Windows platforms. You should use `freeserial` only if you need to connect to the serial port from another application after a serial port object has been connected to that port, and you do not want to exit MATLAB.

Events, Callbacks, and Function Handles. Action properties and action functions are now referred to as callback properties and callback functions. This new terminology is reflected in new names for the associated properties and functions. The general rule for the name changes is to change "Action" to "Fcn" for properties, and "action" to "callback" for functions. For example, `TimerAction` has been renamed `TimerFcn`, and `instraction` has been renamed `instrcallback`.

Additionally, if you want to automatically pass the object and event information to the callback function, then you must specify the function as either a function handle or as a cell array. Note that you can also specify the callback function as a string. In this case, the callback is evaluated in the MATLAB workspace and no requirements are made on the function's input arguments.

Enhancements to Existing Properties.

- **Terminator Property** – You can configure `Terminator` to a decimal value ranging from 0 to 127, to the equivalent ASCII character, to CR/LF or LF/CR, or to empty (").
- **Timer events** – Some timer events may not be processed if your system is significantly slowed or if the `TimerPeriod` value is too small. The minimum `TimerPeriod` value is now 0.01 second.

Creating Graphical User Interfaces – GUIDE

This section lists the changes made to GUIDE for Release 12.1:

- The Layout Editor **Edit** menu has **Undo** and **Redo** items. You can undo or redo layout actions and property settings (with the exception of the figure `FileName` property).
- The **Application Option** dialog supports a new option for **Command-line accessibility** – **Callback**. This option is now the default.
- The Layout Editor displays the layout grid in the current figure color.
- The Layout Editor context menus have been reorganized.
- The Menu Editor enables you to rearrange the order of menu items.
- The Menu Editor adds callback function stubs to the application M-file.

See *Creating Graphical User Interfaces* in the MATLAB documentation for more information.

Major Bug Fixes

MATLAB 6.1 includes several bug fixes made since MATLAB 6.0. This section describes the particularly important bug fixes.

Development Environment

Help Browser Supports Mouse Wheel

For Windows platforms, the wheel on your mouse will now work in the Help browser.

UNIX Help Browser Search Results Now Highlighted

On UNIX systems, when you perform a full text search using the Help browser, the search terms are highlighted when you view a page.

UNIX Paste Problems Fixed

On some UNIX systems, pasting after a cut or copy would sometimes cause the system to hang. That problem has been fixed. However, due to issues with UNIX itself, the paste does not always work and you might have to do it again.

Mathematics

Memory Leak Fixed in Matrix Multiply

Under certain conditions, matrix multiply (which includes matrix-vector multiply, vector-matrix multiply, and even vector inner products) leaked memory. For example, on a Pentium III under Linux or Windows, any vector inner product of length greater than 15,000 leaked memory. This was observed by MATLAB increasing its use of system resources that were never returned. MATLAB 6.1 uses new ATLAS BLAS libraries that no longer leak memory.

Improved Convergence for `eigs(A,k,'sm')` and `eigs(A,k,0)`

In MATLAB 6.0, `eigs` was reimplemented to use the ARPACK library of routines. Unfortunately, the smallest magnitude case, `sigma = 'sm'` and `sigma = 0`, chose the wrong algorithm. For MATLAB 6.1, the correct ARPACK algorithm is used and convergence is much quicker.

This bug fix introduces a backwards incompatibility. When A is a function $Afun$ and $\sigma = 'sm'$, $Afun$ must now return $Y = A \setminus x$. Prior to MATLAB 6.1, `eigs` required $Afun$ to return $y = A * x$ for this case.

quad Sampling Improved

In MATLAB 6.0, `quad('cos(4*n*x)', -pi, pi)` returned $2 * \pi$ instead of 0. When `quad` initially sampled the function, it incorrectly assumed the function is the constant 1 over the interval $[-\pi, \pi]$ and so returned $2 * \pi$ early. It now samples more carefully and returns 0.

griddata3 Inner Matrix Error Message

In MATLAB 6.0, an internal error sometimes caused `griddata3` to display the error message, `Inner matrix dimensions must agree`. This error has been corrected.

Improved Handling of Degenerate Triangulation

In MATLAB 6.0, there were sometimes problems associated with degenerate triangulation. For example, `convhull` could produce a convex hull that did not cover all the original data. MATLAB 6.1 corrects this problem by replacing the utility function `delaunayc` with `Qhull`.

Error Message Display for Qhull-Related Functions

In MATLAB 6.0, `Qhull`-related functions (e.g., `delaunayn`) displayed error messages in standard error. For UNIX platforms, standard error is different from the command window. For MATLAB 6.1, error messages are displayed in the command window.

histc Computes First Two Bins Correctly

Prior to MATLAB 6.0, `histc` produced the wrong results for the first two bins for cases with extremely nonuniform bin edges. This problem was corrected in MATLAB 6.0.

```
Processor has been on-line since 04/20/2001 14:09:31
The alpha EV4.5 (21064) processor operates at 233 MHz,
and has an alpha internal floating-point processor.
```

The number in parentheses on the third line, in above example (21064), is the number you are interested in.

GLNX86. Enter the following command

```
cat /proc/cpuinfo
```

and look for the following fields in the output (values may vary from the example below)

```
vendor_id   :GenuineIntel
cpu family  :6
model       :8
model name  :Pentium III (Coppermine)
stepping    :1
```

Match up this information with the table in <MATLAB>\bin\glx86\blas.spec.

Note Some versions of glibc 2.1.x have problems with environment variables (and the ability to reliably query them) from within shared library init functions. To take advantage of the BLAS_VERSION feature, you may need to upgrade your machine to glibc 2.2.

HP700. Start with the **System Administration Manager (SAM)** and work your way to the **Processor** tab, as shown below:

System Administrator Manager (SAM) -> Performance monitors -> System properties -> Processor tab

This provides information about the type of processor.

HPUX. MATLAB only supports HPUX running on PA-RISC2.0.

IBM_RS. Contact IBM Technical Support and request the document entitled "Determining CPU Speed in AIX." This is a table of machine types, processor types, and processor speeds.

SGI. Enter the following command

```
sysinfo -a
```

which returns a lot of information. In the first few lines, look for information something like

```
CPU Type is                mips R4400 5.0
```

The information starting with R is what you are interested in. MATLAB ships for the R5000, R8000, R10000 and R12000 (default).

SOL2. Enter the following command

```
uname -m
```

which returns either sun4u for UltraSPARC or sun4m for the older, non-Ultra machines (e.g., Hyper and SuperSPARCs).

WIN32. Start with the **My Computer** icon, and work your way to the **General** tab, as shown below:

My Computer -> Control Panel -> System -> General tab

This should list the family and model number for your computer. On Windows NT and Windows 2000, the same information is on the **Environment** tab, under the System Variable PROCESSOR_IDENTIFIER. Match up this information with the table in <MATLAB>\bin\win32\blas.spec.

Using Another BLAS

You may also use BLAS from other sources than the ones shipped with MATLAB, provided they are in the correct format. This format is a shared library (as opposed to a static library) that exports all the double-precision (starting with d) and double-precision complex (starting with z) BLAS routines from dasum to zupmtr. On HP, IBM_RS, and WIN32, the symbols must be exported without trailing underscores, while for ALPHA, GLNX86, SGI, and SOL2, the symbols must be exported with trailing underscores (e.g., dgemm_).

If the shared library you provide also includes LAPACK symbols like dgefa (or dgefa_), then they will override the MATLAB default implementation, which is based on the Fortran LAPACK from Netlib at <http://netlib.org>.

Upgrading from an Earlier Release

This section describes the upgrade issues involved in moving from MATLAB 6.0 (Release 12.0) to MATLAB 6.1 (Release 12.1). This section about upgrading from an earlier release is organized into the following subsections:

- “Development Environment Issues” on page 8-23
- “Mathematics Issues” on page 8-24
- “Programming and Data Types Issues” on page 8-26
- “Graphics Issue” on page 8-27
- “External Interfaces/API Issues” on page 8-27

Development Environment Issues

subscribe Function No Longer Supported

The subscribe function is no longer supported.

Command History, Preferences, and Favorites

If you uninstall Release 12.0, you will lose the Command History, preferences, and Help browser favorites from Release 12.0.

To keep these files for use in Release 12.1, make a copy of them before uninstalling Release 12. To see where the files are located, run `prefdir` in the Command Window. The relevant files are listed below.

| Filename | File For |
|------------------------------|------------------------|
| <code>cwdhistory.m</code> | Command Window history |
| <code>history.m</code> | Command History |
| <code>matlab.prf</code> | Preferences |
| <code>matlab_help.hst</code> | Help browser favorites |

After uninstalling Release 12, put your backup copy of the files in the location returned by `prefdir` so that Release 12.1 can use the files.

Help Browser Favorites

If you use favorites you created for the documentation in the Release 12.0 Help browser, those favorites may point to an incorrect or invalid location in Release 12.1. You will need to delete any invalid favorites and add those favorites again.

Source Control

If you use Microsoft Visual SourceSafe with the MATLAB source control features, you now need to specify the login information for SourceSafe using preferences. Select **File -> Preferences -> General -> Source Control** from the desktop. Specify the **Username**, **Password**, and **Database**.

Mathematics Issues

Finding Smallest Magnitude Eigenvalues

`eigs(A,k,sigma)` and `eigs(A,B,k,sigma)` return k eigenvalues based on σ . For $\sigma = 'sm'$, `eigs` returns the smallest magnitude eigenvalues.

In MATLAB 6.0, `eigs` was reimplemented to use the ARPACK library of routines. Unfortunately, the smallest magnitude case, $\sigma = 'sm'$ and $\sigma = 0$, chose the wrong algorithm. For MATLAB 6.1, the correct ARPACK algorithm is used and convergence is much quicker.

This bug fix introduces a backwards incompatibility. When A is a function `Afun` and $\sigma = 'sm'$, `Afun` must now return $Y = A \setminus x$. Prior to MATLAB 6.1, `eigs` required `Afun` to return $y = A * x$ for this case.

Possible Changes in Results Returned by Matrix Functions

Starting in MATLAB 6.0 (R12.0), matrix computations are based on LAPACK, a large, multiauthor Fortran subroutine library for numerical linear algebra. While this change has many benefits and matrix functions continue to operate in the same way in MATLAB 6.0, the results returned by matrix functions may differ. Changes in roundoff errors can be seen in most matrix computations. In cases where quantities are not uniquely determined mathematically, results may differ in order and in normalization.

For example:

- Eigenvalues may be returned in a different order.

- Eigenvectors may be normalized differently.
- The signs of columns of orthogonal matrices may differ.
- `rcond` is a better estimate of the reciprocal condition.
- `lu` can now be used to factor rectangular full matrices.

Obsolete Input Arguments

Certain input arguments to these functions have become obsolete. Using these arguments does not result in an error, but they are ignored.

| Function | Description |
|-----------------------|---|
| <code>delaunay</code> | Now ignores the third argument <code>fuzz</code> , which specified a value for the <code>fuzz</code> standard deviation. Now ignores the third argument <code>'sorted'</code> . This argument indicated to <code>delaunay</code> that the given points <code>x</code> and <code>y</code> were sorted, and that duplicate points had been eliminated. |
| <code>convhull</code> | Now ignores the third argument <code>TRI</code> , which provided triangulation data previously computed using <code>delaunay</code> . |

Obsolete Functions

The following MATLAB function has become obsolete. For backwards compatibility, it has not been removed from the language at this time. However, this function may be removed in a future release, and you are encouraged to discontinue its use, or use the function that replaces it

| Function | Description |
|---------------------|--|
| <code>bvpval</code> | Evaluate the numerical solution of a boundary value problem (BVP). Replace with <code>deval</code> , which evaluates the solution of both initial value and boundary value differential equation problems. |

Programming and Data Types Issues

Output from Background and Foreground Commands (UNIX)

In Release 12 on UNIX platforms, a background command (i.e., any system command after which you add a &), such as

```
! cat startup.m &
```

no longer produces any output. Prior to Release 12, a background command sent output to the command window.

If you need to see the output from a command, either do not make the command a background command (i.e., remove the &), or run the background command in a separate xterm. To start another xterm, issue the following command.

```
! xterm &
```

In Release 12, foreground functions (i.e., nonbackground functions) send their output to the diary, if the diary function has been issued. The output is also displayed in the command window (prior to Release 12, foreground function output was only displayed in the command window).

matlab_helper Process

To make the ! and unix commands operate more efficiently, in Release 12 MATLAB creates a secondary process, called matlab_helper, at startup.

This matlab_helper contains those elements of MATLAB necessary to run the ! and unix commands.

Graphics Issue

MATLAB No Longer Supports Terminal Mode

MATLAB no longer runs on nongraphics computer terminals.

External Interfaces/API Issues

Concatenation of Java Objects

When you concatenate Java objects, the class of the resultant object depends on the classes of the input objects, as follows:

- If the input objects are of the same class, MATLAB makes the output object of that class. This was true in Version 6.0 as well.
- If the input objects are of different classes, but all inherit from a common class, MATLAB makes the output object of the common parent class. MATLAB selects the lowest common parent in the Java class hierarchy as the output class. This is new behavior for Version 6.1.

For example, concatenating objects of classes `java.lang.Integer` and `java.lang.Double` creates a new object of class `java.lang.Number`.

- If the input objects are of different and unrelated classes, then MATLAB makes the output object of the `java.lang.Object` class. This was true in Version 6.0 as well.

Obsolete Fortran MX, MEX, MAT, and ENG Functions

The following Fortran MX, MEX, MAT, and ENG functions are considered to be obsolete as of Version 6.1. Support for these functions may be removed from a future MATLAB release.

Table 8-5: Obsolete Fortran MX Functions

| | |
|---------------------------|---------------------------|
| <code>mxCreateFull</code> | <code>mxFreeMatrix</code> |
| <code>mxIsFull</code> | <code>mxIsString</code> |

Table 8-6: Obsolete Fortran MEX Functions

| | |
|--------------|-----------------|
| mexGetEps | mexGetGlobal |
| mexGetFull | mexGetInf |
| mexGetMatrix | mexGetMatrixPtr |
| mexGetNaN | mexIsFinite |
| mexIsInf | mexIsNaN |
| mexPutFull | mexPutMatrix |

Table 8-7: Obsolete Fortran MAT Functions

| | |
|-----------------|------------------|
| matDeleteMatrix | matGetFull |
| matGetMatrix | matGetNextMatrix |
| matGetString | matPutFull |
| matPutMatrix | matPutString |

Table 8-8: Obsolete Fortran Engine Functions

| | |
|------------|--------------|
| engGetFull | engGetMatrix |
| engPutFull | engPutMatrix |

Known Software and Documentation Problems

This section updates the MATLAB 6.1 documentation set, reflecting known MATLAB 6.1 software and documentation problems. It is organized into the following subsections:

- “Development Environment Problems” on page 8-29
- “Documentation Updates” on page 8-30

Development Environment Problems

Displaying Results From lookfor Function

When you run the lookfor function, press **Ctrl+C** to display the results in the command window.

Cannot Go To Top Level of UNC Path

For Windows platforms, you cannot use `cd` or any directory tool in the MATLAB desktop (including the Current Directory browser and Set Path dialog box) to access the top level of a UNC path.

Workspace Browser with Many Variables

If there are many variables in the workspace and the Workspace browser is open, you may experience performance problems. If you expect to have more than 1000 variables in the workspace, close the Workspace browser to avoid performance problems.

UNIX Display Problems When UNIX Client and Server Platforms Differ

If you use MATLAB on UNIX and the platform for the server is different than that for the client, there may be problems with the display of graphics on the client. See the Technical Support Web page for a solution that lists the combinations tested and any known display problems with them.

Sun Solaris 16-Bit Display Not Supported

Sun's Java VM for Solaris does not support 16-bit displays. Therefore you cannot use this configuration with Release 12. Use another display mode instead.

Sun Solaris Arrow Keys Not Working

On some Sun Solaris systems, the arrow keys on the main keyboard are not working properly. Instead, try the arrow keys in the numeric keypad.

Alpha Shortcut Problems When Using Emacs Key Bindings in Editor

On the Alpha platform, if you set the Editor/Debugger preference for key bindings to Emacs, the shortcuts for **Undo** (**Ctrl+_**) and **Copy** (**~+W**) do not work.

Display Problems with Xoftware

If you use Xoftware on a PC to run MATLAB on a UNIX platform, you need to do the following to avoid display problems:

- 1** Go to the Xoftware **Control Panel**.
- 2** From the **Options** menu, select **Configuration**.
- 3** Select the **Window** tab.
- 4** From the **Options** listing, select **Concurrent Window Manager**.
- 5** Under **Settings**, select **Off**.
- 6** Click **OK**.

Documentation Updates

Editor/Debugger Example - Graphic and Information Incorrect

In the printed book *Using MATLAB* (Version 6), on page 7-19, the graph shown is incorrect. For the correct graph, see the same page in the Help browser at **MATLAB -> Using MATLAB -> Development Environment -> Editing and Debugging M-Files -> Debugging M-Files -> Trial Run for Example**.

On page 7-29, in “Correcting Problems and Ending Debugging, Completing the Example,” step 3 is incorrect. It should instead read “In `collatzplot.m` line 12, change the string `plot_seq` to `seq_length(m)` and save the file.”

interp1 Extrapolation of Out-of-Range Values

A new argument enables `interp1` to perform extrapolation for out-of-range values for all methods. It also enables you to specify a scalar to be returned for out-of-range values.

The PDF version of the `interp1` reference page incorrectly states that the default for all methods is for `interp1` to perform extrapolation for out-of-range values. In fact, `interp1` performs extrapolation as the default only for the 'spline', 'pchip', and 'cubic' methods. For all other methods, it returns NaN for out-of-range values. This behavior is unchanged from Version 5.

The HTML reference page for `interp1` is correct.

